Introduction to Logic Synthesis with VHDL

VHDL

Hardware Description Languages (HDLs) are used to design and simulate digital circuits. In this course we will use VHDL[1] rather than the other popular HDL, Verilog. This lecture covers a subset of VHDL that is sufficient for designing circuits ("logic synthesis"). Other language features, namely processes and sequential statements, are required for simulating circuits.

Signal Assignment

The first example is a circuit called a **both** that has one output signal (**c**) that is the AND of two input signals (**a** and **b**). The file **both.vhd** contains the following VHDL description:

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

-- 'both' : An AND gate

entity both is port (
   a, b: in std_logic;
   c: out std_logic ) ;
end both ;

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

Note the following about VHDL syntax:

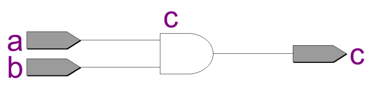
- VHDL is case-insensitive. There are many capitalization styles. I prefer all lower-case. You may use another style as long as you are consistent.
- Everything following two dashes "**--**" on a line is a comment and is ignored.
- Statements can be split across lines and end with a semicolon. Indentation styles vary but an **end** should be indented the same as its corresponding **begin**.
- semicolons separate (not terminate) the port declarations
- entity and signal names begin with a letter followed by letters, digits or underscore ("_") characters.
- The **library** and **use** statements make available some commonly-used types and functions.

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*). This is the difference between HDLs and conventional programming languages – HDLs allows us to specify the concurrent behaviour of hardware.

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 **std_logic** can use various operators including **and**, **or**, **xor** and **not**. **not** has higher precedence than the other logical operators, all of which have equal precedence. Parentheses can be used to force evaluation in a certain order.

From this VHDL 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:



Exercise 1: What changes would result in a 3-input OR gate?

Conditional Assignment

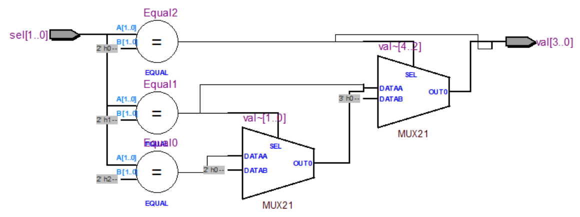
Conditional assignments model multiplexers. For example:

[1]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.

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

entity dec4 is
   port (
   sel : in std_logic_vector (1 downto 0) ;
   val : out std_logic_vector (3 downto 0)) ;
end dec4 ;

architecture rtl of dec4 is
begin
   val <=
      "0001" when sel = "00" else
      "0010" when sel = "01" else
      "0100" when sel = "10" else
      "1000" ;

end rtl ;
```

is a decoder which synthesizes to:



Note:

- Arrays model buses. The array declaration specifies the range of possible index values. Each index value corresponds to one bit of the bus. The values of the indices of `std_logic_vector`s are typically declared to have decreasing (`downto`) values so that constants read left-to-right in the conventional order.
- Conditions are tested in order and the first true one determines the value assigned.
- There is an unconditional assignment at the end which is needed to avoid creating a latch.

`std_logic_vector` literals are formed by enclosing an ordered sequence of binary or hexadecimal values in double quotes after a leading B (optional) or X respectively. For example, `B"1010_0101"` and `X"A5"` are equivalent. `std_logic` literals are `'0'` and `'1'`.

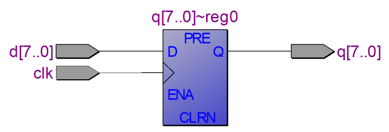
Registers

If none of the conditions of a conditional assignment are true then the assignment should not happen and the assigned value should not change. This models transparent latches. The function `rising_edge()` is true on the rising edge of a signal and can be used to model edge-trigerred registers.

For example,

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

-- 8-bit edge-triggered register

entity reg8 is port (
   d: in std_logic_vector (7 downto 0) ;
   clk: in std_logic ;
   q : out std_logic_vector (7 downto 0) ) ;
end reg8 ;

architecture rtl of reg8 is
begin
  q <= d when rising_edge(clk) ;
end rtl ;
```

synthesizes to:

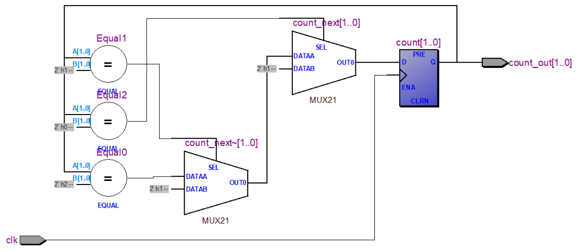


Exercise 2: How could you create a transparent latch?

State Machines

Conditional assignments can be used to model state machines. One conditional assignment computes the next state from the current state and aother implement the state register. For example, a 2-bit counter:

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

entity count2 is
   port (
   clk : in std_logic ;
   count_out : out std_logic_vector (1 downto 0)) ;
end count2 ;

architecture rtl of count2 is
   signal count, count_next : std_logic_vector(1 downto 0) ;
begin
   count_next <=
      "01" when count = "00" else
      "10" when count = "01" else
      "11" when count = "10" else
      "00" ;

    count <= count_next when rising_edge(clk) ;

    -- due to bug in Quartus 13.0sp1
    count_out <= count ;

end rtl ;
```

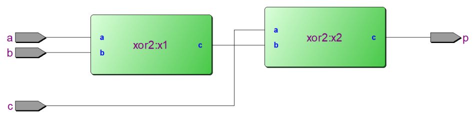


Component Instantiation

A component can be instantiated (included) in an architecture.

The following example shows how two 2-input exclusive-or gates can be used to build a 3-input parity-check circuit. This type of description is called *structural* and is the HDL equivalent of a schematic.

```
library ieee ;
use ieee.std_logic_1164.all ;
use work.mypkg.xor2 ;

entity parity is
   port ( a, b, c : in std_logic ; p : out std_logic ) ;
end parity ;

architecture rtl of parity is
   -- internal signals
   signal x : std_logic  ;
begin
   x1: xor2 port map ( a, b, x ) ;
   x2: xor2 port map ( c, x, p ) ;
end rtl ;
```



- the `use work.mypkg.xor2` makes available the `xor2` component declaration from the `mypkg` library
- the `port map` clause is used to "map" (connect) the component's ports to signals in the architecture.

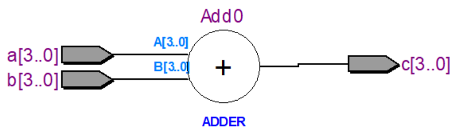
Arithmetic

The IEEE `numeric_std` package defines defines the `signed` and `unsigned` types to which the usual arithmetic operators (`+`, `-`, `*`, `/`, `**`, `>`, `<`, `<=`, `>=`, `=`, `/=`) can be applied. However, it may not be possible to synthesize complex operators.

For example, the following 4-bit adder:

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

entity adder4 is
   port (
   a, b : in unsigned (3 downto 0) ;
   c : out unsigned (3 downto 0) ) ;
end adder4 ;

architecture rtl of adder4 is
begin
   c <= a + b ;
end rtl ;
```

results in:



`std_logic` signal can also have undefined ('X', useful for simulation), high-impedance values ('Z' useful for implementing tri-state outputs)[3].

Exercise 3: Rewrite the state machine example using **`unsigned`** signals.

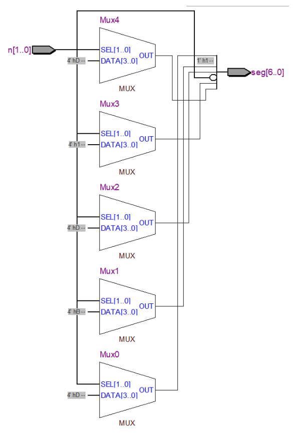
Selected Assignment

The selected assignment statement models a multiplexer whose control value selects an input.

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

-- 7-segment LED driver for 2-bit input values

entity led7 is port (
   n: in std_logic_vector (1 downto 0) ;
   seg: out std_logic_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:

[3]Don't care, '-', can sometimes be used for synthesis.



Note:

- the select value is compared to all values simultaneously.
- the keyword **`others`** indicates the default value to assign when none of the other values matches the selection expression. *Always include an* **`others`** *clause.*
- commas separate the clauses

Vectors

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

Exercise 4: If **`x`** is declared as **`bit_vector (3 downto 3)`** and in an architecture the assignment **`x<="0011"`** is made, what is the value of **`x(3)`**?

The **`rol`**, **`ror`**, **`sll`**, **`srl`**, **`sla`** and **`sra`**, operators rotate and left/right arithmetic (sign bit preserved)/logical shift vectors.

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

Exercise 5: Write a VHDL description that uses '&' to assign **`y`** a bit-reversed version of a 4-bit vector **`x`**.