PERL在IC设计中的应用

中电华大电子设计有限公司　华胜华　刘伟平

摘要:由于IC设计的复杂性,使得在设计过程中有大量的数据需要处理。本文介绍了在IC设计过程中,如何使用PERL语言加快IC设计流程,并且给出了PERL语言的相关知识、以及下载和安装方面的内容。文章并没有涉及PERL编程本身。

1.PERL简介

1.1 背景

IC设计是一个很复杂的过程,需要处理多种数据。为此,多种昂贵的EDA工具被开发出来用于IC设计的各个流程。如:Cadence的Ncverilog、Dracula;Synopsys公司的dc_shell等等。但是,只有这些工具是远远不够的,设计中的许多数据还必须有效地组织起来,而目前的EDA工具对数据的组织处理效率往往不能满足实际需求。

目前大家的做法一般有两种:(1)手工进行数据处理。(2)编制工具进行数据处理。手工的方式是大家比较常用的,其劣势不言而喻。编制工具一般会采用语言脚本(script),如:batch文件,sh/csh等等或者采用C语言。PERL是一脚本语言,由于其诸方面的优越性,在实际的IC设计中作为辅助手段被广泛采用。

1.2 什么是PERL

PERL是Practical Extraction and Report Language的缩写。它是一种通用的应用广泛的编程语言,凡是其他编程语言能够使用的场合,它都可以使用。它的最为特殊的优点是能把各种应用软件工具连接在一起,因此也被称作"胶水语言"。

1.3 PERL的特点

不需编译:直接解释运行。

可移植性强:至少可以在20多种操作系统上运行,不需修改。

处理和表达能力强:兼有多种语言的精华和特性。完全支持正则表达式。

效率高:优化的内核,使得解释运行的效率不亚于某些编译执行的语言。

1.4 下载PERL

PERL工具包是免费的,可以在网上自由下载。下载地址是:http://www.perl.com/Downloads

运行于Windows平台上的ActiveState Tool公司的PERL的下载地址:http://www.ActiveState.com

由于支持的平台比较多,可以选择自己所要用的平台版本进行下载。下载一般是一个软件包,应该下载其最新版本。

1.5 安装PERL

1.5.1 Windows

推荐使用ActiveState Tool公司的PERL产品。

该公司提供的工具可以自行安装 PERL，其安装工作就像安装 Windows 的其他产品一样，方便、快捷。

1.5.2 UNIX/Linux

在 UNIX/Linux 的平台上安装需要 ANSI C 编译器，同时下载 PERL 源程序包（文件名类似于 stable.tar.gz）。用以下命令进行安装：

```
% gzip -c -f -d stable.tar.gz | tar xfv -
% sh Configure
% make
% make test
```

上述命令的执行中，如有提问，则采用其缺省值即可。然后以超级用户的身份登陆，再继续安装：

```
% su
# make install
```

然后回到普通用户。如下面的命令执行正确，则安装成功：

```
% perl -v
```

1.6 运行 PERL

对于 Windows 平台，必须先产生一个命令行的窗口。用命令行的方式运行 PERL。对于 UNIX/Linux 则在普通的 shell 下运行。可以用下面的三种方法运行 PERL：

(1)交互式执行

```
% perl [options]
```

然后再标准输入 PERL 的语句，交互执行。

(2)文件方式

这种方式下，需要先写好 PERL 语言程序，例如：test.pl。然后用下面的方式运行：

```
% perl [options] test.pl
```

(3)直接运行 PERL 程序

```
% test.pl
```

这种方法中，需要在文件首增加类似如下的行：

```
#! /usr/local/bin/perl [options]
```

2.IC 设计流程和 PERL

在 IC 设计流程中，PERL 可以作为专业 EDA 工具之间平滑连接的“胶水”发挥重要作用。

2.1 IC 设计流程简介

对于一个 IC 设计，大体上可以分为以下几个主要处理过程：

- 整体设计
- 逻辑设计(编码)
- 逻辑验证(仿真，包括 RTL 级和门级)
- 批量测试（大规模仿真，包括 RTL 级和门级）
- 逻辑综合
- 版图设计(后端，包括布图，提取等等)

上述的许多步骤中，都要涉及到大量数据处理的问题。由于芯片设计的复杂性、仿真要求的高覆盖率，使得为 EDA 软件提供数据输入和将若干个工具的数据连接起来变得很复杂。PERL 则在这里充当了良好的“胶水作用”。

2.2 PERL 的作用

在有数据需要处理的地方，PERL 就会起作用。在 IC 设计中，逻辑设计、逻辑验证和大规模验证等等，都可以用 PERL 编制工具，以加快整个处理流程。

2.2.1 逻辑设计

(1) 可以用 PERL 程序对 HDL (Verilog 或 VHDL，将以 Verilog 作例子)语言做扩充，加快编码过程，并可减少人工的失误。

(2)可以用 PERL 程序自动产生部分代码，尤其是生成需要较复杂计算、规律性强而又容易出错的代码。

2.2.2 逻辑仿真

对于逻辑仿真来说，除了仿真工具外，最重要的就是激励的生成。另外仿真结果的处理也很重要。没有工具的帮助，这些事情将会是既繁复，又容易出错。

2.2.3 批量测试

这里的"批量"有两方面的含义。其一，是指要仿真的数据量比较大；其二，是指仿真本身注重的是仿真的"输入"和"输出"。另外，"批量测试"中的数据本身可能有相关性，需要对批量测试的"输入"和"输出"数据作统一处理。例如，在通信芯片的仿真中，要仿真的数据可能是一段声音，需要在发送前将声音数据进行"分块"，而在接收后需要将所得的数据进行"合并"。

很多芯片设计过程中，这种"批量测试"要么给忽略了，要么是用手工进行简单的操作。如果采用编制 PERL 程序的办法，可以将上述复杂的操作变成一条简单的命令。快捷、方便，而且不出错。

实际上，利用 PERL 程序，可以方便地将"批量测试"扩展成"基于逻辑仿真的演示环境"。既可进行逻辑仿真，又可以方便处理其输入/输出。

3.PERL 对 Verilog 语言做扩充

在 IC 设计中，HDL（以使用最多的 Verilog 为例，VHDL 类似）程序的编制是芯片具体实现的开始。对于规模较大的芯片，Verilog 代码的编制量很大。好多代码是可以用另外一些代码生成出来的，例如：instance 调用。而另外一些代码则规律比较明显，但是写起来也很烦，类似等等许多种情况。对于这些手工编写繁复，而且极容易出错的代码，则可以用 PERL 对 Verilog 语言做扩充。

这里所谓的"扩充"，实际上只是在源程序代码里增加必要的"预处理元素"，然后用 PERL 编制的程序对该代码进行预处理，输出的代码则为普通的 Verilog 程序。

这里列出几个重要的"预处理元素"，

#define <name> <define-body>

#instance <cell-name> <instance-name>

#cellpin_begin

#cellpin_end

这一部分请参见[1]。

4.PERL 自动生成 Verilog 数据

4.1 自动生成数据的必要性

这里的"数据"实际上是 Verilog 源代码的一部分。由于它可以用其他的数据运算出来，因此看起来还是"数据"。如果都是用手工计算，然后将计算的结果转换成 Verilog 所表示的格式，则特别容易出错。而且，这种错误很难查。另外，有些芯片的 spec 中，给定许多有规律的表格，例如列出在什么什么条件下，某某该取值多少。如果把这些表格全部手工转换成 Verilog 的程序，也是既繁琐，又容易出错的事情。

采用 PERL 编程的办法，则很容易解决这个问题。用 PERL 程序自动生成想要的数据，或者对给定的表格进行处理，然后其他 Verilog 源程序引用该结果。其好处：

- 不容易出错：如果有错误，则只需要改程序，不需动源码。
- 良好的数据一致性：用程序生成的数据可以保持一致性。要纠错/变更则统一改变，不会留下死角。
- 自动的数据转换：程序可以输出符合 Verilog 要求的任何数据表示法，不需要人工转换。
- 自动的数据取整：可以根据需要对结果进行取整操作：四舍五入、上取整、下取整等等。
- 自动位宽计算：程序可以自己计算或者指定所需的总线位宽，不需要人工进行计算。
- 调用方便：程序可以自动生成进行调用的语句，调用的源程序只用将该段 Copy/Paste 则可。
- 批量处理：同一项目中类似的东西只需要多次运行程序，不需要一点一点地改。
- 可重复利用：版本更新、算法改变、或者支持不同的配置，只需在程序上作轻微的改变，并作为选项则可。

作为一个良好的编程习惯，对于"数据"来说，用 function 进行描述是最好的办法。这样，PERL 程序可以生成一个个独立的 function，然后由主控程序用 include 进行引用。

4.2 自动生成数据的例子

设想有一个 ASIC 芯片，它的一个信号的生成，需要 7 个选择条件，而且对于不同的条件组合需要不同的运算(具体的逻辑略)。采用 PERL 编程的办法，可以生成数据如下：

```
//
// file: f_init_pos.in
//
// included verilog source file for:
// "init_pos  (initial position for light in light_pos
())"
//
// generated by "data_gen.pl,  version 1.1a",  at "
Mon Jan 12 21:00:12 2004"
//
// === don't edit this file manually ===
//


//
// needed bits: 5-bits
// given  bits: 5-bits
//

/* how to call this function:
    wire[4:0] temp;
    assign temp[4:0] = f_init_pos(slt1, slt0, slvt,
slwt, slmt, slkt, slft);
*/


//
// cases generated from instruction "table_init_pos"
//

function [4:0] f_init_pos;
    input     slt1;
    input     slt0;
    input     slvt;
    input     slwt;
    input     slmt;
    input     slkt;
    input     slft;

    casez ({ slt1, slt0, slvt, slwt, slmt, slkt, slft }) //
synopsys full_case
        7'b0000??0:   f_init_pos[4:0] = 5'd23; //
type1_case0
        7'b0000??1:   f_init_pos[4:0] = 5'd23; //
type1_case1
        7'b0010??0:   f_init_pos[4:0] = 5'd29; //
type1_case2
...
        7'b1101??1:   f_init_pos[4:0] = 5'd3;      //
 type4_casej
        7'b11110?0:   f_init_pos[4:0] = 5'd7;      //
 type4_casek
        7'b11110?1:   f_init_pos[4:0] = 5'd8;      //
 type4_casel
        7'b11111?1:   f_init_pos[4:0] = 5'd12; //
type4_casen
    endcase
endfunction
```

程序的有效之处不在于生成了上述的一个文件，而在于一次生成将近 100 个类似的文件。

5.PERL 自动生成仿真激励

5.1 自动生成仿真激励的必要性

对于 IC 设计来说，仿真是至关重要的。如何为 test bench 提供激励，是仿真时的一大难题。大多数比较复杂的芯片，在其内部都会有一个寄存器阵列。该阵列中的数据被分成一个个位宽不等的寄存器，由这些寄存器来控制该芯片的工作。

对于 Verilog 仿真器来说，它所看到的只是一比

特一比特的数据。激励编制者只能给 Verilog 提供十六进制或者二进制的数据。如果都是手工编制，由于寄存器都有自己的定义，位宽各不相同，这样的情况下，手工编制特别烦、累，而且极易出错。

对于作为激励的数据文件来说，如何让读者搞明白：哪些是缺省值，哪些是设定的值，哪些值是相关的等等也是很重要的。而且对于不同设置，某些值的缺省值可能是变的。如果用手工编制各种不同的激励数据，这些也是很难办的。做到这一点有以下的办法：

- 文件名中体现
- 在激励文件中增加注释
- 在其他文件中增加对文件的说明

这对于手工作业来说，更是增加了困难。人工写激励的问题有：

- 增加可读性则增加了额外的工作量
- 很难保证激励数据、注释和文件名的一致性
- 对于复杂的设置来说很难遍历，容易产生 case 的遗漏
- 难以维护，设置稍有变化，则需改数据本身，数据的一致性难以保证
- 不可复用，对于版本的不同和设置变化，所有的数据都必须重新写

采用 PERL 编程序的方法，则可很方便地解决这个问题。可以：

- 将缺省数据放在程序里
- 数据的相关性由程序自动完成
- 将改变的寄存器设置作为程序运行的选项
- 由程序提供别名(alias)，把某些设置集用一个别名代表

为了增加激励数据文件的可读性。程序除了输出必要的数据之外，还做如下工作：

- 所有的设置反映在文件名上
- 将当前所有的缺省值显示在激励数据文件的注释中
- 将当前设置列在数据文件的注释中
- 将所有的别名对应的设置列在数据文件的注释中
- 自动生成一个列表文件，表明生成的文件名和对应的设置
- 输出文件生成时间，和 PERL 程序的版本号

5.2 自动生成仿真激励的例子

假设在一个 ASIC 芯片中有一个寄存器阵列，可以用<8-bit 地址><8-bit 数据>的方式进行设置。这个寄存器阵列被分成比特数不等的 60 多个寄存器，8-bit 地址并没有全部使用，如果要设置这些寄存器就必须一比特一比特地设。为了仿真的高覆盖率，遍历这些寄存器是必须的，但也是很困难的。

采用 PERL 编程的方法，上述难题则不再是难题。因为篇幅限制，这里只列出寄存器阵列中的一小部分，遍历的使用并未列出。PERL 程序名为 pat_gen.pl，如下：

```
% pat_gen.pl
% pat_gen.pl vloc=110011
% pat_gen.pl vloc=110011 pson
```

上述三个命令分别输出三个文件：

```
* default.dat
* vloc110011.dat
* vloc110011_pson.dat
```

对应文件的内容如下：

(1)default.dat：

```
// CHIP register array setting data file
// created by "pat_gen.pl, version 1.2b", at "Mon Jan 12 22:02:59 2004"
// === don't edit this file manually without permission ===
//
// defined setting list
//
// ] vloc uses the default value: vloc = "011100"
11000000_00000000
```

```
01110001_01101110
00011010_11001111
00011011_00011100
```

(2)vloc110011.dat:

```
// CHIP register array setting data file
// created by "pat_gen.pl, version 1.2b", at "Mon Jan 12 22:03:05 2004"
// === don't edit this file manually without permission ===
//
// defined setting list
//
// ] vloc is set: vloc = "110011"
// ] "vloc"              ==> "110011"
11000000_00000000
01110001_01101110
00011010_11001111
00011011_00110011
```

(1) vloc110011_pson.dat:

```
// CHIP register array setting data file
// created by "pat_gen.pl, version 1.2b", at "Mon Jan 12 22:03:08 2004"
// === don't edit this file manually without permission ===
//
// defined setting list
//
// ] vloc is set: vloc = "110011"
// ] "vloc"              ==> "110011"
// ] "psave"             ==> "1"        using alias "pson"

11000000_00000001
01110001_01101110
00011010_11001111
00011011_00110011
```

6.PERL 用于仿真输入输出文件处理

6.1 仿真输入输出文件处理的必要性

在对芯片进行仿真的后期，需要对实际的情况进行批量的仿真。有些时候是拿一个实际的例子，设法将例子的数据转换成仿真器可以接受的格式，然后启动仿真器，再设法把仿真器的输出转换成例子所能接受的格式。

实际的数据都需要进行转换才能够被 Verilog 仿真器所使用，这种转换被称为 b2a（二进制转成 ASCII）。Verilog 仿真器本身可以输出的只有 ASCII 格式，也必须进行转换才能进一步使用，这种转换被称为 a2b(ASCII 转成二进制)。一般没有现成的工具可以做 a2b 和 b2a 的工作。

为了兼容不同的平台，例如：Windows 和 Unix，有时不得不在两个平台间交换数据，这需要进行 Unix 和 DOS 之间文本格式进行转换。尽管 Unix 平台上有 dos2unix 和 unix2dos 的工具，但是 DOS 平台上却没有现成的工具，这也是一个问题。

对于有些以“块”为处理单位的芯片，在将数据送入仿真器之前，必须先进行“分块”。而在仿真后，必须将仿真结果的小块“合并”起来。

有些情况下，批量仿真实际上希望变成“基于仿真器的演示系统”。既要能够看到所有的输入，又得能搞看到所有的输出，还得能够将输入输出进行比较。另外，在需要的时候还应该可以查看所要检查内部信号的波形。

如果上述所有的工作都用手工完成，简直是不可能的。这也是大多数 ASIC 设计中，无法进行真正的批量仿真的原因。而用 PERL 实现一个“基于仿真器的演示系统”是很现实的。为了可重用、通用，并且便于编制，按照 Unix 的风格，可以用若干个小工具的集合完成这样一个演示系统。

6.2 基于仿真器的演示系统的例子

在一个 ASIC 芯片设计中“基于仿真器的演示

(下转第 31 页)

终定局,然后依据这些模块进行最后的布线。物理验证则是在每一次布线后都需要进行。当所有的模块都最后定局以后,将他们全部输入并存储在数据库内,并且在最顶层重新进行验证。在进行最后的验证时,数据库内已经具有全部所有模块级的描述。这时,应进行最后的 RCX(RC 参数提取)操作(同时也同样需要将结果输送给 AMS 流程以便进行验证时应用),同样,也需要进行最终的供电网络分析和电迁移分析。

芯片的最后整理工作是为了使数据库能够为磁带交出准备好必须的补充修正。这些修正任务是依赖整个数据库运行的,并且经常由于规模巨大的数字部分内容存在使得数字部分成为一个规模巨大的实体。有时设计需要在此层次上进行一些微小的修改(例如改变某些金属连线)。另外在磁带交出以前,由于需要加入划片线,增加商标标记等任务因此需要访问整个芯片的数据库。最后,还需要进行金属化层的填充,或者按照代工厂商的规定要求进行一些加工;然后才能够交付磁带。

Virtuoso 芯片集成设计流程的运行需要依靠整个 Virtuoso 设计平台的工具系统,这些工具系统包括:Preview 选项,Virtuoso Chip Assembly Router, Virtuoso Analog VoltageStorm 供电网络设计选项, Assura DRC/LVS/RCX 验证套件,以及 Virtuoso Chip 编译器等。此外,Virtuoso 芯片集成设计流程还需要依靠 OpenAccess 功能。OpenAccess 功能使得该设计流程能够处理巨大规模的混合信号设计。

5. 结论

Virtuoso 芯片集成设计流程是 Virtuoso 设计平台的一个重要组成部分,它能够以全定制的方式对不同门类的电路成份进行全面的集成。它和其它解决方案,例如 Encounter 设计平台,的多方面的密切配合使用,使得设计能够针对不同的设计任务选择使用不同的,所需要的正确的解决方案。Virtuoso 芯片集成设计流程可以用来满足,从宏单元的开发,到整个混合信号芯片集成的各种各样设计任务的需求。CIC

上接第 41 页

系统"可以包括如下的 PERL 程序:

- a2b.pl
- b2a.pl
- dos2unix.pl
- unix2dos.pl
- pat_gen.pl
- action_gen.pl
- batch_run.pl

batch_run.pl 是主控程序,在适当的时候调用上述工具集里的工具。它自己也需完成如下功能:

- 调用仿真器
- 进行原始数据分块处理
- 进行结果数据合并处理
- 建立必要的目录
- 调用其他应用程序

结论

PERL 是一个十分有用的语言。IC 设计是一个复杂的过程,在 IC 设计流程中采用 PERL 语言编制适当的工具,可以大大加快整个工作流程,减少人为的错误。CIC

参考文献

[1] vpp - 基于 PERL 的 Verilog 语言扩充

[2] Clinton Pierce: "Sams Teach Yourself Perl in 24 Hours"