第3章

基于FPGA的数字电路设计

3.1 高级描述语言编译和芯片版图生成流程

作为通用的软件计算和硬件设计平台，基于通用处理器的软件编译和基于FPGA的硬件设计过程有很多相似之处。早期简单的软件程序，可以直接采用汇编语言来编写。随着程序规模的扩大以及对源代码的可读性、可维护性和可重用性的需求，我们必须借助于软件编译器来自动实现从高级语言描述到汇编语言和二进制可执行程序的转换过程。软件编译器就是把用户用高级语言描述的源代码翻译成目标处理器上可以运行的二进制可执行程序。对于FPGA编译器来说，它也是把用户输入的RTL描述源代码翻译成可以下载到FPGA器件的位流文件，从而配置成用户所需要的电路的实现方式。因此，我们先讨论这两种通用计算方式的特点，然后基于熟悉的高级语言软件编译流程来理解FPGA的位流文件产生过程。

3.1.1 基于通用处理器的软件编译流程

经典的通用型处理器和FPGA都具有现场可编程性，因此它们都属于通用的计算平台，但是两者的计算效率是不同的。对于经典的通用处理器来说，计算程序的复杂度主要表现在程序所需要的存储器空间和运行时间。对于给定的硬件资源，包含微处理器和存储器，只要它们能够一直保持供电状态，那么从理论上来说所能运行程序的复杂度主要受机器运行寿命的限制，即越复杂的计算则需要更多的运行时间。为了提高计算效率，从而减少计算程序的运行时间，通用处理器应采取越来越复杂的流水线结构和更高的时钟频率。存储器的容量也急剧扩大，供微处理器在更短的时间内访问更多的数据。近年来由于电路工作的时钟频率已经逐渐趋近物理极限，并且电路的功耗成为比时序性能更优先考虑的因素，通用处理器开始采用多核结构，即增加硬件上的并行性(parallelism)和并发性(concurrency)来提高计算效率。如多核CPU和GPU都是典型硬件并行化的例子。另一方面，对于通用的FPGA计算平台来说，器件内部的各种硬件资源，都是可以并行工作的。相对于通用处理器来说，虽然FPGA要比通用微处理器难用很多，并且用户还需要掌握硬件设计的基本知识，但是同一个计算任务一般在FPGA上的计算效率要比通用微处理器高几十倍左右，尽管FPGA电路的最高时钟频率一般要远远低于通用处理器的最高工作频率。由于硬件资源规模的有限性，即一个FPGA器件不可能具有无限多的可编程逻辑资源，有些计算任务在给定的FPGA器件上就无法运行。因此，为了进一步提高计算能力，一块FPGA芯片上具有的芯片规模基本上是按照指数规律增大。这种硬件资源规模增大的

速度和2004年之前处理器时钟频率的提高速度基本上都遵循摩尔定律。另外，系统级FPGA芯片还要嵌入更多的专用硬件资源，用于进一步提高粗粒度计算的效率和晶体管的资源利用率。

因此，从可计算性和计算效率的角度来看，通用处理器和FPGA所具有的可编程性的性能有以下区别。

(1) 对于通用处理器来说，单核处理器的计算任务大体上还是基于串行的工作方式，即在同一时刻处理器按照串行的方式执行指令，因此一个复杂的计算任务相对就要耗费较多的计算时间。通用处理器硬件资源本身除了器件使用寿命外，并没有对运行时间提供其他限制。但是对于FPGA来说，计算任务是在各硬件资源上并行执行的，计算任务的可计算性主要取决于器件的硬件资源规模。这就是说，FPGA的硬件资源规模基本上就约束了计算任务的复杂度。第6章所讨论的可重构方法是在时间上重新利用FPGA的可编程资源。这种方法为突破FPGA硬件资源规模的限制提供了有效途径。对于静态可编程的非可重构FPGA来说，硬件资源规模束缚了计算任务的规模。

(2) 对于通用处理器来说，计算效率可以用在给定的硬件资源及其计算性能下计算程序所需要的运行时间来衡量。计算效率的优化主要是利用软件的方法进行。FPGA的计算效率主要取决于在确定器件下电路工作的最高时钟速度。

对于单核通用微处理器来说，相应的软件编译和运行环境已经非常成熟。如GCC(GNU compiler collection，GNU套装编译器)工具和Linux操作系统就能支持多种CPU结构。但是对于通用的FPGA器件来说，不管编译工具还是运行环境，目前远没有达到由统一的后端布局布线工具支持不同公司FPGA器件的水平。对于高级语言编译器来说，一般分为前端分析和后端综合两大部分[1]。例如，对于常用的GCC工具来说，它的前端部分支持多种语言：C、C++、Java、Fortran、Ada；后端综合部分支持多种处理器，例如Intel系列处理器、ARM、PowerPC等处理器结构。如果今后需要支持一种新语言，只要在前端分析部分增加对这种语言的前端分析即可，后端综合部分都是可以共享的。另一方面，如果需要支持一种新型处理器的编译功能，只需要在后端综合部分添加即可，前端分析部分完全可以共享。

如图3-1所示，编译器首先读取源程序代码，进行词法、语法分析。词法分析就是把代码中的各个字符串转换为单词，包括高级语言的关键字和用户定义的标识符等。语法分析就是把经过词法分析后的各个单词按照高级语言所定义的语法规则构成合法的树结构。如果输入源程序没有不认识的单词和语法错误，那么编译器就由输入源程序得到对应的语法树结构，再经语义分析，确定源程序的含义。经过语义分析后，语法分析所生成的树结构中的节点就赋予了程序的语义信息，即编译器理解了程序的行为和功能。如果源程序没有语义错误，那么就可以生成中间代码的表达方式。以我们日常所用的自然语言为例，“学生”、“是”、“我”都是合法的单词，但是这些单词的不同组合，“我学生是”是有语法错误的表达方式，“我是学生”才能正确地表达语义。一般来说，这种正确表达语义的中间格式(intermediate representation)是以RTL(register transfer language，寄存传输语言)来描述。对于RTL语言的具体格式定义，可以参考GCC的在线文档[2]或GCC参考文献[3]。对于GCC工具来说，命令行“gcc -dr 输入文件名”就可以产生对指定输入文件转换得到的RTL中间输出文件，其中命令行参数“-dr”表示输出RTL的调试(debug)信息，即分别取自RTL和debug的首字母。默认情况下，输出的文件扩展名为.rtl。

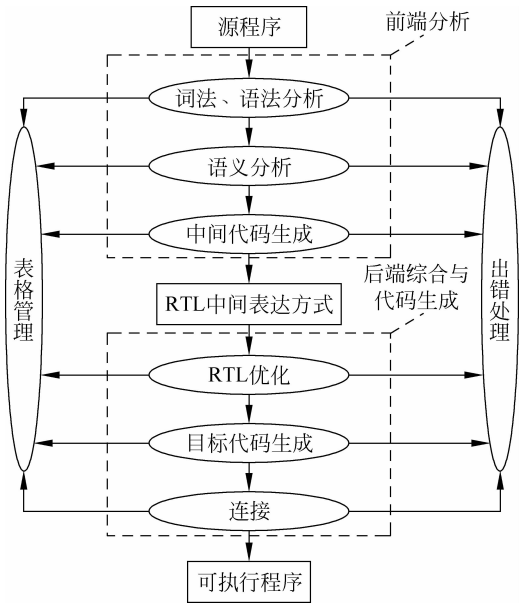


图 3-1 高级语言编译流程[1]

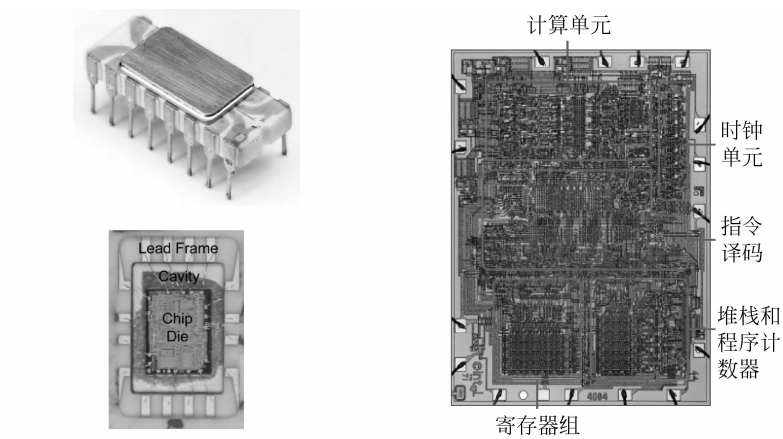
源程序经过前端分析得到 RTL 中间表达方式后，后端综合阶段就基于这种 RTL 中间表达格式和处理器的硬件结构信息，进行 RTL 优化，并生成能够在目标处理器上运行的二进制代码。RTL 优化过程包含和处理器无关的树结构优化，以及和具体处理器结构相关的优化。一般来说，输入源程序会调用一些库函数，或者大规模源程序一般由多个源文件来描述。因此多个源文件所转换到的目标(object)文件就需要连接成一个可执行(executable)程序，从而可以确定各个目标文件之间的函数调用关系及其函数调用所需要的首地址。对于 C/C++语言来说，这个可执行目标程序只包含一个 main 函数。在处理器运行时它就可以从 main 函数被装载到内存后的首地址开始运行指令。如果程序是在操作系统的管理下运行的，那么在用户程序的 main 函数前一般还会连接操作系统提供的引导程序。因此，虽然 C/C++等语言规范是和操作系统无关的，但是编译器一般和操作系统相关。同一个编译器在不同的操作系统下也需要提供不同的函数库，以保证用户程序的正常运行。

在高级语言的编译过程中，还需要从源程序中提取并保留各个变量的名字、用户定义类型的名字、函数的名字、参数个数和返回类型等函数原型信息。编译程序就把这些名字和类型等信息记录到符号表中，方便各源文件的查找调用，或者在程序连接时验证检查，抑或是在调试程序时方便找到声明或引用某符号的行号等信息。在编译过程中符号表的查找是非常频繁的，因此就需要编译程序能够有效地管理这些符号表格。这就是图 3-1 中的“表格管理”功能。

在编译流程中各个功能模块出现的各种错误，例如前端分析流程中词法、语法分析模块检测到单词组成错误、括号不匹配或者其他语法错误，语义分析模块发现某运算符和它的运算对象类型不符等都需要输出合适的错误提示信息，帮助用户有效地修改源程序代码。因此图 3-1 中的“出错处理”需要判断并产生合适的诊断、警告和错误等信息，帮助用户快速定位代码中的错误和可能存在的危险，提高用户编写程序的效率。

3.1.2　基于 EDA 工具的数字电路设计流程

随着通用计算机软硬件技术的不断发展，它的应用领域不断扩大，逐渐渗透到我们生活和工作的各个领域，从而改变了我们的生存和生活方式。一开始数字电路基本上是基于晶体管，利用笔和纸的方法进行手工设计，这就是早期所谓的晶体管-晶体管逻辑(TTL)电路。例如 1971 年 11 月 Intel 公司发行的世界第一块通用处理器 4004，位宽为 4，能够满足当时计算器的功能需求。图 3-2(a)所示为双排 16 管脚封装的 4004 芯片。去除封装材料后，可以清楚地看出芯片裸片四边各有 4 个引脚。图 3-2(b)所示为芯片内部的版图，它共集成了 2300 个晶体管。随着芯片集成度和规模的扩大，一般超大规模集成电路会集成高达上亿个晶体管，因此基于晶体管的手工设计方法已经无法满足需求。这就好像只用汇编语言无法编写大规模的软件程序一样，数字电路的设计需要计算机辅助设计方法，这样才能设计更高集成度和更大规模的芯片。硬件电路设计水平的持续发展，也有利于设计出更高性能的通用处理器，从而提高了软件程序的运行效率，另外也对电子设计自动化软件提出更高的需求。因此，硬件电路和设计自动化软件两者相辅相成，是集成电路在过去近半个世纪以来基本上按照摩尔定律所预测的那样呈指数发展的一个重要基础。



(a) 封装后的芯片和去除封装材料的裸片　　(b) 芯片内部版图

图 3-2　Intel 四位通用处理器芯片 4004 及其版图[4]

从当前数字集成电路产品的角度来看，主流的方法包括芯片设计和芯片制造两个基本独立的步骤。这种设计和制造的分工可以提高芯片的生产效率并能降低成本，因此芯片设计者的主要目标是如何设计出制造芯片所需要的电路版图，它包含了电路中所有的晶体管位置、大小和相互之间复杂的连接关系。GDSII 文件格式是芯片版图信息描述的工业界标准，芯片制造商可以根据 GDSII 文件提供的信息恢复版图形状并制造出所需要的芯片裸片，再经过封装、测试就可以作为合格的产品销售。如第 1 章所述，集成电路产品主要经过芯片设计、芯片制造和封装测试三个技术环节。本节主要讨论的是芯片设计技术，它又可分为前端综合设计和后端布图设计两个部分，如图 3-3 所示。前端综合设计主要是读入电路的 RTL 描述，或者是电路原理图的描述，由逻辑综合 EDA 工具输出针对一个半导体工艺

库优化的门级网表。后端布图设计主要是输入前端产生的门级网表，确定版图布局规划(floorplan)，由布局布线 EDA 工具输出针对同一个工艺库的版图文件。为了保证布图后的电路和前端综合后的网表具有相同的功能并且达到预期的时序性能，还要对布局布线后的电路运行后端时序仿真、静态时序分析和其他的验证过程。另外考虑到芯片功耗、热点(hot spot)对电路性能和寿命的影响，深亚微米集成电路还需要运行功耗分析和热分析等 EDA 工具。最后在产生 GDSII 文件前，对芯片版图做设计规则检查(design rule check，DRC)，确保芯片版图满足指定工艺的电气和物理要求，例如电源线的驱动、金属线的宽度和间隔要求等。只有布图软件产生的版图通过了后端时序分析、功耗分析、LVS(layout versus schematics)版图验证、DRC 检查后，所产生的 GDSII 版图文件才可以递交给芯片制造商生产。由于 FPGA 和专用芯片的后端设计目的、工具链和基本方法差别较大，因此本书主要介绍专用芯片的前端设计过程。它和 FPGA 的前端设计比较类似。对于专用芯片的后端设计流程，有兴趣的读者可以参考文献[5]。

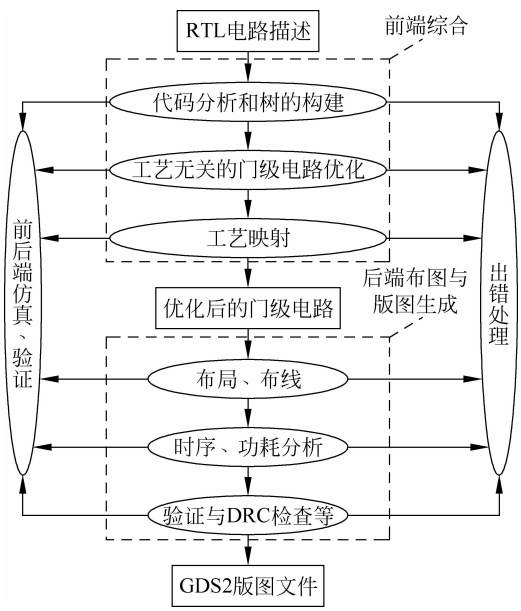


图 3-3 基于 EDA 工具的数字集成电路设计流程

当数字电路硬件工程师需要设计一款芯片时，一般采用硬件描述语言(HDL)来描述所需要的电路功能，然后由各种 EDA 设计工具进行仿真、综合和布图得到芯片制造所需要的版图文件。以 Verilog 硬件描述语言为例，一般在电路设计过程中包含以下三个层次[6]。

(1) 行为级或算法级：这是 Verilog 语言最高的抽象级别。相比较而言，VHDL 语言在行为级描述方面比 Verilog 语言具有更强大的功能。但是 Verilog 仿真器的速度一般要比 VHDL 仿真器快很多。在国内 Verilog 语言比 VHDL 语言的使用普及率要高。这个级别所描述的主要是算法行为，而不关心具体的硬件实现。这种描述方式和 C 语言编程很类似。各种循环语句，例如“while”、“for”和各种算术逻辑运算符是行为级描述的典型用法。这个层次有时也称为系统级描述。

(2) 数据通路级：设计者先设定基本的数据通路(data path)结构。数据通路确定了触发器和寄存器的数据传输方式或基本的流水线结构。为了实现数据在各个触发器或寄存器之间的变换和传输，就需要和数据通路配合的控制流程(control flow)。在这个层次的描述中就包含了时钟节拍及其寄存器的时序信息。因此这个层次的代码经常被称为 RTL (register-transfer level，寄存传输级)描述。

(3) 门电路级：电路描述中的各个模块直接确定了逻辑门(不是逻辑运算符)及其相互之间的连接关系。常用的逻辑门有 and、or、xor 等。用门级描述的电路实际上就是硬件电路的实现，它需要设计者已经熟悉硬件电路的构建。相对算法级和数据通路级来说，门级电路描述是偏向硬件层次的底层描述方式。

一般来说，高层次的描述方式灵活性较大，但是所得到的电路性能相对较差。除非用户对硬件电路设计非常熟悉，一般我们不采用门级描述来实现数字电路。对于 Verilog 语言来说，以上三个层次的描述也不是非常严格，并且它的行为级描述功能不如 VHDL 硬件描述语言强大。在混合采用以上三种描述层次时，我们有时也把行为级和数据通路级的混合描述方式统称为 RTL 描述，相应的电路结构如图 3-4 所示。这个图和第 2 章的图 2-7(a)是一致的。输入数据经过各组合电路的变换后在寄存器之间传输，直到输出所需要的变换结果。当输入数据只有一位时，那么图中的存储单元就是一个触发器。寄存器一般表示多位数据的存储，例如 8 位、16 位、32 位存储器等。多位寄存器之间的数据传输就通过一组连线连接。在 RTL 描述的电路中，输入数据在同步时钟信号的作用下，经过多级的组合逻辑电路转换。中间数据转换结果放到寄存器中，最后的转换结果由输出端得到。当然，输入和输出也可以增加寄存器，可以保证和前后级电路的同步。对于复杂的数字电路来说，可能包含多个时钟和多种连线或者总线结构。目前主流的工业标准 RTL 硬件描述语言是 Verilog 和 VHDL。Verilog 语言借助了很多 C 语言中的语法和语义，入门比较容易。VHDL 语言在语法和语义上的要求要比 Verilog 更为严格，比较适合于更高层次的系统级电路描述。本书中采用 Verilog 语言作为示例。

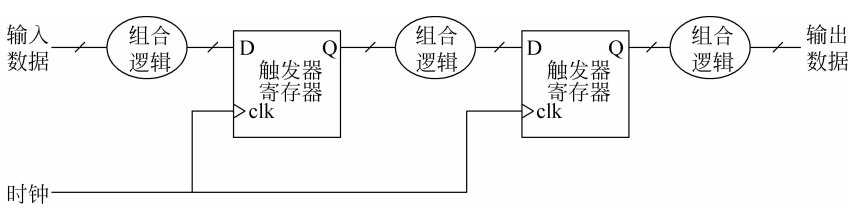


图 3-4　RTL 语言描述的数字电路基本结构

在图 3-3 的流程中，逻辑综合 EDA 工具首先读入用户的 RTL 描述文件，进行语法语义分析。如果代码分析没有错误，那么就进行电路结构的构建(elaboration)，得到类似于图 3-4所示的数据通路或者控制逻辑电路结构。一般来说在编译程序内部电路结构是用树的形式来表达，其中树的节点表示各种逻辑和算术运算，树的边表示各种运算之间的输入输出关系。在构建 RTL 电路结构时，主要是基于模板(template)的方式把 RTL 代码中的各种描述结构转换成逻辑电路，即编译器会识别代码中的关键字和一些固定的描述模板，从而生成优化的电路结构。

图 3-5 给出了三种基于 Verilog 语言的电路描述 RTL 模板。如果图中各模块的输入输

出不只是一位，而是多位的总线或者是数组形式，那么转换后的电路就是一位电路的重复复制。对于时序部分，边沿触发的触发器一般通过“always @(posedge 信号名)”来识别，如果数据位宽不只是一位，那么综合工具就会把它转为寄存器结构，其中触发器或者寄存器的输入信号可以由复杂的组合逻辑电路产生。最常用的组合电路可以通过 assign 语句来识别。在图 3-5 中，assign 语句可以由等价于 C 语言中的问号表达式、if-else、case 或者逻辑运算符来实现，它们都会生成右边的数据选择器或者更一般的逻辑电路。一般来说，右边的逻辑电路都是由简单的基本逻辑门构成。如果要利用一些已经优化的 IP 核，那么在代码分析时就要识别这些 IP 核，避免使用基本门电路来实现这些 IP 核的功能。这些组合逻辑电路可以通过输入输出级联进一步构建更复杂的组合电路。同时复杂组合电路的输出又可以输入到寄存器的输入端，从而构建出复杂的时序电路，满足系统应用的需求。对于 Verilog 和 VHDL 语言，两大国际标准 IEEE 1364.1 和 IEEE 1076.6 分别定义了可综合的 RTL 语言模板。通过标准中所定义的模板，逻辑综合工具在构建的时候就可以产生初始的门级电路结构，供后续作电路优化。

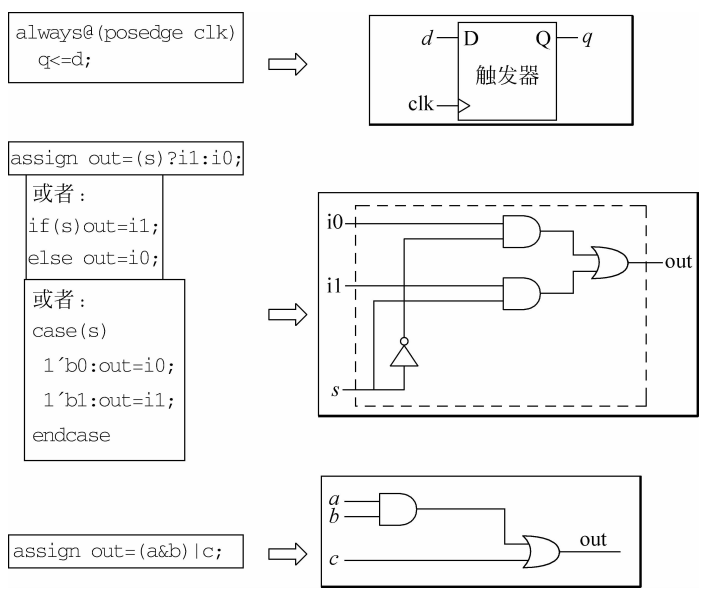


图 3-5 逻辑综合工具会把固定的 Verilog 描述模板转换为相应的电路结构

例如，图 3-6 是美国 IEEE/ACM 组织于 2002 年在 IWLS(international workshop on logic synthesis)国际会议上发布的一个基准测试电路描述。这个电路很简单，名称为“shiftreg_synth”，包含两个输入、一个输出，其中一个是时钟输入端。这个电路 Verilog 代码中的“\”类似于 C 语言中的转义字符，表示从它开始到空格字符为止中间的任何字符串都构成合法标识符。例如“\[11]”表示“[11]”是合法标识符，而不是作为总线或数组中的一位。另外，这个电路共有三个存储单元，初始值为“0”，每次时钟上边沿存储新数据。这些新数据是由一些组合逻辑计算得到。这个 Verilog 代码经过逻辑综合工具进行代码分析后，根据如图 3-5 所示的模板规则，就构建得到如图 3-7 所示的初始电路结构。注意到 Verilog 代码中的各种逻辑运算都直接转换为相应的逻辑门。三个一位存储单元转换为三个触发器 v1、v2、v3。图中标出了代码中给定的输入输出、线网、存储单元名字。因此，这个初始电

路结构和 Verilog 代码之间存在着很清楚的对应关系。但是，并不是所有的 Verilog 代码都可以转化为硬件单元。例如代码中的“initial”语句设定各存储单元的初始值。而在硬件触发器中，这个初始值在硬件上电后可能是随机值。因此像 RTL 代码中“initial”、“time”、“real”等语句都只是用于仿真，而不能构建或综合成硬件单元。对于图 3-6 中的 Verilog 描述，如果要初始化存储单元值，那么可以添加一个“reset”输入端进行复位。另外我们可以注意到图 3-7 中的电路具有图 3-4 所示的典型 RTL 逻辑电路结构。

```
module \shiftreg_synth(v0,clk,\v4.3);
input v0,clk;
output\v4.3;
reg v1,v2,v3;
wire \[11],\[13],\v4.0 ,\v4.1,\[9];
assign
   \[11] = \v4.1,
   \[13] = v0,
   \v4.0 = (~v3&v2)|(v3&~v2),
   \v4.1 = (~v3&v0)|(v3&~v0),
   \v4.3 = v1,
   \[9] = \v4.0;

always @(posedge clk)
begin
  v1 = \[9];
  v2 = \[11];
  v3 = \[13];
end

initial begin
  v1 = 0;
  v2 = 0;
  v3 = 0;
end
endmodule
```

图 3-6　IWLS 2002 测试例子 Verilog 代码[7]

在图 3-3 所示的流程中，RTL 代码经过构建后就得到一个初始的逻辑电路，接下去就对这个电路进行逻辑优化，即在保持电路的输入输出功能不变的前提下，根据布尔代数性质，尽量减少电路内部的逻辑单元数目、输入输出延时、功耗等性能指标。这个步骤一般称做“工艺无关(technology independent)的逻辑电路优化”。在学术界，AIG (and-inverter graph)图是一种非常有效的工艺无关电路的表达方式，它可以表达任意复杂的组合逻辑功能。由于 AIG 表达法的简洁性，它可以很有效地进行各种逻辑变换和优化操作。例如，图 3-8(a)是利用 UC Berkeley 的 ABC 开源工具根据图 3-6 中的 Verilog 描述所生成的 AIG 图。和图 3-7 相比，图 3-8(a)同样包含三个触发器：v1L、v2L 和 v3L，其中触发器的名字后缀“L”在 ABC 中取自 Latch 的首字母，意为时序触发器。但是电路中的组合电路都是用 AIG 的形式，其中节点表示一个 AIG 运算，连线表示各运算的连接关系。实线表示直接相连；虚线表示取反连接，即 AIG 输出经过取反后再连到下级门的输入端。注意到时钟“clk”信号虽然连接到时序存储单元的时钟端，但是在 AIG 中是悬空的。这是因为在芯片设计中，时钟信号直接影响到电路的工作速度。时钟信号在版图中一般和其他逻辑信号是分开处理的，这样可以保证时钟信号能准确地到达各存储单元的时钟输入端，避免产生较大的时钟偏差(clock skew，有时也称为时钟偏移)和时钟延迟(clock latency，有时也称时滞)。对于专用芯片来说，在后端的布图软件中，电路布局后

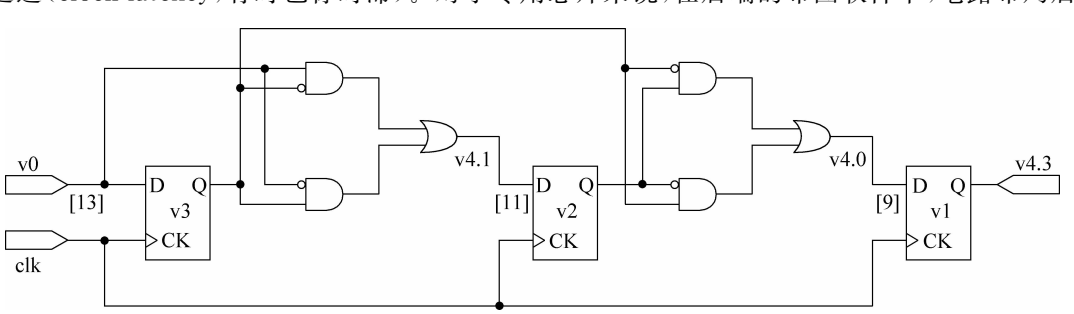
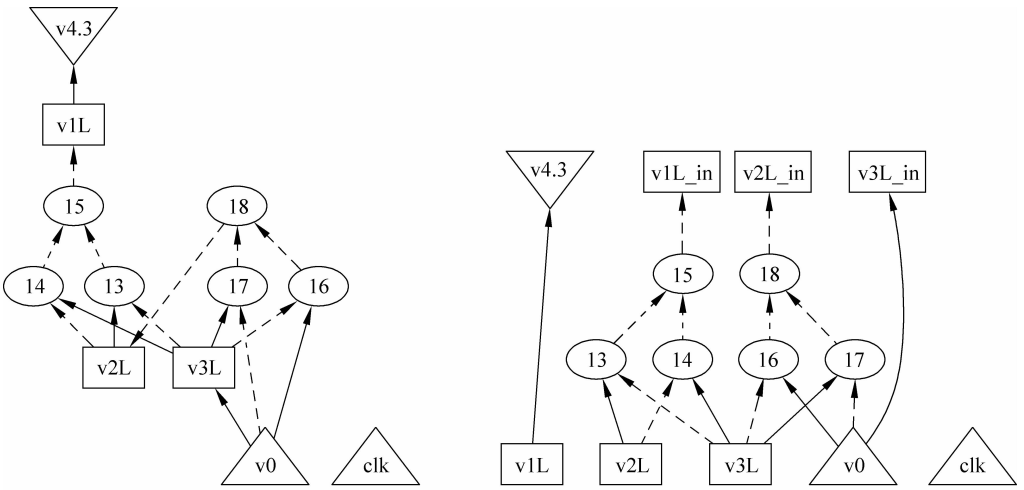


图 3-7　由逻辑综合工具构建的初始电路结构

一般有专门的时钟树综合工具(clock tree synthesis)保证产生最佳的时钟树结构。在FPGA芯片版图设计时,一般也有专门的金属层是给时钟树用的,这样就能保证得到所需要的时钟树优化结构。因此在逻辑优化时,可以暂时不考虑时钟树的互连连接。



(a) 时序电路的AIG表示　　(b) 纯组合电路的AIG表示

图 3-8　ABC 构建的 AIG 时序电路和转换后的纯组合电路

对于电路设计者和逻辑优化来说,触发器或者寄存器的位置和数目在逻辑优化前后一般不会改变。要达到这个目的,它们的输入端一般作为组合电路的输出端,它们的输出端作为下级组合电路的输入端。在优化完成后,再把所有的触发器或者寄存器放回到原来的位置。例如,图 3-8(b)是暂时拿去触发器的 AIG 图,其中 v1L_in、v2L_in 和 v3L_in 是新增的对应于三个触发器输入端的组合输出端;而 v1L、v2L 和 v3L 是新增的对应于三个触发器输出端的组合输入端。这种 AIG 图的优点是后续的优化过程只需要关注图中的纯组合逻辑部分,避免考虑时序单元的存在。在组合电路优化完成后,只要在 v1L_in、v2L_in、v3L_in 和 v1L、v2L、v3L 之间插入三个触发器就可恢复原始时序电路的功能。由于图 3-8 中的电路非常简单,因此工艺无关的门级电路优化并不能减少任何的 AIG 数目或者是输入到输出所经过的级数。因此对这个简单的电路来说,图 3-8(b)也就是优化后的电路结构。

根据图 3-3 的设计流程,优化后的门级电路需要进行工艺映射(technology mapping),就得到相对于某一个工艺库优化的门级电路,如图 3-9 所示。在工艺库中包含了芯片制造商针对某一种半导体工艺所能提供的各种逻辑单元和 IP 核。因此工艺映射的过程就是把工艺无关(technology-independent)逻辑优化后的逻辑电路映射到工艺库中的各种逻辑单元,从而针对具体的工艺库实现了用户电路功能。

对于图 3-8 中的时序电路,ABC 工具利用通用的工艺库进行工艺映射。通用工艺库只包含了基本的逻辑门,例如“与”、“或”、“非”、“异或”等逻辑门。ABC 工具工艺映射后的结果如图 3-10 所示,图中分别给出了 ABC 中“show”命令所显示的结果及其电路图形式。其中图 3-10(a)每个逻辑节点上方的标记表示对应的逻辑单元名字,buf、xor 分别表示缓冲器和异或门。每个节点内部除了节点标号外,还给出了函数信息。例如“01 1”和“10 1”这两行表示 xor 门,即两个输入变量输入相异时输出为 1。这样工艺映射后的电路就能实现

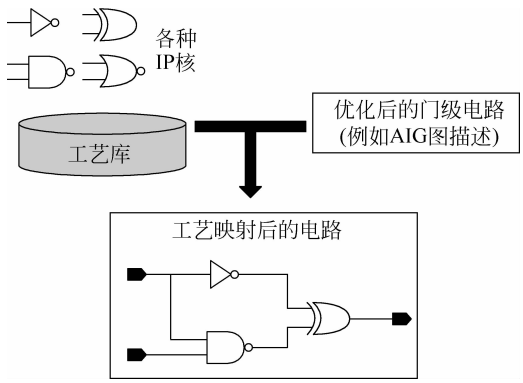


图 3-9　工艺映射基本流程

图 3-6 中 Verilog 代码描述的功能。由于缓冲器对电路的功能没有影响，因此在图 3-10(b)的电路图中并没有画出两个缓冲器。

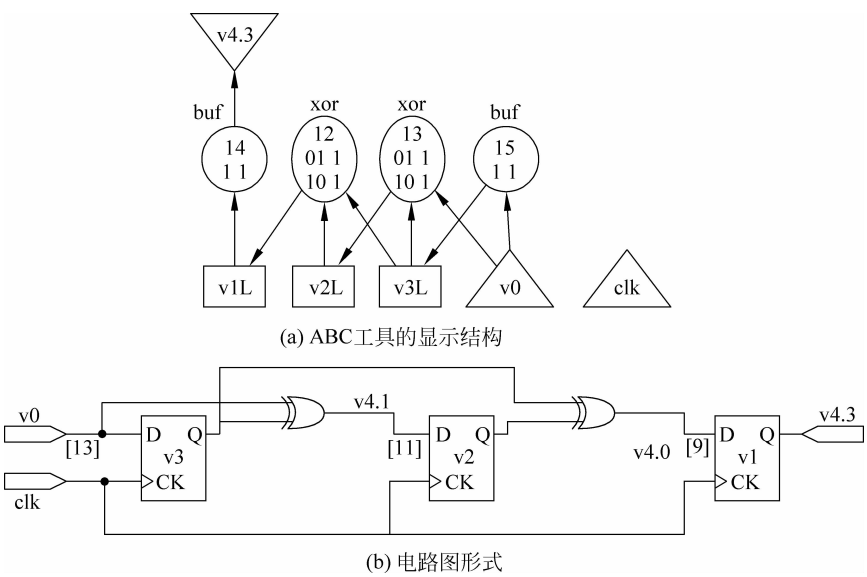


图 3-10　工艺映射后的电路

如图 3-3 所示，在得到工艺映射后的结果后，前端的综合流程就已结束。接下去就开始后端的专用芯片布图流程[5]。值得一提的是，如果我们比较图 3-1 和图 3-3 两个流程，可以发现如果能够连接软件编译器输出的 RTL 描述和 EDA 前端综合工具输入的硬件 RTL 描述，那么就可以直接从高级语言，例如 C/C++、Java、Fortran，直接生成高级语言描述的芯片版图，这样可以极大地提高芯片设计的效率——用户不需要学习硬件描述语言就可以完成芯片的自动化设计。不过这方面的工作一般称做高层次综合，目前还只是处于研究阶段，并没有非常成熟的自动化产品。有兴趣的读者可以参见文献[8]及后续大量关于高层次综合的研究论文。