

## L11.1 VHDL Identifiers, Reserved Words Mon. March 4, 2002

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 as identifiers is NOT allowed.

Reserved words are AQUA in emacs.

Using reserved words usually provokes an understandable error comment.

Legal Examples are `CLK`, `Three_StateEnable`, `h23`, `Reg_12`

Illegal Examples are `_clk`, `3_State_Enable`, `large#num`, `clk_`, `Three__State`, `register`

Some reserved words are `abs`, `access`, `impure`, and `postponed`.

In other words, there are too many to remember!

To see what is a reserved word, notice the color in your editor.

This is another good reason for "incremental" compilation.

Start with something that compiles and add code a block at a time!

## L11.2 – L10.22 Don't Care, Ampersand Wed. February 27, 2002

Don't cares are represented by a `-` (hyphen).

`'_'` single quotes for a character

`"_"` double quotes for a string (vector)

(others => `'_'`) for something independent of length

`&` (ampersand) to concatenate strings or signals

`"01" & "111"` is the same as `"01111"`

`'0' & "1111"` is the same as `"01111"`

Remember, VHDL is strongly typed.

`a + b` is valid only if `a` and `b` are of the same length.

The result is of the SAME length as `a` (or `b`).

If you want it to be one bit longer, then use

`c <= ('0' & a) + ('0' & b)`

-- Of course, `c` must be defined to be

-- one bit longer than `a`.

## L11.3 – L10.23 Predefined Attributes Wed. February 27, 2002

`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'`)

as synthesis only worries about 1 and 0.

Transactions occur when a signal is evaluated, whether or not the signal value changes.

Time, if not specified, is in deltas.

`s'event` – true only if an event occurred in the current delta cycle

`s'active` – true only if a transaction occurred on `s`

`s'last_event` – time elapsed since the last event on `s`

`s'last_value` – value of `s` before the previous event on `s`

`s'last_active` – time elapsed since the previous transaction on `s`

## L11.4 – L10.24 Predefined Attributes Wed. February 27, 2002

Signal attributes that create a signal

Mainly used for simulation

`s'delayed[(time)]` – creates a signal the same as `s` but delayed by the specified time

`s'stable[(time)]` – a Boolean signal that is true if `s` had no events for the specified time

`s'quiet[(time)]` – a Boolean signal that is true if `s` had no transactions for the specified time

`s'transaction` – a signal of type `bit` that changes for every transaction on `s`

L11.5 – L10.25 Array Attributes Wed. February 27, 2002

Signal s : std\_logic\_vector( 7 downto 3)

```
s'left = 7      s'high = 7
s'right = 3     s'low = 3
s'length = 5
```

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
```

L11.6 – L10.26 User Defined Attributes Wed. February 27, 2002

Type state\_type is (idle, state1,state2);  
attribute state\_encoding of state\_type: is sequential;  
-- or one\_hot, zero\_hot, gray  
attribute enum\_encoding of state\_type: is "11 01 00";  
-- or whatever assignment you want to make  
-- within an entity  
attribute pin\_numbers of counter:Entity is  
"clk:13 reset:2 &  
" count(3):3;  
-- Note the space before count(3) above  
-- within an entity  
attribute pin\_avoid of mydesign: entity is "21 24 26";  
-- the following are not recommended  
attribute lab\_force of mysig: signal is a1;  
attribute node\_num of buried: signal is 202;  
attribute low\_power of mydesign: entity is "b g e";  
attribute slew\_rate of count(3): signal is slow; -- or fast

L11.7 – L10.27 Sometimes Useful Wed. February 27, 2002

Sum splitting occurs because of too many product terms.

Balanced (default) has better timing but uses more macrocells.  
Cascaded uses fewer macrocells and is slower.

attribute sum\_split of mysig: signal is cascaded;

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.

Registered equations are natural factoring points so only use synthesis\_off on combinational signals.

attribute synthesis\_off of sel: signal is true;

L11.8 Which Way Is Better? Mon. March 4, 2002

```
-- Different methods of expressing combinational logic
library ieee;
use ieee.std_logic_1164.all;
entity comb is
  port(ld1, ld2, oe1, oe2 : in std_logic;
        in1 : in std_logic_vector(1 downto 0);
        in2 : in std_logic_vector(1 downto 0);
        data1 : inout std_logic_vector(1 downto 0);
        data2 : inout std_logic_vector(1 downto 0));
end comb;
architecture archcomb of comb is
begin
  method1: process(oe1, in1)
  begin
    if oe1 = '1' then data1 <= in1;
    else data1 <= "Z1"; -- N.B. Z must be UPPERCASE!
    end if;
  end process method1;
  -- method2
  data2 <= in2 when oe2 = '1' else "Z1";
end architecture archcomb;
-- The equations generated are the same!
-- data1_1 = in1_1      data1_1.OE = oe1
-- data2_1 = in2_1      data2_1.OE = oe2
-- /data1_0 = oe1 * /in1_0 or data1_0 = /oe1 + in1_0
-- /data2_0 = oe2 * /in2_0 or data2_0 = /oe2 + in2_0
```

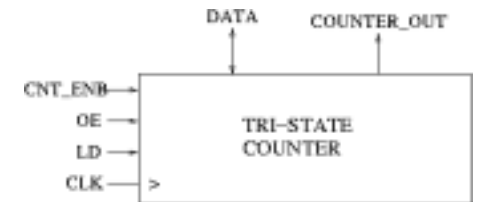
L11.9 Tri-State to Multiplex I/O pins Mon. March 4, 2002

```
-- Use tri-state logic to multiplex IO pins.
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all; -- needed for integer + signal
entity ldcnt is
    generic (width : integer := 3);
    port(clk, ld, oe, cnt_enb : in std_logic;
        -- Use counter_out_out only for simulation.
        counter_out : out std_logic_vector(width - 1 downto 0);
        data : inout std_logic_vector(width - 1 downto 0));
end ldcnt;
```



L11.10 Tri-State Architecture Mon. March 4, 2002

```
-- purpose: count with an output enable
architecture archldcnt of ldcnt is
    signal counter : std_logic_vector(width - 1 downto 0);
begin
    counter_out <= counter;
    data <= counter when oe = '1' else (others => 'Z');
    -- N.B. Z must be UPPERCASE!
    cnt: process(clk)
    begin
        if rising_edge(clk) then
            if ld = '1' and oe = '0' then
                counter <= data;
            elsif cnt_enb = '1' then
                counter <= counter + 1;
            end if;
        end if; -- rising_edge(clk)
    end process cnt;
end architecture archldcnt;
```



L11.11 Simulation of Tri-State Ctr Mon. March 4, 2002

Note that data(2 downto 0) are white (meaning an input) when oe is low.

Also that counter doesn't load unless oe is low.



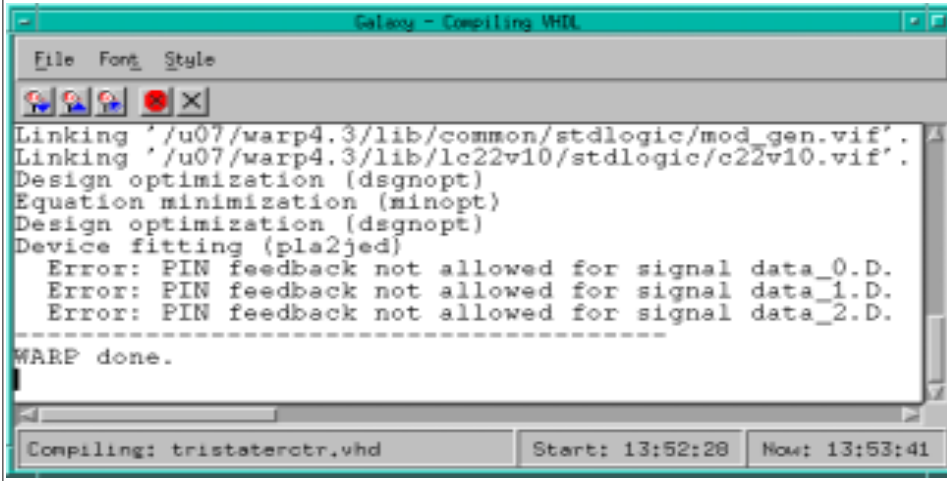
L11.12 Without Counter\_Out Mon. March 4, 2002

One has to poke around a bit to get the right jed nodes.  
 It is better to use counter\_out (if you can).



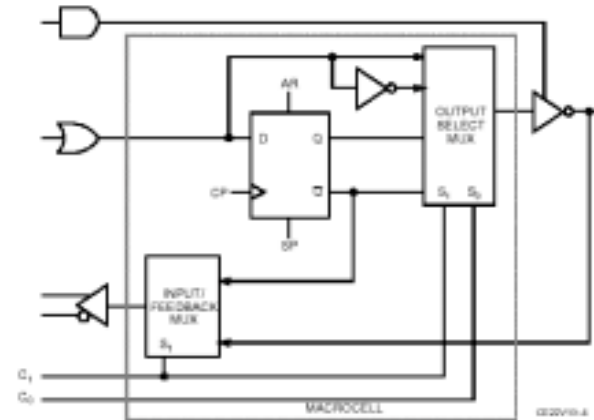
L11.13 Without Counter\_Out for a 22v10 Mon. March 4, 2002

The tri-state counter works without counter\_out, but you get this error if you try to use a 22v10. Why is this?



L11.14 – L4.14 22V10 Macrocell Wed. February 13, 2002

Flexible ‘Macrocells’  
 10 Macrocells, varying number of product terms  
 The clock is still from pin 1.



L11.16 What Did It Take? Mon. March 4, 2002

```
C2 = cin * a1 * a0 * /b1 * /b0 * /MODULE_2_g1_a0_g0_u0_ga_g1_uao
+ cin * /a1 * a0 * b1 * /b0 * /MODULE_2_g1_a0_g0_u0_ga_g1_uao
+ cin * a1 * /a0 * /b1 * b0 * /MODULE_2_g1_a0_g0_u0_ga_g1_uao
+ cin * /a1 * /a0 * b1 * b0 * /MODULE_2_g1_a0_g0_u0_ga_g1_uao
+ /a0 * /b0 * MODULE_2_g1_a0_g0_u0_ga_g1_uao
+ a0 * b0 * MODULE_2_g1_a0_g0_u0_ga_g1_uao
+ /a1 * /b1 * MODULE_2_g1_a0_g0_u0_ga_g1_uao
+ a1 * b1 * MODULE_2_g1_a0_g0_u0_ga_g1_uao
+ /cin * MODULE_2_g1_a0_g0_u0_ga_g1_uao
```

This is not really for reading, but look at the rather large number of product terms and the cascading of c<sub>2</sub>.

```
c1 = a1 * /a0 * /b1 * /b0
+ /cin * a1 * /b1 * /b0
+ /a1 * /a0 * b1 * /b0
+ /cin * /a1 * b1 * /b0
+ /a1 * a0 * /b1 * b0
+ cin * /a1 * /b1 * b0
+ a1 * a0 * b1 * b0
+ cin * a1 * b1 * b0
+ /cin * a1 * /a0 * /b1
+ cin * /a1 * a0 * /b1
+ /cin * /a1 * /a0 * b1
+ cin * a1 * a0 * b1
```

```
c0 = cin * /a0 * /b0
+ /cin * a0 * /b0
+ /cin * /a0 * b0
+ cin * a0 * b0

MODULE_2_g1_a0_g0_u0_ga_g1_uao =
a0 * b1 * b0
+ a1 * a0 * b0
+ a1 * b1
```

L11.15 A Simple Adder Mon. March 4, 2002

```
Library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all; -- needed for integer + signal
entity twoadd is
port(cin : in std_logic;
a, b : in std_logic_vector(1 downto 0);
c : out std_logic_vector(2 downto 0));
end twoadd;

architecture toomany of twoadd is
signal a_int, b_int : std_logic_vector(2 downto 0);
begin
a_int <= '0' & a;
b_int <= '0' & b;
c <= a_int + b_int when cin = '0' else a_int + b_int + 1;
end architecture toomany;
```

L11.17 A Better Adder Mon. March 4, 2002

We create two internal signals that are two bits longer than the input signals by concatenating a zero on the left and cin on the right.  
 We then create the output signal by using the left four bits of the sum of these two internal signals.

```
-- oneadd.vhd
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all; -- needed for integer + signal
entity oneadd is
    port(cin : in std_logic;
         a, b : in std_logic_vector(1 downto 0);
         c : out std_logic_vector(2 downto 0));
end oneadd;
architecture justright of oneadd is
    signal a_int, b_int, c_int : std_logic_vector(3 downto 0);
begin
    a_int <= '0' & a & cin;
    b_int <= '0' & b & cin;
    c_int <= a_int + b_int;
    c <= c_int(3 downto 1);
end architecture justright;
```

L11.18 Best Mon. March 4, 2002

This one takes a lot fewer product terms.  
 But the cascading with a module is still there.

However, the efficiency of realization is good enough to fit in a 16v8.  
 The other realization required a 22v10 which has more product terms available.

```
c_2 = b_1 * MODULE_1_g1_a0_g0_u0_ga_g1_ua0
      + a_1 * MODULE_1_g1_a0_g0_u0_ga_g1_ua0
      + a_1 * b_1
/c_2 = /b_1 * /MODULE_1_g1_a0_g0_u0_ga_g1_ua0
      + /a_1 * /MODULE_1_g1_a0_g0_u0_ga_g1_ua0
      + /a_1 * /b_1

c_1 = a_1 * /b_1 * /MODULE_1_g1_a0_g0_u0_ga_g1_ua0
      + /a_1 * b_1 * /MODULE_1_g1_a0_g0_u0_ga_g1_ua0
      + /a_1 * /b_1 * MODULE_1_g1_a0_g0_u0_ga_g1_ua0
      + a_1 * b_1 * MODULE_1_g1_a0_g0_u0_ga_g1_ua0
/c_1 = /a_1 * /b_1 * /MODULE_1_g1_a0_g0_u0_ga_g1_ua0
      + a_1 * b_1 * /MODULE_1_g1_a0_g0_u0_ga_g1_ua0
      + a_1 * /b_1 * MODULE_1_g1_a0_g0_u0_ga_g1_ua0
      + /a_1 * b_1 * MODULE_1_g1_a0_g0_u0_ga_g1_ua0

c_0 = cin * /a_0 * /b_0
      + /cin * a_0 * /b_0
      + /cin * /a_0 * b_0
      + cin * a_0 * b_0
/c_0 = /cin * /a_0 * /b_0
      + cin * a_0 * /b_0
      + cin * /a_0 * b_0
      + /cin * a_0 * b_0

MODULE_1_g1_a0_g0_u0_ga_g1_ua0 =
    a_0 * b_0
    + cin * b_0
    + cin * a_0
/MODULE_1_g1_a0_g0_u0_ga_g1_ua0 =
    /a_0 * /b_0
    + /cin * /b_0
    + /cin * /a_0
```

L11.19 A More Realistic example Mon. March 4, 2002

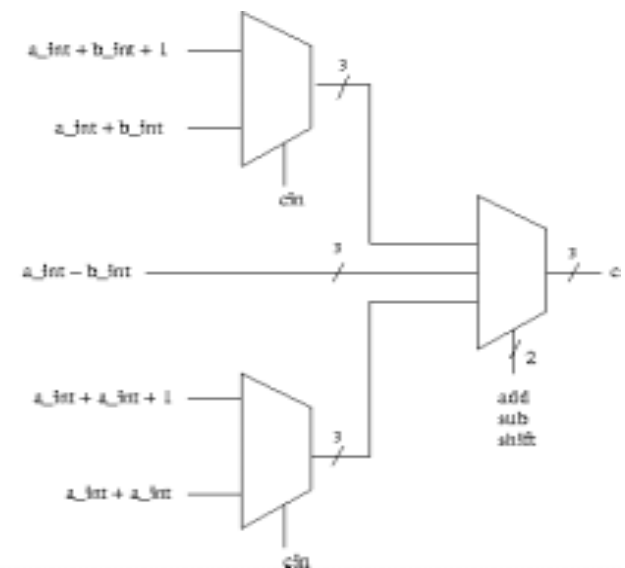
See the files in

</mit/6.111/vhdl/warp/implicit.adders/>

They implement an ALU which can add, sub, or shift left.

L11.20 Many Mon. March 4, 2002

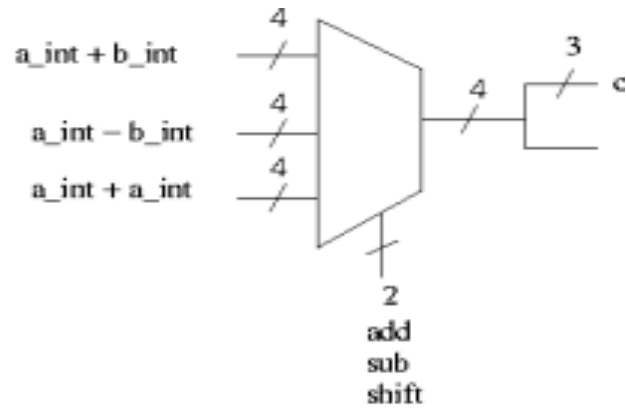
**MANY**



L11.21

Better

Mon. March 4, 2002

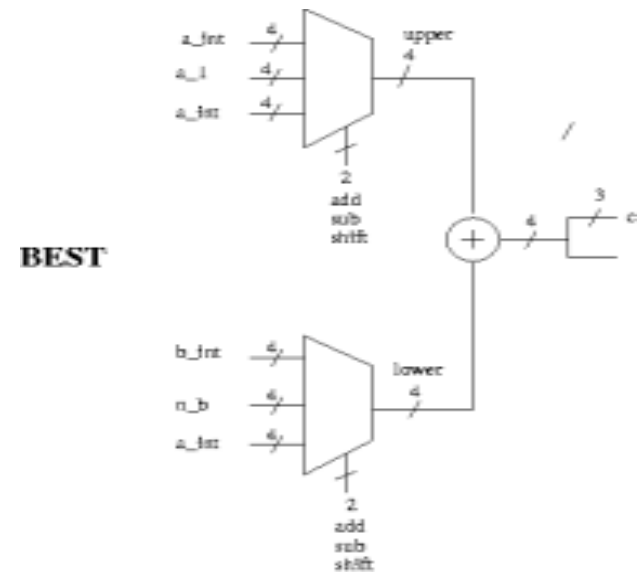


**BETTER**

L11.22

Best

Mon. March 4, 2002



**BEST**