HZ BOOKS
华章教育

电子与电气工程丛书

WILEY

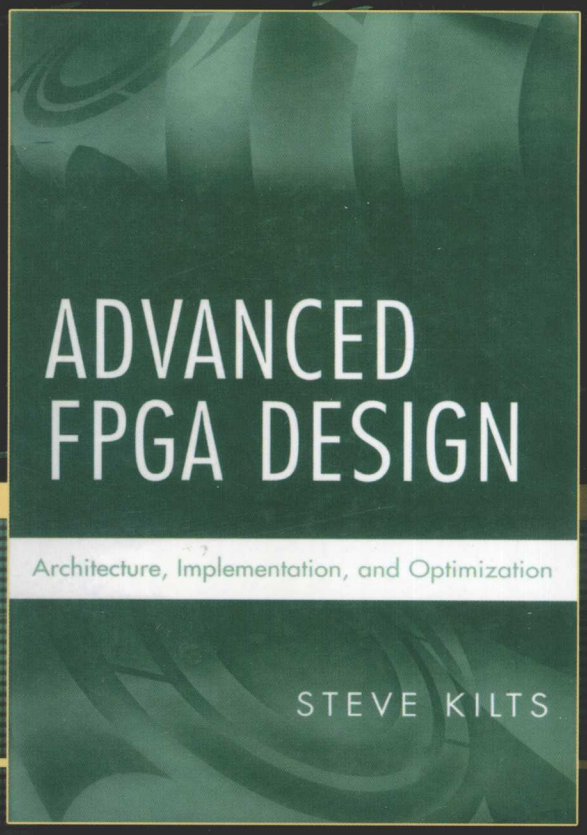
高级FPGA设计

结构、实现和优化

Advanced FPGA Design

Architecture, Implementation, and Opimization

（美）Steve Kilts 著
孟宪元 译



机械工业出版社
China Machine Press

电子与电气工程丛书

高级FPGA设计

结构、实现和优化

Advanced FPGA Design

Architecture, Implementation, and Opimization

（美）Steve Kilts 著

机械工业出版社
China Machine Press

本书主要讲解了FPGA设计、方法和实现。这本书略去了不太必要的理论、推测未来的技术、过时工艺的细节，用简明、扼要的方式描述FPGA中的关键技术。主要内容包括：设计速度高、体积小、功耗低的体系结构方法，时钟区域，实现数学函数，浮点单元，复位电路，仿真，综合优化，布图，静态时序分析等。

本书把多年推广到诸多公司和工程师团队的经验以及由白皮书和应用要点汇集的许多知识进行浓缩，可以帮助读者成为高级的FPGA设计者。

Steve Kilts: Advanced FPGA Design: Architecture, Implementation, and Optimization (9780470054376/0470054379)

本书版权登记号：图字：01-2007-4191

图书在版编目（CIP）数据

高级FPGA设计：结构、实现和优化/（美）克里兹（Kilts, S.）著；孟宪元译. —北京：机械工业出版社，2009.2

（电子与电气工程丛书）

书名原文：Advanced FPGA Design: Architecture, Implementation, and Optimization

ISBN 978-7-111-25547-5

Ⅰ. 高…　Ⅱ. ①克…　②孟…　Ⅲ. 可编程序逻辑器件－系统设计　Ⅳ. TP332.1

中国版本图书馆CIP数据核字（2008）第175385号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：曾　珊

北京慧美印刷有限公司印刷

2009年2月第1版第1次印刷

184mm×260mm · 15.75印张

标准书号：ISBN 978-7-111-25547-5

定价：35.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

译 者 序

FPGA 技术自 20 世纪 80 年代中期出现至今，经历了 20 多年的发展，正在得到越来越广泛的应用，包括各个领域的数字系统、数字信号处理（DSP）系统和嵌入式系统都会用到 FPGA 器件。与此同时，FPGA 也引起电子设计技术的深刻变革，FPGA 的可编程特性使得设计者可以在应用现场，利用计算机上自动化设计软件完成数字系统产品样机的设计、更新和调试，不仅缩短了产品的上市时间，而且也能适应技术标准和协议的更新和升级，延长了产品的寿命周期。

FPGA 技术的这些特点要求设计者的设计能力不断提高，除了自动化设计软件在不断完善和升级，还要求设计者有更丰富的系统设计知识、软/硬件的设计本领和面对实际问题的处理能力。为了满足广大 FPGA 设计者的上述需要，获得明尼苏达大学电子工程理科硕士学位的 Steve Kilts，作为 Spectrum Design Solutions 的共同创建者和主要工程师，他与他的团队已经成功地完成许多委托项目，作者具有丰富和广泛的 FPGA 设计经验，包括在音频、DSP、高速计算和总线结构、IC 测试、工业自动化和控制、嵌入微处理器、PCI、医疗系统设计、商用飞机和 ASIC 样机等广泛的领域。同时他在瞄准速度高、面积小和功率低的 FPGA 设计中进行性能的舍取方面积累了多年的经验。

“高级 FPGA 设计——结构、实现和优化”一书强调 FPGA 设计和实现中的高级课题，工程师和计算机科学家通过此书可以加速掌握 FPGA 设计的学习过程，由于强调实际的设计，逻辑和实践的方法，使读者可应对特殊的设计挑战，显著减少设计中的弯路，使读者增长和补充可行的经验，这些实用的参考包括：

- 说明每个课题的波形图和电路图
- 用 Verilog 程序说明典型问题的例子
- 给出大量应用的案例研究
- 每章结尾的小结

此书按照典型的设计流程来安排各章的次序。前几章讨论结构，然后是仿真，再是综合，接着是布图等。书中其余的章节是实例，作者选择 Verilog 作为硬件描述语言，选择 Xilinx 公司作为 FPGA 的销售商，选择 Synplicity 作为综合和布图的工具，书中覆盖的课题可以方便地映射到 VHDL 语言、Altera 销售商、Mentor Graphics 的工具，所以，即使对于使用其他技术的读者，此书仍然是有价值的。

对于为了获得高层次 FPGA 设计技巧的工程师和计算机科学家来说，本书是理想的。同时，本书可用来作为内行经验的参考，对电子工程和计算机科学的高年级学生和硕士生来说，本书也是一本杰出的教科书。

李丹和刘涛等硕士生参与了本书的翻译过程，讨论过许多理解和翻译方面的问题，但是由于译者水平有限，译文中难免有不妥当、不确切乃至错误之处，敬请读者批评指正。

译者

2008 年 10 月

前　言

在设计顾问的业务中，我已经面对了无数的 FPGA 设计、方法和技术。无论客户是列于财富 100 强的名单，或只是一个初创的公司，他们将不可避免地做了一些正确的事情和做错了许多事情。在广泛的工业范围中已经做了多种的设计之后，我开始开发自己的库，并从这些经历的混杂知识进行直观的推断，当指导新的 FPGA 设计工程师时，我从这些经验中，草拟我的建议和意见。到目前为止，在讨论关于 FPGA 设计的专门实践时，这些经验很多已经作为专门的白皮书和应用要点的设计参考。这本书的目的是，把多年推广到更多公司和工程师团队的经验以及由专门的白皮书和应用要点汇集的许多知识，浓缩进一本书，可以用来完善工程师的知识和帮助他们成为高级的 FPGA 设计者。

有许多关于 FPGA 设计的书籍，但是其中很少真正详细地讲述高层次的实用课题。本书试图删去不太必要的理论，推测未来的技术及过时工艺的细节，而用简明、扼要的格式写成一本不占用读者时间来讲述各种课题的书。书中的很多章节假设读者理解有一定的基础，为了简洁起见，不总是涉及背景信息和/或理论框架。本书覆盖在真实设计中已经遇到的深层次的课题，在一些方面，这本书取代有限的工业经历，进入到经验的指导，将有助于读者免去学习少数困难的问题。这是先进的、实用的方法，也是本书的特色。

需要说明的是，本书不像小说那样从头到尾有连贯的情节，对于一组相互没有内在联系的高级课题，要写成连贯的情节是不可能的。为了组织这本书，我已经按照典型的设计流程来安排章节的次序。先讨论结构，然后是仿真，再是综合，接着是布图等。在书的开头提供的内容流图做了说明。为了便于进一步参考，章节排列在内容流图中相应方框的边上。

本书有专门章节包含大量的例子。为了简洁起见，我选择了 Verilog 作为默认的硬件描述语言，Xilinx 公司作为默认的 FPGA 销售商，选择 Synplicity 作为综合和布图的工具，书中覆盖的课题可以方便地映射到 VHDL 语言、Altera 销售商、Mentor Graphics 的工具等等，为完整起见所包含的这些内容可能只起一些要点的作用。所以，对于即使使用其他技术的读者，本书仍然是有价值的。如果有任何反馈意见，请发邮件到 steve. kilts@ spectrumdsi. com。

致谢

在我的经历中，我曾经特别荣幸地与许多杰出的数字设计工程师一起工作。开始在 Medtronic，后来为 Honeywell Guidant Teradyne Telex Unisys AMD ADC 等公司，以及大量 FPGA 应用的小型或初创的公司做顾问。我的知识在很大程度上要归功于由主要的 FPGA 厂商发表的应用说明和白皮报告。这些资源包括无价的实际探索，而这些不包含在标准的工程课程中。

针对本书，我特别感谢 Xilinx 和 Synplicity 公司，他们提供了整本书所利用的 FPGA 设计工具，以及大量的关键评论者。著名的评论者包括 Synplicity 的 Peter Calabrese、Sunburst Design 的 Cliff Cummine、Synplicity 的 Pete Danile、Axcon 的 Anders Enggaard、NuHorizons 的 Paul Fuchs、Xilinx 的 Don Hodapp、Synplicity 的 Ashok Kulkarni、Spectrum Design Solutions 的 Rod Landers、Logic 的 Ryan Link、Verein 的 Dave Matthews、Roman-Jones 的 Lance Roman、Polybus 的 B. Joshua Rosen、iSine 的 Gary Stevens、Xilinx 的 Jim Torgerson 和 Larry Weegman。

目　录

第 1 章　高速度结构设计

在采用任意编码方式时，高级工具的优化程度常常不足以满足大多数设计约束的要求。本章讨论数字设计中三个主要物理特性的第一个：速度。本章也讨论在 FPGA 中结构优化的方法。

根据问题的内容不同，速度有三种基本定义：流量（Throughput）、时滞（Latency）和时序（Timing）。在 FPGA 处理数据的内容中，流量定义为每个时钟周期处理的数据量。流量的通常度量是每秒的位数。时滞定义为数据输入与处理的数据输出之间的时间。时滞的一般度量是时间或时钟周期。时序定义为时序元件之间的逻辑延时，当一个设计没有“满足时序”时，意味着关键路径的延时，即触发器之间的最大延时比预定的时钟周期大，这些延时由组合逻辑延时、时钟到输出延时、布线延时、建立时间、时钟偏移等组成。时序的标准度量是时钟周期和频率。

在本章的课程中，将详细讨论以下的内容：

- 高流量结构使设计每秒可以处理的位数最大化。
- 低时滞结构使一个模块输入端到输出端的延时最小化。
- 时序优化可减少关键路径的组合延时。
  - 添加寄存器层次分割组合逻辑结构；
  - 并行结构使分开的时序执行操作成为并行操作；
  - 把逻辑结构规定展平成特权编码信号；
  - 寄存器平衡使围绕流水线的寄存器重新分配组合逻辑；
  - 重新安排路径把关键路径的操作转到非关键的路径。

1.1　高流量

高流量设计是与稳定状态数据率有关的设计，但很少涉及任何规定的数据段通过设计要求的传播时间（时滞）。高流量设计的概念与福特提出制造大量汽车的装配线的概念相同。在处理数据的数字设计中，定义这个概念为较抽象的术语：流水线（pipeline）。

流水线设计概念上十分类似于装配线运行，其中原材料或输入数据进入前端，然后通过制造和处理的各个级，最后作为完成的产品或输出数据退出。流水线设计的优越性是新数据在前面的数据完成之前就可以进行处理，就像汽车在装配线上加工一样。流水线几乎应用于所有高性能的器件中，且对各种规定的结构是无限制的，例如包括 CPU 指令集、网络协议堆栈、密码引擎等。

从算法的观点看，在流水线设计中一个重要的概念是“拆开环路”。作为一个例子，考虑以下一段代码，它类似于在软件实现求 X 的三次幂中使用的代码，即在微处理器中一组顺序指令执行的指令。

```
XPower = 1;
for (i=0;i < 3; i++)
  XPower = X * XPower;
```

注意上面的代码是一个递归的算法。相同的变量和地址被存取直到计算完成。因为微处理器一次只执行一个指令，所以没有利用并行性。（为讨论的目的，只考虑单核的处理器。）一个类似的实现可以在硬件中产生。考虑下面相同算法的 Verilog 语言实现（没有考虑输出范围）。

```
module power3(
  output [7:0] XPower,
  output       finished,
  input  [7:0] X,
  input        clk, start); // the duration of start is a
                               single clock
  reg    [7:0] ncount;
  reg    [7:0] XPower;
  assign finished = (ncount == 0);
  always@(posedge clk)
    if(start) begin
      XPower <= X;
      ncount <= 2;
    end
    else if(!finished) begin
      ncount <= ncount - 1;
      XPower <= XPower * X;
    end
endmodule
```

在上面的例子中，相同的寄存器和计算资源被重新利用，直到计算完成，如图 1-1 所示。由于这类迭代实现，新的计算直到前面的计算已经完成才可以开始。这个迭代方案十分类似于软件实现，也应注意，要求某些握手信号来表示开始和完成一次计算。外部模块也必须利用这个握手信号来传递新数据到模块并接收一个完成的计算。这个实现的性能是

流量 = 8/3，或 2.7 位/时钟

时滞 = 3 时钟

时序 = 关键路径中一个乘法器延时

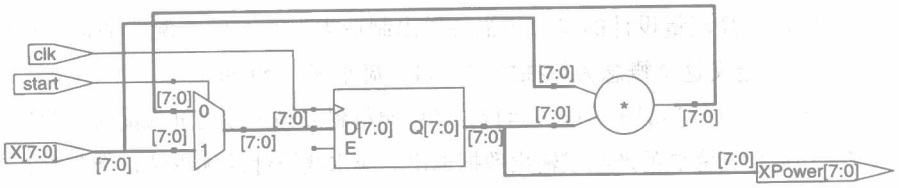


图 1-1 迭代实现

与相同算法的流水线版本对比：

```
module power3(
  output reg [7:0] XPower,
  input            clk,
  input      [7:0] X
  );
  reg        [7:0] XPower1, XPower2;
  reg        [7:0] X1, X2;
  always @(posedge clk) begin
```

```
    // Pipeline stage 1
    X1      <= X;
    XPower1 <= X;
    // Pipeline stage 2
    X2      <= X1;
    XPower2 <= XPower1 * X1;
    // Pipeline stage 3
    XPower <= XPower2 * X2;
  end
endmodule
```

在上面的实现中，X 的数值传递到两个流水线级，每级独立的资源计算相应的乘法操作。注意，当 X 在第二级流水线用来计算最后的三次幂的同时，X 的下一个数值可以送到第一级流水线，如图 1-2 所示。

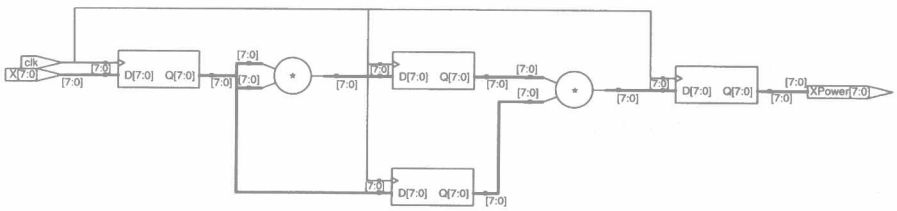


图 1-2 流水线实现

X^3 的最后计算（Xpower3 资源）和 X 下一数值的第一次计算（XPower2 资源）二者同时进行，这个设计的性能是：

流量 = 8/1，或 8 位/时钟

时滞 = 3 时钟

时序 = 关键路径中一个乘法器延时

流量的性能增加超过了迭代实现的 3 倍。通常，如果要求 n 次迭代环路的算法拆开后，流水线实现将呈现 n 倍的流量性能增加。因为流水线实现仍要求 3 个时钟来传播最后的计算结果，就时滞而论性能没有损失。类似地，因为关键路径仍然只包含一个乘法器，所以时序并没有恶化。

拆开一个迭代环路会增加流量。

为如此拆开环路所付出的代价是增加面积。迭代实现要求单个寄存器和乘法器（一些控制信号没有表示在图中），但是流水线实现要求为 X 和 XPower 二者分开的寄存器，为每个流水线级分开的乘法器。面积优化在第 2 章讨论。

拆开一个迭代环路的代价是成比例地增加面积。

1.2 低时滞

低时滞设计是通过最小化中间处理的延时来尽可能快速地把数据从输入端传递到输出端的设计。通常，低时滞设计将要求并行性、去除流水线、缩短逻辑，可能减少设计中的流量或降低最大时钟速度。

返回到三次幂的例子，没有对迭代实现进行明显的时滞优化，因为每个接连的乘法操作必须为下一步操作进行寄存。但是，流水线实现有明显的路径来减少时滞。注意每个流水线级，每个乘法的积必须等待下一个时钟沿到来才传递到下一级。去除流水线寄存器可以使输入到输出时序最小化。

```
module power3(
  output [7:0] XPower,
  input  [7:0] X
  );
  reg    [7:0] XPower1, XPower2;
  reg    [7:0] X1, X2;

  assign XPower = XPower2 * X2;

  always @* begin
    X1      = X;
    XPower1 = X;
  end

  always @* begin
    X2      = X1;
    XPower2 = XPower1*X1;
  end
endmodule
```

在上面的例子中，寄存器从流水线中分离出去，每一级是前级的组合表达式，如图1-3所示。

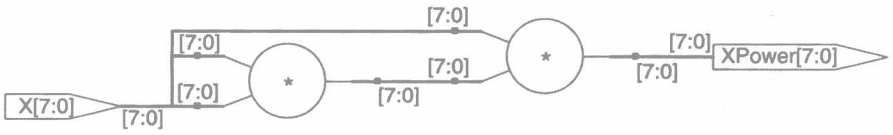


图1-3 低时滞实现

这时的设计性能为：

流量 = 8位/时钟（假设每个时钟一个新的输入）

时滞 = 在1和2个乘法器延时之间，0时钟

时序 = 关键路径中2个乘法器延时

由于移去流水线寄存器，这个设计的时滞已经减少到低于单个时钟周期。

可以通过移去流水线寄存器来减少时滞。

在时序上的损失是明显的，前面的实现理论上可以运行到系统时钟周期接近单个乘法器的延时，但是在低时滞实现中，时钟周期必须至少是2个乘法器的延时（取决于其实现）加上在关键路径中的任何外部逻辑。

移去流水线寄存器的损失是增加寄存器之间的组合延时。

1.3 时序

时序指的是一个设计的时钟速度。在设计中任何两个时序元件之间的最大延时将决定最大

的时钟速度。时钟速度的概念比这一章中另一处讨论的速度/面积权衡有更低层次的抽象，因为时钟速度一般不直接与这些拓扑有关，虽然在这些结构中的权衡将确实对时序有影响。例如，不清楚设计的细节，设计者不可能知道流水线拓扑是否会比迭代运行更快。最高速度或最大频率可以直接按照著名的最大频率方程定义（不管时钟到时钟的抖动）：

方程式 1-1　最大频率

$$F_{max} = \frac{1}{T_{clk-q} + T_{logic} + T_{routing} + T_{setup} - T_{skew}}$$

其中 F_{max} 是时钟可允许的最大频率，T_{clk-q} 是从时钟到达直至数据到达 Q 端的时间，T_{logic} 是逻辑通过触发器之间的传播延时，$T_{routing}$ 是触发器之间的布线延时，T_{setup} 是下一个时钟上升沿之前数据必须到达 D 端的最小时间（建立时间），T_{skew} 是启动触发器和捕捉触发器之间时钟的传播延时。

下一节讨论改善时序性能的各种方法和要求的权衡。

1.3.1　添加寄存器层次

第一个结构的时序改进策略是添加中间的寄存器层次到关键路径，这个技术应该利用在高度流水线的设计，附加的时钟周期的时滞不违反设计的技术条件，整个功能将不受进一步增加的寄存器的影响。

例如，假设以下有限脉冲响应（FIR）实现的结构不满足时序要求：

```
module fir(
  output [7:0] Y,
  input  [7:0] A, B, C, X,
  input        clk,
  input        validsample);
  reg    [7:0] X1, X2, Y;

  always @(posedge clk)
    if(validsample) begin
      X1 <= X;
      X2 <= X1;
      Y <= A* X+B* X1+C* X2;
    end
endmodule
```

结构上，所有乘/加操作发生在一个时钟周期，如图 1-4 所示。

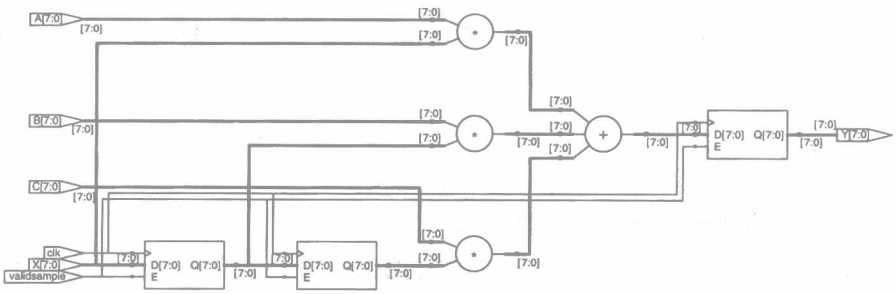


图 1-4　带长路径的 MAC

换言之，一个乘法器和一个加法器组成的关键路径比最小时钟周期的要求大，假设时滞要

求不固定在一个时钟周期，添加额外的中间寄存器到此乘法器，使设计进一步流水线。第一层次是容易的：只是在乘法器和加法器之间添加一个流水线层次。

```
module fir(
  output [7:0] Y,
  input  [7:0] A, B, C, X,
  input        clk,
  input        validsample);
  reg    [7:0] X1, X2, Y;
  reg    [7:0] prod1, prod2, prod3;

  always @ (posedge clk) begin
    if(validsample) begin
      X1    <= X;
      X2    <= X1;
      prod1 <= A * X;
      prod2 <= B * X1;
      prod3 <= C * X2;
    end
    Y <= prod1 + prod2 + prod3;
  end
endmodule
```

在上面的例子，加法器用一个流水线级与乘法器分开，如图 1-5 所示。

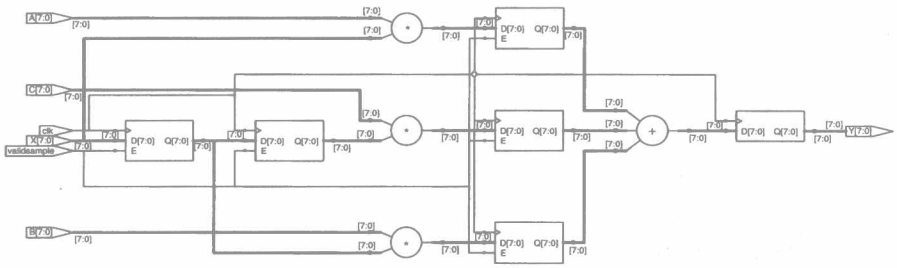


图 1-5 添加的流水线寄存器

对于流水线乘法器是好的选择，因为计算可以很容易分解成级。把乘法器和加法器分解成可以单独寄存的级，使附加流水线成为可能。

把关键路径分成两个更小延时的路径，添加寄存器层次改进时序。

这些功能的各种实现涵盖在其他章节，但是结构一旦分解成级，附加的流水线如上面的例子一样容易理解。

1.3.2 并行结构

第二个结构的时序改进策略是重新组织关键路径，以致并行地实现逻辑结构。当通过一列串联的逻辑估值的一个函数可以分解和并行地估值时，就应该利用这个技术。例如，假设前一节讨论的标准流水线 3 次幂设计不满足时序的要求。为了产生并行结构，可以将乘法器分解成独立的操作，然后重新组合他们。例如，一个 8 位的二进制乘法器可以用字段 A 和 B 表示：

$$X = \{A, B\}$$

其中 A 是最高有效位字段，B 是最低有效位字段。

因为在 3 次幂的例子中被乘数等于乘数，乘法操作可以重新组织如下：

$$X * X = \{A, B\} * \{A, B\} = \{(A * A), (2 * A * B), (B * B)\}$$

这样把问题简化为一个串行的 4 位乘法器，然后重新组合乘积，可以用以下的模块实现：

```
module power3(
  output [7:0] XPower,
  input  [7:0] X,
  input        clk);
  reg    [7:0] XPower1;
  // partial product registers
  reg    [3:0] XPower2_ppAA, XPower2_ppAB, XPower2_ppBB;
  reg    [3:0] XPower3_ppAA, XPower3_ppAB, XPower3_ppBB;
  reg    [7:0] X1, X2;
  wire   [7:0] XPower2;

  // nibbles for partial products (A is MS nibble, B is LS
                                   nibble)
  wire   [3:0] XPower1_A = XPower1[7:4];
  wire   [3:0] XPower1_B = XPower1[3:0];
  wire   [3:0] X1_A      = X1[7:4];
  wire   [3:0] X1_B      = X1[3:0];
  wire   [3:0] XPower2_A = XPower2[7:4];
  wire   [3:0] XPower2_B = XPower2[3:0];
  wire   [3:0] X2_A      = X2[7:4];
  wire   [3:0] X2_B      = X2[3:0];

  // assemble partial products
  assign XPower2         = (XPower2_ppAA << 8)+
                           (2*XPower2_ppAB << 4)+
                            XPower2_ppBB;
  assign XPower          = (XPower3_ppAA << 8)+
                           (2*XPower3_ppAB << 4)+
                           XPower3_ppBB;

  always @(posedge clk) begin

    // Pipeline stage 1
    X1           <= X;
    XPower1      <= X;

    // Pipeline stage 2
    X2           <= X1;
    // create partial products
    XPower2_ppAA <= XPower1_A * X1_A;
    XPower2_ppAB <= XPower1_A * X1_B;
    XPower2_ppBB <= XPower1_B * X1_B;

    // Pipeline stage 3
    // create partial products
    XPower3_ppAA <= XPower2_A * X2_A;
    XPower3_ppAB <= XPower2_A * X2_B;
    XPower3_ppBB <= XPower2_B * X2_B;
  end
endmodule
```

这个设计没有考虑任何溢出的问题，只是用来说明要点，乘法器可以拆成能够独立操作的更小的功能，如图 1-6 所示。

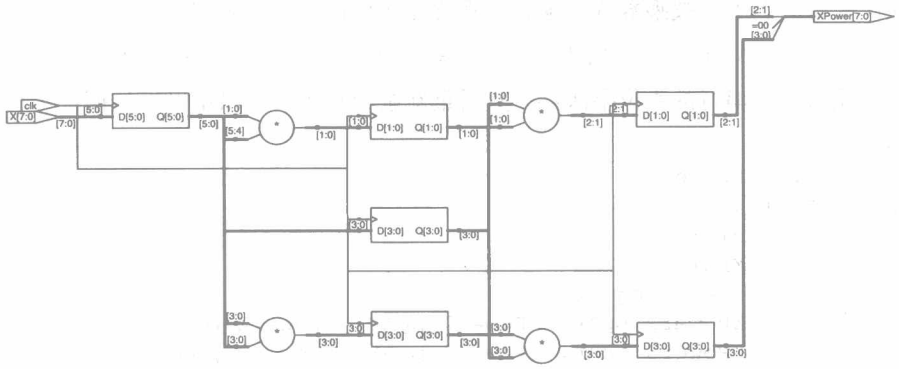


图 1-6 分级的乘法器

通过把乘法器拆成可以并行执行的更小的操作，最大的延时可以减小到通过任何子结构的最长延时。

把一个逻辑功能分成大量可以并行估值的更小的功能，减少路径延时为子结构的最长延时。

1.3.3 展平逻辑结构

第三个结构的时序改进策略是展平逻辑结构。这是与前一节定义的并行结构概念紧密相连的，但是专门应用于因为特权编码而链接的逻辑。一般情况下，综合和布局图工具有足够的智能来复制逻辑减少扇出，但是还没有足够的智能来拆开串行方式编码的逻辑结构，或者没有与设计的特权要求有关的足够的信息。例如，考虑下面来自地址译码器用于写入 4 个寄存器的控制信号：

```
module regwrite(
  output reg [3:0] rout,
  input            clk, in,
  input      [3:0] ctrl);

  always @(posedge clk)
    if(ctrl[0])       rout[0] <= in;
    else if(ctrl[1]) rout[1] <= in;
    else if(ctrl[2]) rout[2] <= in;
    else if(ctrl[3]) rout[3] <= in;
endmodule
```

在上面的例子中，每个控制信号用与其他控制信号有特权关系来编码，这类特权编码的实现如图 1-7 所示。

如果控制线是来自另一个模块地址译码器的选通，则每个选通对其他的选通是相互排斥的，因为它们都代表唯一的地址。但是，这里已经仿照特权判决那样编码。由于控制信号的特性，以上的代码将准确地仿照并行方式编码那样操作，但是，综合工具未必会有足够的智能来识别它，特别当地址译码器发生在另一个寄存器层次后面。

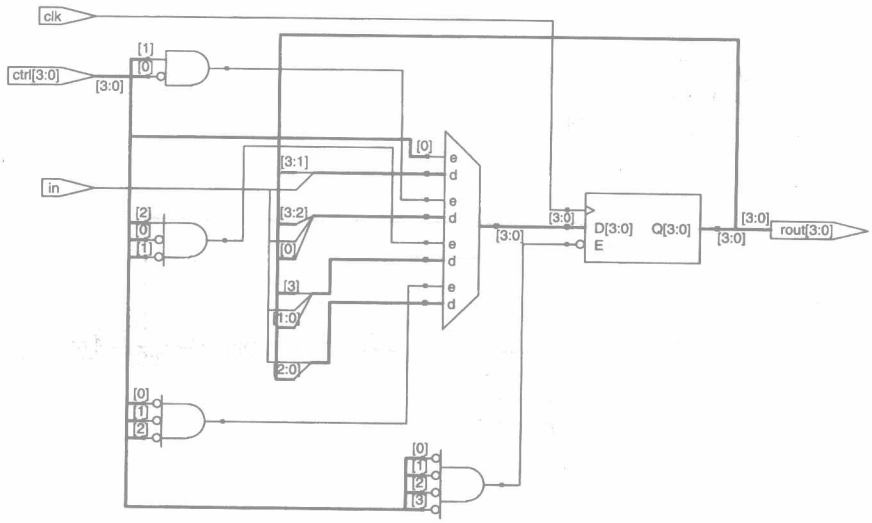


图 1-7 特权编码

为了去除此特权，展平此逻辑，可以按照如下方式给这个模块编码：

```
module regwrite(
  output reg [3:0] rout,
  input            clk, in,
  input      [3:0] ctrl);

  always @(posedge clk) begin
    if(ctrl[0]) rout[0] <= in;
    if(ctrl[1]) rout[1] <= in;
    if(ctrl[2]) rout[2] <= in;
    if(ctrl[3]) rout[3] <= in;
  end
endmodule
```

如在门级实现中可以看到的，利用了无特权逻辑，如图 1-8 所示。每个控制信号独立作用，独立地控制它的相应的 rout 位。

去除不需要的特权编码，展平逻辑结构，减少路径延时。

1.3.4 寄存器平衡

第 4 个策略称为寄存器平衡。概念上讲，这个方法是平等地重新分布寄存器之间的逻辑，减少任何两个寄存器之间最坏条件的延时。这个技术应该随时利用在关键路径和相邻路径之间逻辑高度不平衡时。因为时钟速度只由最坏条件路径来决定，可以做小的改变而成功地重新平衡关键逻辑。

许多综合工具也有称为寄存器平衡的优化功能，这个特性实质上将重新确定专门的结构，改变按预定方式围绕逻辑的寄存器。对譬如大乘法器等通常的结构这是有用的，但是它是受限制的，将不改变逻辑也不识别定制功能。取决于工艺，可能要求更昂贵的综合工具来实现。因此，理解这个概念和有能力按定制逻辑结构重新分配逻辑十分重要。