VHDL 编程的一些心得体会

http://www.cnelectron.com.cn 日期: 2005-04-29 阅读: 55 字体:大 中 小 双击鼠标滚屏

VHDL 是由美国国防部为描述电子电路所开发的一种语言，其全称为(Very High Speed Integrated Circuit) Hardware Description Language。与另外一门硬件描述语言 Verilog HDL 相比，VHDL 更善于描述高层的一些设计，包括系统级（算法、数据通路、控制）和行为级（寄存器传输级），而且 VHDL 具有设计重用、大型设计能力、可读性强、易于编译等优点逐渐受到硬件设计者的青睐。但是，VHDL 是一门语法相当严格的语言，易学性差，特别是对于刚开始接触 VHDL 的设计者而言，经常会因某些小细节处理不当导致综合无法通过。为此本文就其中一些比较典型的问题展开探讨，希望对初学者有所帮助，提高学习进度。

一． 关于端口

VHDL 共定义了 5 种类型的端口，分别是 In, Out,Inout, Buffer 及 Linkage，实际设计时只会用到前四种。In 和 Out 端口的使用相对简单。这里，我们主要讲述关于 buffer 和 inout 使用时的注意事项。

与 Out 端口比，Buffer 端口具有回读功能，也即内部反馈，但在设计时最好不要使用 buffer，因为 buffer 类型的端口不能连接到其他类型的端口上，无法把包含该类型端口的设计作为子模块元件例化，不利于大型设计和程序的可读性。若设计时需要实现某个输出的回读功能，可以通过增加中间信号作为缓冲，由该信号完成回读功能。

双向端口 Inout 是四种端口类型中最为特殊的一种，最难以学习和掌握，为此专门提供一个简单程序进行阐述,部分程序如下：

… …

① DataB<=Din when CE=’1’ and Rd=’0’ else

② (others=>’Z’);

③ Dout<=DataB when CE=’1’ and Rd=’1’ else

④ (others=>’1’);

… …

程序中 DataB 为双向端口，编程时应注意的是，当 DataB 作为输出且空闲时，必须将其设为高阻态挂起，即有类似第②行的语句，否则实现后会造成端口死锁。而当 DataB 作为有效输入时， DataB 输出必须处于高阻态，对于该例子中即，当 CE=’1’ and Rd=’1’时，输出 DataB 应处于高阻态。

二．信号和变量

常数、信号和变量是 VHDL 中最主要的对象，分别代表一定的物理意义。常数对应于数字电路中的电源或地；信号对应某条硬件连线；变量通常指临时数据的局部存储。信号和变量功能相近，用法上却有很大不同。

表 1 信号与变量主要区别
信号 变量
赋值延迟 至少有△延时 无，立即变化
相关信息 有，可以形成波形 无，只有当前值
进程敏感 是 否
全局性 具有全局性，可存在于多个进程中 只能在某个进程或子程序中有效
相互赋值关系 信号不能给变量赋值 变量可以给信号赋值
对于变量赋值操作无延迟，初学者认为这个特性对 VHDL 设计非常有利，但这只是理论上的。基于以下几点原因，我们建议，编程时还是应以信号为主，尽量减少变量的使用。

（1）变量赋值无延时是针对进程运行而言的，只是一个理想值，对于变量的操作往往被综合成为组合逻辑的形式，而硬件上的组合逻辑必然存在输入到输出延时。当进程内关于变量的操作越多，其组合逻辑就会变得越大越复杂。假设在一个进程内，有关于变量的 3 个 级连操作，其输出延时 分别为 5ns,6ns,7ns,则其最快的时钟只能达到 18ns。相反，采用信号编程，在时钟控制下，往往综合成触发器的形式，特别是对于 FPGA 芯片而言，具有丰富的触发器结构，易形成流水作业，其时钟频率只受控于延时最大的那一级，而不会与变量一样层层累积。假设某个设计为 3 级流水作业，其每一级延时分别为 10ns,11ns,12ns,则其最快时钟可达 12ns。因此，采用信号反而更能提高设计的速度。

（2）由于变量不具备信息的相关性，只有当前值，因此也无法在仿真时观察其波形和状态改变情况，无法对设计的运行情况有效验证，而测试验证工作量往往会占到整个设计 70％～80％的工作量，采用信号则不会存在这类问题。

（3）变量有效范围只能局限在单个进程或子程序中，要想将其值带出与其余进程、子模块之间相互作用，必须借助信号，这在一定程度上会造成代码不够简洁，可读性下降等缺点。当然，变量也具有其特殊的优点，特别是用来描述一些复杂的算法，如图像处理，多维数组变换等。

三．位（矢量）与逻辑（矢量）
bit 或其矢量形式 bit_vector 只有’0’和’1’两种状态，数字电路中也只有’0’和’1’两种逻辑，因此会给初学者一个误区，认为采用位（矢量）则足够设计之用，而不必像 std_logic 那样出现’X’,’U’,’W’各种状态，增加编程难度。但实际情况却并非如此，以一个最简单 D 型触发器设计为例
… …
① process(clk)
② begin
③ if clk’event and clk=’1’ then
④ Q<=D;
⑤ end if;
⑥ end process;
… …
实际中 clk 对数据端 D 的输入有一定的时间限制，即在 clk 上升沿附近（建立时间和保持时间之内），D 必须保持稳定，否则 Q 输出会出现亚稳态，如下图所示。

图 1 建立时间和保持时间

当 clk 和 D 时序关系不满足时，由于 bit 只有’0’或’1’，系统只能随机的从’0’和’1’中给 Q 输出，这样的结果显然是不可信的；而采用 std_logic 类型，则时序仿真时会输出为一个’X’，提醒用户建立保持时间存在问题，应重新安排 D 和 clk 之间时序关系。

此外，对于双向总线设计（前面已提及）、 FPGA/CPLD 上电配置等问题，如果没有’Z’,’X’等状态，根本无法进行设计和有效验证。

四．关于进程

进程（Process）是 VHDL 中最为重要的部分，大部分设计都会用到 Process 结构，因此掌握 Process 的使用显得尤为重要。以下是初学和使用 Process 经常会出错的例子。

1. 多余时钟的引入

在设计时往往会遇到这种情况，需要对外部某个输入信号进行判断，当其出现上跳或下跳沿时，执行相应的操作，而该信号不像正常时钟那样具有固定占空比和周期，而是很随机，需要程序设计判断其上跳沿出现与否。这时，很容易写出如下程序：

① process(Ctl_a) -- Ctl_a 即为该输入信号
② begin
③ if Ctl_a’event and Ctl_a=’1’ then
④ … … ; --执行相应操作
⑤ end if ;
⑥ end process;

由于出现第③行这类语句，综合工具自动默认 Ctl_a 为时钟，某些 FPGA 更会强行将该输入约束到时钟引脚上。而设计者的初衷只是想将其作为下位机的状态输入以进行判断。上面的程序容易造成多时钟现象，增加设计的难度。解决的办法可以如下，将 Ctl_a 增加一级状态 Ctl_areg 寄存，通过对 Ctl_a 和 Ctl_areg 状态判断上跳与否，改正程序如下：

① process(clk)
② begin
③ if clk’event and clk=’1’ then
④ Ctl_areg<=Ctl_a;--产生相邻状态
⑤ if Ctl_areg=’0’ and Ctl_a=’1’ then--上跳判断
⑥ … … ; --执行相应操作
⑦ end if;
⑧ end if;
⑨ end process;

程序中第④行用以产生两个相邻状态，第⑤行对前后状态进行判断是否有上跳现象发生。其中，需注意的是 clk 的时钟频率应明显快于 Ctl_a 信号的变化频率，以保证正确采样。

2. 输出多驱动

误用 Process 经常会引起输出多驱动源的发生，即在两个以上的进程内对同一信号赋值操作。以下程序就出现了这类情况：

⑴ Proc_a: process(clk)
⑵ begin
⑶ if clk’event and clk=’1’ then

⑷ Dout<=Din_A;
⑸ end if
⑹ end process;;
⑺
⑻ Proc_b:process(sel_en)
⑼ begin
⑽ if sel_en=’1’ then
⑾ Dout<=Din_B;
⑿ end if;
⒀ end process;
进程 Proc_a 和 Proc_b 中都出现了对 Dout 的赋值语句，设计者原本的想法是，只要合理控制好 clk 和 sel_en 输入，使其不发生冲突，即 clk 上升沿时 sel_en 不为’1’；sel_en 为’1’时，不出现 clk 的上升沿，这样 Proc_a，Proc_b 两个进程就不会发生冲突。但综合时，综合工具会将所有可能情况全部罗列进去，包括第⑶行和第⑽行同时成立的情况，此时对于 Dout 就有 Din_A 和 Din_B 两个输入驱动，Dout 不知接收哪一个，因此该程序无法综合，改正的方法是只要将两个进程合并成一个即可。
由于进程在 VHDL 中的重要性，对此专门做了一个总结如下：
（1）一个进程中不允许出现两个时钟沿触发，（Xilinx 公司 CoolRunner 系列 CPLD 支持单个时双钟的双触发沿除外）
（2）对同一信号赋值的语句应出现在单个进程内，不要在时钟沿之后加上 else 语句，如 if clk’event and clk=’1’ then - else … 的结构，现有综合工具支持不了这种特殊的触发器结构
（3）当出现多层 IF 语句嵌套时，最好采用 CASE 语句替代，一是减少多层嵌套带来的延时，二来可以增强程序的可读性
（4）顺序语句如 IF 语句、CASE 语句、LOOP 语句、变量赋值语句等必须出现在进程、函数或子程序内部，而不能单独出现在进程之外
（5）进程内部是顺序执行的，进程之间是并行运行的；VHDL 中的所有并行语句都可以理解为特殊的进程，只是不以 Process 结构出现，其输入信号和判断信号就是隐含的敏感表

五．关于 VHDL 学习中的几点说明
与软件语言相比，VHDL 最重要的特点就在于它的并行运行特性，当设计好的电路上电后，器件内部所有信号将同时并发工作，而不会以软件方式按照程序顺序执行，即使在进程内部也是趋向并行工作的。例如以下程序：
① process(clk)
② begin
③ if clk’event and clk=’1’ then
④ <= ;
⑤ <= ;
⑥ end if;;
⑦ end process;
综合的结果两个独立的 D 型触发器，虽然进程内部应按顺序执行，但是硬件实现后，只要采样到时钟上升沿， 和 状态会同时翻转，而不会先执行 的变化，然后才会去执行 的转变。因此，VHDL 学习过程中，应加强硬件概念的理解，没有硬件概念或是硬件概念不强，在设

计时，往往会将 VHDL 设计以软件编程的方式来处理，而得出一些不可思议的结果。
作为一门硬件描述语言，VHDL 几乎可以用来描述现有的大型系统数字电路、算法以及其它设计。但是，限于目前综合工具的水平，VHDL 中的许多语法还不能支持，例如：

```
dout<=din after 5 ns;
```

综合时就无法达到如此精度，因此这条语句主要用来编写测试激励，而很少出现在设计实体中。类似的情况还有很多，目前 VHDL 设计使用的也只是整个标准中的一部分，这也正是 VHDL 的“可综合子集”性质，它一定程度上限制了 VHDL 的广泛应用，但是随着综合技术的发展，这种情况会逐渐得以改善，VHDL 也将在各个领域中发挥出愈来愈重要的作用