第二章 VHDL语言基础

2.3 VHDL程序主要构件

- Library 库
- Package 包
- Entity 实体
- Architecture 结构体
- Configuration 配置

2.3.1 库

- IEEE库
  - `std_logic_1164`: *functions & data types for multi-level logic*
  - `std_logic_arith`: *defines signed, unsigned types and basic arithmetic operations for representing integers in standard ways*
  - `std_logic_signed`: *signed arithmetic functions*
  - `std_logic_unsigned`: *overloading operators for mixed operation, coversion between different data types*
  - for datatype `std_logic` and `std_logic_vector`, **only logical operations can be applied without packages `std_logic_signed` and `std_logic_unsigned`**
- STD库
  - 默认库，定义最基本的数据类型，调用时**不需要显式说明**
- WORK库
  - 存放用户定义的单元和包，调用时**不需要显式说明**

2.3.2 实体

```
ENTITY entitiy_name IS
    GENERIC(const_name:dtype:=value); --[类属参数说明],分号结尾
    PORT(port_name:port_direction dtype); --[端口说明],分号结尾
END [ENTITY] entity_name; --分号结尾
```

- `entity_name` 不能以数字开头
- [类属参数说明] 是**常数**，用于说明时间参数、总线宽度等静态信息
- [端口说明] 的dtype有四种，分别为：
  - IN
  - OUT
  - INOUT
  - BUFFER: 内部有**反馈**
- -[端口说明]格式如下

```
PORT(port1_name:port1_direction dtype; --每个声明之间用分号分隔
    port2_name:port2_direction dtype; --每个声明之间用分号分隔
    port3_name:port3_direction dtype); --最后一个声明不需要分号分隔，但外部括号
```

2.3.3 结构体

```
ARCHITECTURE architecture_name OF entity_name IS
  [DEFINITION]  --定义语句，可定义signal,constant,type,function,component等
BEGIN
  [concurrent statements]
END architecture_name
```

- `architecture_name` 不能以数字开头
- `DEFINITON` **only valid in its defining architecture**
- Each design entity composes only one entity and one architecture. Therefore, it is necessary to specify which of the architectures should be used at any particular time through `configuration declaration`

2.3.4 配置

```
CONFIGURATION configuration_name OF entity_name IS
  FOR architecture_name
  END FOR;
END configuration_name;
```

- `architecture_name` 指多个结构体中要选用的结构体名称

2.4 VHDL数据对象

2.4.1 常量

```
CONSTANT const_name:dtype:=value --注意要有CONSTANT做引导
```

2.4.2 变量

```
VARIABLE variable_name:dtype:=value
```

- Defined in `process` or sub-program (`function`, `procedure`)
- It is **Local** but not Global
- Valid only in the **sequential** areas within a process, sub-program (not within the architecture body)

2.4.3 信号

2.4.3.1 信号的定义

```
SIGNAL signal_name:dtype:=value  --使用:=对信号赋初值
                                 --这个初始值只用于仿真，综合器不支持
```

- 信号在声明时用:= 赋初值
- 信号赋值使用 <= 注意此操作有**延时**

    - 对于在进程内的信号赋值操作，每次进程process被触发后，虽然进程内有赋值语句，但是**只有当这次进程挂起时信号的赋值语句才会生效**

- **All PORTS of an ENTITY are signals by default**

2.4.3.2 信号与变量的比较

- SIGNAL是global量，可在多个PROCESS中传递，但是VARIABLE是local量，仅在当前的进程、子程序中有效
- SIGNAL除了值以外还存放了历史信息、波形值等，可用于仿真，但是VARIABLE不可以仿真
- PROCESS只对信号敏感，不对变量敏感
- 变量不与任何实际电路连线对应，不代表变量赋值行为不产生与之对应的硬件结构。变量赋值语句既然是可综合的，就一定会对硬件结构产生影响。

信号与变量的比较（考试答案版）

1. 声明形式与赋值符号不同。
2. 有效域不同。信号在结构体内、进程外定义，变量在进程内定义。信号有效域为整个结构体，可在不同进程中传递数值；变量的有效域是定义该变量的进程。
3. 赋值操作的执行不同。变量赋值是立即生效，在执行下一条语句是，该变量的值为上一句新赋的值；信号赋值具有延时性，信号的赋值语句即使被执行也不会使信号立即发生代入，下一条语句执行时，仍使用原来的信号值（信号是在进程挂起时发生代入）。
4. 对应硬件结构不同。信号对应实际电路的某个节点或信号线；变量是一个抽象值，它不与任何实际电路连线对应。注意：变量不与任何实际电路连线对应，不代表变量赋值行为不产生与之对应的硬件结构。变量赋值语句既然是可综合的，就一定会对硬件结构产生影响。
5. 应用场合不同。在实际应用中，信号的行为更接近硬件的实际情况，因此更多地使用信号进行电路内部数据传递，只在描述一些用信号很难描述的算法时才用到变量。变量用于进程语句和子程序中中间的数据存储。

2.5 VHDL数据类型

2.5.1 Standard Datatype

Integer

- For representation of bus width, **bit operations/logic operations are not permitted**, needs **range specification**
- 4 byte length

Real

- Floating point numbers
- A majority of EDA tools do not support float operations

Natural/ Positive

- Subset of integer
- Needs **range specification**

Bit

- Single value
- Single quote ' '

Bit_vector

- Double quote " "

Character

- ASCII value
- Case sensetive
- Single quote ' '

String

- Vector of Character
- Double quote " "

Boolean

- Two status, TRUE or FALSE

Time

- For simulation

Severity level

- error reminding

2.5.2 IEEE Defined Datatype

std_logic

- 8-valued logic
- Logic Levels:
    - 'X': unknown, impossible to determine this value
    - 'Z': high impedance
    - '0': logic 0
    - '1': logic 1
    - 'W': weak signal, impossible to tell if it should be 0 or 1
    - 'L': weak signal that should go to 0
    - 'H': weak signal that should go to 1
    - '-': don't care
- **resolved logic system**: if any two std_logic signals are connected to the same node, then conflicting logic levels are automatically resolved according to the rules
- Attention: **case-sensitive** , e.g. High impedance is represented by **'Z'** rather than 'z' .

	X	0	1	Z	W	L	H	-
X	X	X	X	X	X	X	X	X
0	X	0	X	0	0	0	0	X
1	X	X	1	1	1	1	1	X
Z	X	0	1	Z	W	L	H	X
W	X	0	1	W	W	W	W	X
L	X	0	1	L	W	L	W	X
H	X	0	1	H	W	W	H	X
-	X	X	X	X	X	X	X	X

std_logic_vector

- no extrapackage required
- only logical operation
- with `IEEE.std_logic_unsigned` or `IEEE.std_logic_signed` included, arithmetic operations are allowed.

std_ulogic

- Unresolved logic system
- Output wires should never be connected together directly
- Logic Levels:
  - 'U': uninitialized
  - 其他8种与`std_logic`相同

signed and unsigned

- Package `IEEE.std_logic_arith.all` should be included.
- Arithmetic operations only

2.5.3 User-defined datatypes

```
TYPE Type_name IS Type_def OF basic_dtype;
-- or
TYPE Type_name IS Type_def;
```

User-defined datatypes

- Enumeration types
  - used for state machine

```
TYPE week IS (sun, mon, tue, wed, thu, fri, sat) ;
```

- Subtype
  - sub-set of existing types

```
SUBTYPE natural IS INTEGER RANGE 0 TO INTEGER'HIGH;
```

- Integer, Real
  - 若整数和实数的取值范围太大，综合其将无法综合，因此需要限定范围

```
TYPE percent IS INTEGER RANGE -100 TO 100;
```

- Array

```
TYPE array_name IS ARRAY(range) OF dtype;
```

例：

```
TYPE stb IS ARRAY(7 DOWNTO 0) OF std_logic;
```

- Records
  - 与array类似，只不过records可以包含不同数据类型的元素

```
TYPE birthday IS RECORD
  day: INTEGER RANGE 1 TO 31;
  month: month_name;
END RECORD;
```

2.5.4 Data Conversion

程序包	函数名	功能
STD_LOGIC_1164	TO_STDLOGICVECTOR(A)	由 BIT_VECTOR 转换为 STD_LOGIC_VECTOR
	TO_BITVECTOR(A)	由 STD_LOGIC_VECTOR 转换为BIT_VECTOR
	TO_STDLOGIC(A)	由 STD_LOGIC 转换 STD_LOGIC
	TO_BIT(A)	由STD_LOGIC转换BIT
STD_LOGIC_ARITH	CONV_STD_LOGIC_VECTOR(A,n) (n为位长)	由INTEGER，UNSIGNED，SIGNED 转换 STD_LOGIC_VECTOR
	CONV_INTEGER(A)	由UNSIGNED，SIGNED转换为INTEGER
STD_LOGIC_UNSIGNED	CONV_INTEGER(A)	由 STD_LOGIC_VECTOR 转换为INTEGER

2.6 VHDL运算符

2.6.1 算术运算符

- ** 乘方
- REM 取余
  - a REM b所得结果与a符号相同，绝对值小于b绝对值
- MOD 取模
  - a MOD b所得结果与b符号相同，绝对值小于b绝对值
- SLA SRA 算术左移 算术右移
  - 左移保持最低位不变，右移保持最高位不变，所有数据（包括最高位）左移或右移
  - 见书本P21图2-6

综合：加减乘可以综合，除要满足除数为2的n次幂才可以综合（移位），其他运算均不可综合

2.6.2 逻辑运算符

注意P22图2-7的SLL SRL ROL ROR

2.6.3 关系运算符

- /=不等于
- 所有关系运算符的两个操作数必须**类型相同**

2.7 VHDL基本语句

- NULL为顺序语句中的用法，unaffected为并行语句的用法，两者均表示没有操作

2.7.1 Concurrent Statements

2.7.1.1 When...else Statement

```
赋值目标 <= 表达式 WHEN 赋值条件 ELSE
           表达式 WHEN 赋值条件 ELSE
           表达式 WHEN 赋值条件 ELSE
           ...
           表达式; --最后才有';'，其他位置没有
```

优先级从最上面的语句开始向下逐一递减，按顺序判断

2.7.1.2 With...select Statement

```
-- 类比switch case语句
WITH 选择表达式 SELECT
赋值目标 <= 表达式 WHEN 条件, --注意用','分隔
           表达式 WHEN 条件,
           表达式 WHEN 条件,
           ...
           表达式 WHEN OTHERS;  --注意最后用';'
                                --最后应当用OTHERS涵盖所有未指定的情况
                                --若最后要不做操作，则表达式使用`UNAFFECTED`
```

所有语句同时(simutaneously)判断，不分优先级

2.7.1.3 Component Instantiation Statement (元件例化语句)

```
--------- 将设计实体定义为元件，该实体已经预先定义好 ---------
COMPONENT component_name IS
  GENERIC(类属表);  --optional
  PORT(端口名表)    --compulsory, instantiation
END COMPONENT;

------------------------ 调用方式 -------------------------
USE work.components.all;

object_name: component_name GENERIC MAP(...); --写入例化的类属参数
                            PORT MAP(...);  --写入端口映射
```

- **在ARCHITECTURE中应放在BEGIN语句前**
- 端口映射有两种方式
  - 对照COMPONENT的端口名表，按顺序写入例化的端口
  - 使用 例化端口名>=端口名表中的端口名 的格式

实例见P54-55加法器的封装

2.7.1.4 Generate Statement

For Generate

```
---------For Generate--------
[标号:]FOR 循环变量 IN 取值范围 GENERATE
        说明部分;
       BEGIN
        并行语句;
       END GENERATE[标号];
```

- 生成L个component单元，需要长度为L+1的array of signals作为中继连接，形成component单元"链表"，表头表尾需要分配连接到上述array of signals的头尾

If Generate

```
---------If Generate--------
[标号:]IF 条件 GENERATE
        说明部分;
       BEGIN
        并行语句;
       END GENERATE[标号];
```

- 生成L个component单元，需要长度为L-1的array of signals作为中继连接，形成component单元"链表"，表头表尾不需要分配连接(课本P29例2-13)

2.7.1.5 Process Statement

*WAIT*语句格式见课本*P26例2-10*

- 启动Process的两种方式：敏感参数表和WAIT语句，两者不能并存
- 敏感参数表和WAIT语句的内容必须为signal
- **WAIT不能被综合器综合**，只能用于测试基准

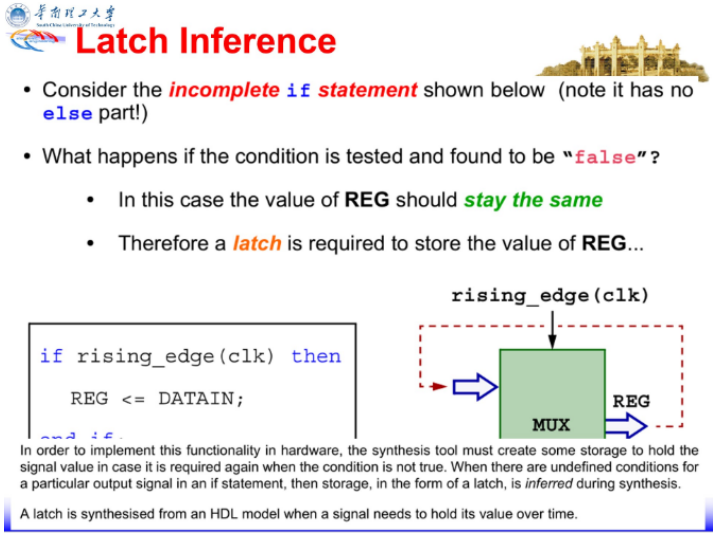
2.7.2 Sequential Statements

2.7.2.1 If-then-else statements

```
IF first_conditon THEN
  statements;
ELSIF second_condition THEN     --注意为 elsif
```

```
  statements;
ELSE
  statements;
END IF;
```

- 每一个condition必须为boolean类型，可以为signal, constant, variable
- If statements are sythesised using **multiplexers**。综合器用多路选择器或基本逻辑门的组合来实现电路。用多路选择器实现电路时，if...elsif...else中隐含的优先关系会被消去，这是设计师应该注意的问题。
- **Incomplete `if` statement (without `else` part) may introduce register: the sequential logic and combinational logic are mixed in the same process (introduce combinational logic in sequential logic or introduce sequential logic in combinational logic), unwanted register may be introduced.**



2.7.2.2 Case Statements

```
CASE control_expression IS
  WHEN test_expression1 => statements1;
  WHEN test_expression2 => statements2;
  ...
  WHEN OTHERS => statements_others; --used to guarantee that all
                                    --the possibilities are covered
                                    --`NULL` can be used to represent
                                    --'no operation'
END CASE;
```

2.7.2.3 For Loops

```
FOR loop_parameter IN conditions LOOP
  statements;
END LOOP;
```

- The `loop_parameter` is **implicitly declared and cannot be references outside the loop**
- `loop parameter` cannot be modified inside the loop
- `for loops` are synthesised by unrolling the loop and generating multiple copies of the hardware described within the loop

2.7.2.4 While Loops

```
WHILE conditions LOOP
  statements;
END LOOP;
```

- **`While loops` are not synthesisable since it is not generally known how many iterations will be executed**

2.7.2.5 Infinite Loop

```
标号: LOOP
  statements;
END LOOP 标号;
```

控制loop的关键字：

- `EXIT`： 满足条件退出循环
  - `EXIT 标号 WHEN 条件`
- `NEXT`：满足条件直接进入下一个循环
  - `NEXT 标号 WHEN 条件`

2.7.2.6 Wait Statement

综合器要求`WAIT`语句必须放在进程的首部或者尾部，并且一个进程中的`WAIT`语句不能超过一个

- `WAIT`
- `WAIT ON`
  - Accept multiple signals
- `WAIT UNTIL`
  - Accept only one signal
  - Must be the first statement in the process
- `WAIT FOR`
  - intended for **simulation only**
  - example: `WAIT FOR 5ns`

2.7.3 Attributes Statements

- 使用`rising_edge(clk)`时，`clk`只能是`std_logic`

2.7.3.1 Data Attributes

- return information regarding a `data vector`
  - d'LOW: Returns lower array index
  - d'HIGH: Returns upper array index
  - d'LEFT: Returns leftmost array index
  - d'RIGHT: Returns rightmost array index
  - d'LENGTH: Returns vector size
  - d'RANGE: Returns vector range
  - d'REVERSE_RANGE: Returns vector range in reverse order

If the signal is of enumerated type, then:

- d'VAL(pos): Returns value in the position specified
- d'POS(value): Returns position of the value specified
- d'LEFTOF(value): Returns value in the position to the left of the value specified
- d'VAL(row, column): Returns value in the position specified; etc.

2.7.3.1 Signal Attributes

- serve to monitor a signal (return `TRUE` or `FALSE`)

Signal Attributes:

- s'EVENT: Returns true when an event occurs on s
- s'STABLE: Returns true if no event has occurred on s
- s'ACTIVE: Returns true if s = '1'
- s'QUIET ⟨time⟩: Returns true if no event has occurred during the time specified
- s'LAST_EVENT: Returns the time elapsed since last event
- s'LAST_ACTIVE: Returns the time elapsed since last s = '1'
- s'LAST_VALUE: Returns the value of s before the last event; etc.

Examples:

```
IF (clk'EVENT AND clk='1')...          -- EVENT attribute used
                                       -- with IF
IF (NOT clk'STABLE AND clk='1')...     -- STABLE attribute used
                                       -- with IF
WAIT UNTIL (clk'EVENT AND clk='1');    -- EVENT attribute used
                                       -- with WAIT
IF RISING_EDGE(clk)...                 -- call to a function
```

2.8 测试基准

2.9 其他构件

2.9.1 Block

- **Concurrent Statements**

```
label: BLOCK (guard expression) --BLOCK is excuted only when the
                                --guard expression is TRUE
  [declarative part]
BEGIN
  concurrent_guarded_an_unguarded_statements
END BLOCK label;
```

2.9.2 Function

- **Sequential Statements**
- **Cannot** have the statements: `WAIT`, `SIGNAL` declarations and `COMPONENT` instantialtions

```
-------------------------Declaration-----------------------
FUNCTION function_name (parameters: parameter_type) RETURN return_dtype;
---------------------------Body--------------------------
FUNCTION function_name (parameters: parameter_type) RETURN return_dtype IS
  Statements;
BEGIN
  Sequential Statements;
END function_name
```

- 函数的参数显然是输入类型，因此不需要用`IN OUT`等方向类型说明
- 函数参数只能为`constant`和`signal`
- No range specification should be included
- Only **one** return value
- 不允许在函数中使用元件例化语句
- 如果只在一个结构体中定义并调用函数，则仅在结构体的说明部分定义函数体即可
- 如果将函数封装入`package`中，那么在程序开始要导入`package`；函数在`package`中定义，且要包括`Declaration`和`Body`两部分

2.9.3 Procedure

```
-------------------------Declaration-----------------------
--注意没有返回值说明
PROCEDURE procedure_name (parameters: parameter_type);
---------------------------Body--------------------------
PROCEDURE procedure_name (parameters: parameter_type) IS
  Statements;
BEGIN
  Sequential Statements;
END procedure_name
```

- 参数需要方向类型说明
- Return **more than one** value
- 参数可以为`constant signal` 和 `variable`
- 输出默认为`variable`类型
- 不允许在过程中使用元件例化语句

2.9.4 Package

```
PACKAGE package_name IS
  Package_head_statements;
END package_name;
```

```
PACKAGE BODY package_name IS
  package_somponents_and_statements;
END package_name
```