Very Hard Design Language

[toc]

Basis

- 不区分大小写
- --注释
- 层级关系：库>包>实体>结构>顺序代码块

Framework

例子：三人表决器

```
library ieee;
use ieee.std_logic_1164.all;
---------------------------------
entity VOTE is
    port
    (   A1, A2, A3  :   in  bit;
        F           :   out bit );
end;
---------------------------------
architecture VOTE_arch of VOTE is
    signal F1,F2,F3 : bit;
begin
    F1 <= A1 and A2;
    F2 <= A2 and A3;
    F3 <= A1 and A3;
    F  <= F1 or F2 or F3;
end;
```

Library

结构：library.package.project

```
--常用library及其package

-------ieee--------
library ieee;
use ieee.std_logic_1164.all;

--------std---------
--VHDL标准资源库，实际程序中无需声明（默认打开）
library std;
use std.standard.all;
```

```
-------work---------
--当前工作库，存放当前设计的所有代码，无需声明
library work;
use work.all;
```

Entity

```
entity <entity_name> is
    [declarations]
    port
    (
        port_name1 : signal_mode1 data_type1;
        port_name2 : signal_mode2 data_type2;
        port_name3 : signal_mode3 data_type3
    );
end;
```

端口模式

- `in`：输入，右值
- `out`：输出，左值
- `inout`：双向
- `buffer`：带反馈的输出

Architecture

```
architecture <archi_name> of entity_name is
  [data_obj_declarations]
  [component_declarations]
  begin
      <statement>
  end;
```

Data object

- non-static
  - `signal`：表示电路内部连接
    - 定义在实体、实体端口、结构首部，向下全局
  - `variable`：在**顺序代码块**中表示一些局部信息，**即时更新，不是实际电路连接**
    - 仅在process、function、proccedure中定义
- static
  - `constant`：常数
    - 包、实体、结构中皆可定义，向下全局
  - `generic`：类属，用于说明entity结构参数的静态信息
    - 定义在实体中，可在其结构中使用

```
signal name : data_type [ := default_value];
--signal赋初值是不可综合的，仅用于仿真

variable name : data_type [ := default_value];

constant name : data_type := value;

generic
(
    name1 : data_type1 [ := default_value1];
    name2 : data_type2 [ := default_value2];
);
```

Attribute

- 信号对象s的属性
  - 信号类属性
    - s'event：s变化返回true（可综合）
    - s'stable：s稳定无变化返回true（可综合）
  - 数值类属性
    - s'length
    - s'range

Data type

常用类型

- Numeric type
  - integer：来自std库的standard包，32位
  - real：来自standard，必须明确写出小数点，**不可综合**
- Enumerated type
  - boolean：来自standard，包含false、true，只支持逻辑运算
  - bit与bit_vector：来自standard，只支持逻辑运算
  - std_logic与std_logic_vector：来自ieee库的std_logic_1164包，有8种取值（其中仅'1'、'0'、'Z'可综合）
    - 使用falling_edge(s)、rising_edge(s)来特指下降沿、上升沿
  - signed与unsigned：来自ieee库，形式类似std_logic_vector，但只支持算术运算，不支持逻辑运算

```
entity data_type_example is
    port
    (
        port1 : in bit := '1';
        port2 : in bit_vector(0 to 7);
        port3 : in bit_vector(7 downto 0);
        port4 : out std_logic;
        port5 : out std_logic(0 to 3) := "0001";
```

```
        port6 : out std_logic(3 downto 0)
    );
end;
```

- Physical
  - **不可综合**
  - `time`：`fs`的整数倍
- Array

```
type array_name is array(<array range>) of <type>;
-- examples
type bit_8 is array(7 downto 0) of bit;
type bit_4_8 is array(0 to 3) of bit_8;
type bit_4_8_ is array(0 to 3, 7 downto 0) of bit;
constant bit_table : bit_4_8 :=
(
  ('1', '0', '1', '1', '0', '1', '0', '0'),
  ('1', '0', '1', '1', '0', '1', '0', '0'),
  ('1', '1', '1', '0', '0', '1', '0', '1'),
  ('0', '0', '1', '1', '1', '1', '0', '1')
);
```

- User Defined

子类型

```
-- definition
subtype type_name is origin_type range <range>;
-------------------------------------------------
signal a : origin_type;
signal b : type_name;
a <= b;-- right
b <= a;-- wrong
```

* P.S. `<range>`

- Numeric：`<lb> to <ub>`
- Enumerated：`(ele1, ele2, ...)`
- Array：`<lb> to <ub>`或`<ub> downto <lb>`

Operator

- 赋值
  - `<=`：信号赋值
  - `:=`：其他数据对象赋值（以及赋初值）
- 逻辑运算
  - `not`优先级最高

    - and、or、nand、nor、xor并列
    - 等于=、不等于/=
    - &连接
- 算术运算
    - +、-、*、/、**、abs、mod（右值返回）、rem（左值返回）
- 移位
    - sll、srl、sla、sra、rol、ror
- 拼接
    - x <= '1'; y <= x & "1010";
    - y <= ('1', '1', '0', '1', '0');

Concurrent Statements

```
-- 1. when-else
F <= A when flag1 else
     B when flag2 else
     C;
-- 2. with-select
with flag select
  F <= A when valueA,
       B when valueB,
       C when others;
-- 3. use package component
-- 4. after（无法综合，仅用于仿真）
  -- 常量
F <= '0', '1' after 10ns, '0' after 20ns; -- 时延都是相对仿真开始时间而言的
  --信号
F1 <= F2 after 10ns; -- 信号的惯性延迟（信号变化后10ns时读取该信号）
F1 <= F2 transport after 10ns; -- 信号的传输延迟（信号变化时读取该信号并在10ns后赋值）
```

Generate：并行语句的批量描述方式

```
-- for generate/if generate可以相互嵌套
for i in <range> generate
  <statement(i)>
end generate;
------------------
if <condition> generate
  <statement>
end generate;
```

Sequential Statements

- 顺序语句常用于描述时序逻辑电路，非必要不写顺序语句
- HDL的目的是描述电路，并无所谓“执行”的说法，与编程语言有本质不同。
- 顺序语句综合较为复杂，必须清楚所需电路结构否则容易出错

Process：

- variable非实际电路，即刻赋值，仅仅起辅助作用（对变量赋的初值只会在开始执行一次）
- signals in process
  - 敏感表中为该process电路的（非锁存）输入信号
  - 未出现在敏感表中但出现在process中的输入信号，仅在敏感表中信号变化时采样，对应实际电路中的锁存器
  - 输出信号在process最后统一赋值
  - 某信号作为输出同时也作输入，在process中的计算都只取用输入值，它仅在process最后才会得到新的赋值（当该信号位于敏感表中时，电路会因此陷入unstable loop，即进程反复执行导致出错）
  - 一个输出信号有多个赋值语句，以最后的为准（这种信号多次赋值的写法没有意义）
- 绝大多数综合器不支持单个process中存在多个时钟，也不支持时钟触发的if语句带有else

```
-- 定义
-- 写在architecture中
[label: ] process (<sensitivity table>) is
[variable declarations]
begin
  ...
end process;
```

Function & Procedure：

- 与Process的区别：主要用于构建Library，描述常用局部电路以实现代码重用
- Function
  - 输入：常量/信号
  - 有且只能有一个可综合的信号返回值
  - 不同输入输出类型的函数可重载
- Procedure
  - 可有多个输出

```
-- function declaration
function function_name [(<parameter list>)] return <data_type>;
-- function implement
function function_name [(<parameter list>)] return <data_type> is
  [declaration]
begin
  <statement>
  return <expression>;
end function_name;
------------------------------
-- procedure declaration
procedure proc_name [(<parameter list>)];
-- procedure implement
procedure proc_name [(<parameter list>)] is
  [declaration]
begin
```

```
  <statement>
end proc_name;
```

顺序语句：

```
-- 1. wait
-- *碰到wait时会对之前的信号赋值，而不是最后统一赋值
wait on <signal table>; -- 信号发生变化才继续
wait until <condition>; -- 某条件发生才继续(*condition必须要发生变化，即使一开始就满足条件仍会阻塞*)
wait for <time>; -- 延时，只能仿真不可综合
wait on <signal table> until <condition> for <time>; -- 当信号改变之后，某条件发生则继续；time为最长挂起时间
-- 2. if
-- 优先级电路
if <condition> then
  <statement>
elsif <condition> then
  <statement>
else <condition> then
  <statement>
end if;
-- 3. case-when
case <expression> is
  when <value> => <statement>
  when <value> => <statement>
  when others => <statement>
end case;
-- 4. loop
loop
  <statement>
end loop;
-- while-loop
while <condition> loop
  <statement>
end loop;
-- for-loop
for <variable> in <range> loop
  <statement>
end loop;
-- exit/next
exit [when <condition>];
next [when <condition>];
```

Assert

仿真Debug工具

```
-- <condition>不满足时打印信息
assert <condition>
  report "<message>"
  severity <severity>; -- from std.standard.severity_level(default: error)
-- example
assert z = 0
  report "z isn't 0!"
  severity failure;
-- print:
-- Failure: z can't be 0!
```

Define your own packages

元件

```
--component.vhd
library ieee;
use ieee.std_logic_1164.all;
--------------------------------
entity comp_name is
  generic(...);-- 有generic时无法直接实例化（综合）这个元件
  port(...);
end;
--------------------------------
architecture of comp_name is
  begin
    <statement>
  end;
-- *如果不建包，在architecture头部声明后也可以调用元件
```

包

```
--my_package.vhd
library ieee;
use ieee.std_logic_1164.all;
--------------------------------
package package_name is

  -- component declaration
  component comp_name is
    generic(...);
    port(...);
  end component;
  -- function declaration
  -- procedure declaration
  -- constant/signal declaration
  -- subtype declaration
  -- attribute declaration
```

```
  -- alias declaration
  -- file declaration
end;
--------------------------------
package body package_name is

  -- function
  <function_implement>
  -- procedure
  <procedure_implement>

end;
```

包的使用

```
--project.vhd
library ieee;
use ieee.std_logic_1164.all;

library work;
use work.my_package.all;
--------------------------------
entity my_project is
  generic(...);
  port(...);
end;

architecture of my_project is
  begin
    label: comp_name generic map(...) port map(...);
  end;
```

- 端口映射
  - 位置映射`port map(x, y)`
  - 名称映射`port map(x => a, y => b)`
- 类属映射
  - `generic map(<para_list>)`

Configuration：

元件实体具有多种实现（即多个architecture）时，使用configuration指定

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

library work;
use work.my_package.all;
--------------------------------
entity main is
```

```
  port(...);
end;
---------------------------------
architecture main_arch of main is
begin
  label1: comp1 port map(...);
  label2: comp1 port map(...);
  label3: comp1 port map(...);

  label4: comp2 port map(...);
  label5: comp2 port map(...);
end
---------------------------------
configuration main_config of main is
  for main_arch -- specific one of main's architectures

    -- comp1, comp2 is component entities with several architectures
    for label1: comp1
      use entity work.comp1(comp1_arch1);
    end for;
    for label2: comp1
      use entity work.comp1(comp1_arch2);
    end for;
    for label3: comp1
      use entity work.comp1(comp1_arch3);
    end for;

    for all: comp2
      use configuration work.comp2_config -- it's like main_config
    end for;

  end for;
end;
```

File I/O

- 不能被综合
- 只能声明在Process中

```
library ieee;
use std.textio.all;
-------------------------------------------------------------
variable line_var : line;
file file_obj_name: text open read_mode is "<file_name>";
file file_obj_name: text open write_mode is "<file_name>";
-------------------------------------------------------------
readline(file_obj_name, line_var);
read(line_var, data_obj);
-------------------------------------------------------------
write(line_var, data_obj);
writeline(file_obj_name, line_var);
```

FSM

```
entity FSM is
  port(
    clk : in std_logic;
    reset : in std_logic;
    data_in : in std_logic;
    data_out : out std_logic
  );
end;

architecture of FSM is
  type state_type is (s0, s1, s2, s3);
  signal state : state_type;
begin
  -- switching state
  process(clk, reset)
  begin
    if reset = '1' then -- asynchronous reset
      state <= s0;
    elsif rising_edge(clk) then
      case state is
        when s0 => <switching statement>
        when s1 => <switching statement>
        when s2 => <switching statement>
        when s3 => <switching statement>
      end case;
    end if;
  end process;
  -- archi of each state
  process(state [, signal table])
  begin
    case state is
        when s0 => <statement>
        when s1 => <statement>
        when s2 => <statement>
        when s3 => <statement>
      end case;
  end process;
end;
```

TestBench

- synthesizable
  - Device Under Test
- non-synthesizable
  - signal generator
  - testbench

By-Talk about EDA

- 时序逻辑 = 组合逻辑 + D触发器 + CLK
- “数电的尽头是模电”