



LUND
UNIVERSITY

EITF35: Introduction to Structured VLSI Design

VHDL Coding Style

Liang Liu
liang.liu@eit.lth.se



Comments "--": may be "more important"

```
-----  
-- Design : CARRIER SENSE  
-- File Name : CARRIER_SENSE.vhdl  
-- Purpose : Model of CARRIER SENSE process in PCS (IEEE Std 802.3)  
-- Limitation : none  
-- Errors : none known  
-- Include Files: none  
-- Author : Liang Liu, liang.liu@eit.lth.se, Lund University  
-- Simulator : ModelSim 6.5  
-----
```

-- Revision List:

Version	Author	Date	Changes
1.0	Liang Liu	2001/08/03	original created
1.1	Liang Liu	2002/01/04	disable TX_EN to CRS in repeater mode

***Make your code readable!!!
Maintenance and re-usability***



Names

Make your file name and signal name meaningful that others can read your code conveniently

- Suggestion for signal name: `direction_function_feature`

For example: ***`i_clk_r`***, means an input clock with rising edge trigger

- One entity per file and make the file name the same as entity name



Indent The Code Properly

```
architecture one_seg_arch of dff_en is
begin
  process (clk,reset)
  begin
    if (reset='1') then
      q <= '0' ;
    elsif (clk'event and clk='1') then
      if (en='1') then
        q <= d;
      end if;
    end if ;
  end process;
end one_seg_arch;
```

Use two space
instead of 'tab'
for indenting

some tools like vim and emacs treat 'tab' differently



Complete Sensitivity List

3-input and circuit

```
bad: process (a)
begin
    y <= a and b and c;
end process;
```

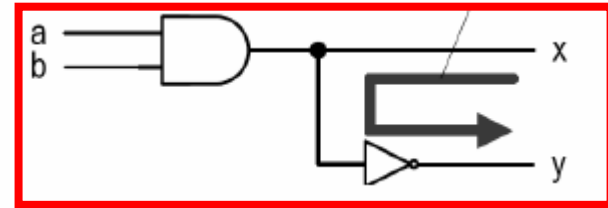
```
Good: process (a, b, c)
begin
    y <= a and b and c;
end process;
```

For a combinational circuit, all inputs need to be included in the sensitivity list!!!



One-Direction Output

```
library ieee;  
use ieee.std_logic_1164.all;  
entity mode_demo is  
    port(  
        a, b: in std_logic;  
        x, y: out std_logic);  
end mode_demo;  
architecture wrong_arch of mode_demo is  
begin  
    x <= a and b;  
    y <= not x;  
end wrong_arch;
```



Use internal signals!!!

```
architecture ok_arch of mode_demo is  
    signal ab: std_logic;  
begin  
    ab <= a and b;  
    x <= ab;  
    y <= not ab;  
end ok_arch ;
```



Complete value assignment

□ For purely combinational processes:
Every output must be assigned a value for every possible combination of the inputs

```
process (SEL, A, B)
begin
if (SEL = '0') then
Y <= A;
end if;
end process;
```



```
process (SEL, A, B)
begin
if (SEL = '0') then
Y <= A;
else
Y <= B;
end if;
end process;
```

Avoid latch in your design



No For Loops (at least for beginners)

- “For” may make your design process last forever



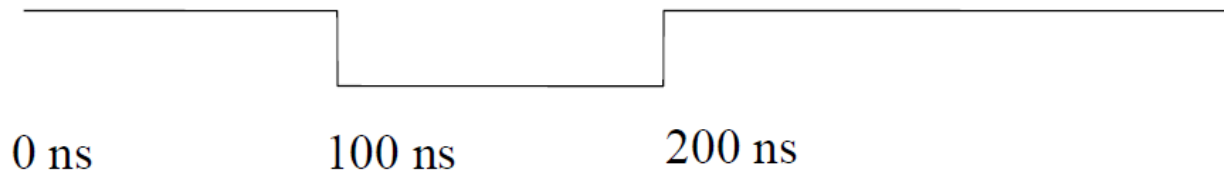
Don't Wait

- ❑ The **'wait'** or **'after'** statement can delay for a certain amount of time, e.g., "wait 10ns;"
- ❑ Only use it in test benches that are not meant to become hardware
- ❑ Do not use them in the design of your hardware
- ❑ EDA Tools will insert delay automatically by adding buffer

Example:

```
A <= '1',  
      '0' after 100 ns,  
      '1' after 200 ns;
```

Not synthesizable



DO NOT introduce uncertainty

□ 'X' and 'U' are legal VHDL, but the synthesized circuit won't behave like you expect.

```
architecture behv of ALU
is
begin
process(A,B,Sel) begin
case Sel is
when "00" =>
Res <= A + B;
when "01" =>
Res <= A + (not B) + 1;
when "1X" =>
Res <= A and B;
when "1U" =>
Res <= A or B;
when others =>
Res <= A;
end case;
end process;
end behv;
```

```
architecture behv of
ALU is
begin
process(A,B,Sel) begin
case Sel is
when "00" =>
Res <= A + B;
when "01" =>
Res <= A + (not B) + 1;
when "10" =>
Res <= A and B;
when "11" =>
Res <= A or B;
when others =>
Res <=A;
end case;
end process;
end behv;
```



Feedbacks

- ❑ Don't use a **combinational feedback**
- ❑ You never really need them.

```
process (a, b)
begin
    c <= c + a + b;
end process;
```



Multiple Source

- Drive every signal from exactly one process or concurrent assignment.

Architecture arc_ad of ad is
begin

```
    e<=a+b;  
    e<=c+d;  
end arc_ad;
```



Boolean v.s. std_logic

□ Don't assign Boolean to std_logic

```
signal a : std_logic;  
signal b : unsigned(7 downto 0);  
a <= (b = x"7E"); -- BAD: result is Boolean,  
a <= '1' when b = x"7E" else '0'; --OK
```

□ Don't test std_logic in a Boolean context

```
signal a, b, foo : std_logic;  
if a then -- BAD: A is not Boolean  
foo <= '1';  
end if;  
b <= '0' when a else '1'; -- BAD: a is not Boolean  
if a = '1' then -- OK  
foo <= '1';  
end if;  
b <= '0' when a = '1' else '0'; -- OK
```



Use Brackets

- Make sure the code operates as you want

Example:

```
if((rst = '0') && (read_address = "1234"))
```

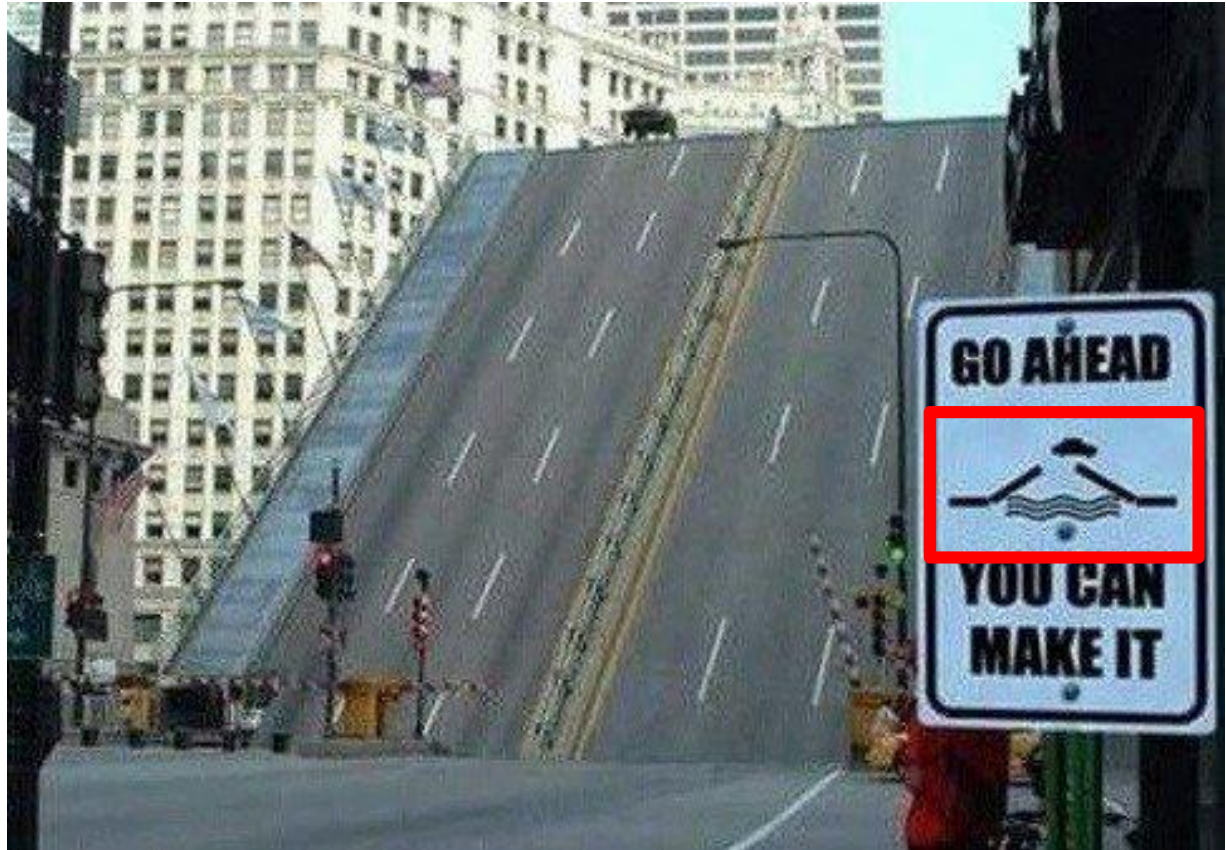


“Rules” for coding sequential circuits

- ❑ Strictly follow the synchronous design methodology; i.e., all registers in a system should be ***synchronized by a common global clock signal*** (otherwise special circuits are needed)
- ❑ The memory components should be coded clearly so that a predesigned cell can be ***inferred from the device library***.
- ❑ Isolate the memory components from the VHDL description and code them in a ***separate segment***. One-segment coding style is not advisable.
- ❑ ***Asynchronous reset***, if used, should be only for system initialization. It should not be used to clear the registers during regular operation
- ❑ Unless there is a compelling reason, a ***variable should not be used*** to infer a memory component.



Two-Segment Coding



- ▣ Separate combinational circuits and registers

Go ahead, two-segment works!



Reading suggestions

Synthesizable code

http://www.eecg.toronto.edu/~laforest/hdl_references/coding_guidelines.pdf

