6.111 Lecture # 10

Topics for today:

Some more details of VHDL and more examples
Shift Register (as in the 74LS194)

Note Lab 2 design should be done by Wednesday

But first,... clock Conventions
This is only a convention but it is widely used. What is important is when devices are triggered.

Positive Edge Triggered devices
Often call it /CLK because setup is when the clock signal is LOW
Most registers are like this

Negative Edge Triggered devices
Often call it CLK because setup is when the clock signal is HIGH
J-K flip flops tend to be like this

VHDL Identifiers

Case InsenSitivE (but best not to rely on this)
First character must be a letter.

Letters, Digits, and Underscores (only)
Two underscores in succession are not allowed.
The last character cannot be an underscore.

Using reserved words is NOT allowed.

Later versions of emacs use color to distinguish reserved words (and other things)
Using reserved words usually provokes an understandable error comment.

Legal Examples
CLK, Three_StateEnable, h23, Reg_12

Illegal Examples
_clk, _State_Enable, large#num, clk_, Three__State, register, begin

VHDL Reserved Words

Some are

abs access after begin
array disconnect file
guarded impure postponed
rem unaffected wait

There are 97: too many to remember!
This is another good reason for “incremental” compilation.

Start with something that compiles and add code a block at a time
VHDL Values: Defined in IEEE 1164.

Values you are most likely to use are '0', '1', '-' (hyphen) is 'don't care'
'Z' (MUST Be upper case) is 'High Impedance'

Vectors are strings
Remember VHDL is strongly typed:
a+b is valid ONLY if a and b have the same length
To assign to a one bit longer number (as in to accommodate overflow)

\[ c \leftarrow ('0' \& a) + ('0' \& b) \]
and of course c must be defined to be one bit longer than a and b

Designation of constants:
'-' is a character
"---" is a string (vector) of length 3
& is the concatenation operator:
"01" & "111" is "01111" and so is '0' & "111"

Packages
Here is a very small package construction

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

ework.std_arith.all; -- needed for integer + signal

topology test_tri is
  port(clk, oe, cnt_enb : in std_logic;
  data : inout std_logic_vector(7 downto 0));
end test_tri;

architecture foo of test_tri is
  signal counter : std_logic_vector(7 downto 0);
  begin
    process (oe, counter)
      begin
        if (oe = '1') then data <= counter;
        else
          data <= "ZZZZZZZ"; -- N.B. Z must be UPPERCASE!
        end if;
      end process;

    process (clk)
      begin
        if rising_edge(clk) then
          if (oe = '0') and (cnt_enb = '1') then
            counter <= counter + 1;
          end if;
        end if;
      end process;
    end architecture foo;
```

Now we can use that package in some top level code:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity toplevel is port (
  s: in std_logic;
  p, q, r: in std_logic_vector(2 downto 0);
  t: out std_logic_vector(2 downto 0));
end toplevel;

architecture archtoplevel of toplevel is
  signal i: std_logic_vector(2 downto 0);
  begin
    -- the first two instantiations are named associations
    m0: mux2to1 port map (a=>i(2), b=>r(0), sel=>s, c=>t(0));
    m1: mux2to1 port map (c=>t(1), b=>r(1), a=>i(1), sel=>s);
    -- the last instantiation is a positional association
    m2: mux2to1 port map (i(0), r(2), s, t(2));
    i <= p and not q;
  end archtoplevel;
```

Here is the use of `inout` (tristate)
Predefined Attributes

`s'event` is read as "s tick event" where s is a signal name.

`rising_edge(event)` is the same as

`s'event and event = '1'`)

A transaction occurs every time a signal is evaluated, whether or not the signal value changes.

Evaluation of one signal can force evaluation of other signals

Array Attributes are particularly useful with generic array sizes

```vhdl
signal s : std_logic_vector(7 downto 3)
  s'left = 7  s'high = 7
  s'right = 3  s'low = 3
  s'length = 5
```

You can even build multiply indexed arrays:

```vhdl
type rom is array (0 to 6, 3 down to 0) of std_logic;
signal r : rom;
  r'left(1) = 0  r'high(1) = 6
  r'left(2) = 3  r'high(2) = 3
  r'right(1) = 6  r'low(1) = 0
  r'right(2) = 0  r'low(2) = 0
  r'length(1) = 7
  r'length(2) = 4
```

74LS194: Bidirectional, loadable shift register

```vhdl
--variable width shift register (like a '194)
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity shift_reg is
  generic (width : integer := 4); -- to start
  port (data : in std_logic_vector(width-1 downto 0); -- input
        s: in std_logic_vector(1 downto 0); -- shift bits
        clk, sl, sr : in std_logic; -- shift bits
        output : out std_logic_vector (width-1 downto 0));
end shift_reg;
```

The part also has an asynchronous clear

So now we are going to write the functionality of this part in VHDL

The use of positional attributes makes this variable width work
-- purpose: simulation of a '194 shift register
architecture first_try of shift_reg is
  signal int : std_logic_vector(width-1 downto 0); -- used internally
  constant right  : std_logic_vector(1 downto 0) := "01";
  constant left   : std_logic_vector(1 downto 0) := "10";
  constant load   : std_logic_vector(1 downto 0) := "11";
  constant hold   : std_logic_vector(1 downto 0) := "00";
begin -- first_try
  output <= int;
  shift_reg: process(clk)
  begin
    if rising_edge(clk) then
      case s is
        when right =>
          int <= sr & int(int'left downto int'right+1);
        when left =>
          int <= int(int'left-1 downto int'right) & sl;
        when load =>
          int <= data;
        when hold =>
          int <= int;
        when others =>
          int <= (others => '-');
      end case;
    end if;
  end process;
end first_try;

Two More attributes that Are Useful Sometimes

Sum splitting occurs when more than 16 product terms are required.
(This depends, of course, on what part you are compiling to)

balanced (default) has better timing but uses more macrocells.
cascaded uses fewer macrocells and is slower.

attribute sum_split of mysig: signal is cascaded;
attribute sum_split of mysig: signal is balanced;

The synthesis_off attribute is used to make the signal a factoring point.

Making a signal a factoring point can result in a reduction of product terms
for a subsequent signal. It also avoids the possibility that a signal can be optimized
away.

Registered equations are natural factoring points so only use synthesis_off on
combinational signals.

attribute synthesis_off of sel: signal is true;

--- attempt at short pulse catcher
library ieee;
use ieee.std_logic_1164.all;
entity spulse is port(
  N_GO, CLK, S_CLK: in std_logic;
  aout, n_aout, xout, p: out std_logic);
end spulse;
architecture behavioral of spulse is
signal A , N_A, X, N_X, N_CLK : std_logic;
begin
  A <= (not N_GO) or (not N_A);
  N_A <= (not A) or (not N_X);
  N_X <= (not X);
  P <= X and N_CLK;
  N_CLK <= not (S_CLK);
  aout <= A;
  xout <= X;
  n_aout <= N_A;
ff: process(CLK)
begin -- process
  if rising_edge(CLK) then
    X <= A;
  end if;
end process;
end behavioral;
So here is what gets synthesized:

With attribute synthesis_off

DESIGN EQUATIONS
(12:40:06)

\[
\begin{align*}
p &= xout.Q \cdot /s_clk \\
xout.D &= aout \\
xout.C &= clk \\
/aout &= n_go \cdot n_aout \\
/n_aout &= /xout.Q \cdot aout
\end{align*}
\]

And without attribute synthesis_off

DESIGN EQUATIONS
(12:41:12)

\[
\begin{align*}
p &= xout.Q \cdot /s_clk \\
xout.D &= aout \\
xout.C &= clk \\
/n_aout &= aout \cdot /xout.Q \\
aout &= aout \cdot /xout.Q + /n_go
\end{align*}
\]