FPGA嵌入式微处理器系统设计

[德] 乌韦·迈耶-贝斯(Uwe Meyer-Baese) 著
杨文波 陈 进 译

清華大學出版社
北 京

北京市版权局著作权合同登记号　图字：01-2022-4814

First published in English under the title
Embedded Microprocessor System Design using FPGAs
by Uwe Meyer-Baese.
Copyright © Springer Nature Singapore Pte Ltd. 2021.
This edition has been translated and published under licence from Springer Nature Switzerland AG. Part of Springer Nature.

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。举报：010-62782989，beiqinquan@tup.tsinghua.edu.cn。

图书在版编目（CIP）数据

FPGA 嵌入式微处理器系统设计 / (德) 乌韦·迈耶-贝斯著 ; 杨文波, 陈进译.
北京 : 清华大学出版社, 2025. 6. -- ISBN 978-7-302-69435-9

I. TP331.2

中国国家版本馆 CIP 数据核字第 202555AJ19 号

责任编辑：王　军
封面设计：高娟妮
版式设计：恒复文化
责任校对：成风进
责任印制：杨　艳

出版发行：清华大学出版社

网　　址：https://www.tup.com.cn，https://www.wqxuetang.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：大厂回族自治县彩虹印刷有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张**：27　　**字　　数**：689 千字
版　　次：2025 年 7 月第 1 版　　**印　　次**：2025 年 7 月第 1 次印刷
定　　价：128.00 元

产品编号：097319-01

前　言

嵌入式微处理器系统无处不在，只要看看你的周围就能明白这一点。你会在手机、数字时钟、GPS、摄像机和互联网路由器以及家用电子娱乐设备中找到它们。一辆现代汽车通常会使用50~100 个微处理器。嵌入式系统通常会在价格、功耗、内存或存储等资源上受限。一台通用计算机通常要耗费数百瓦的功率，而一个时钟或遥控器只需要耗费微瓦级功率，从而能够靠一节AAA 电池运行一年。尽管许多嵌入式系统运行时需要低功耗，但其所实施的算法，像 UMTS 手机中使用的 turbo 纠错码，对计算量的要求却很高。不管怎样，今天，嵌入式处理器能够执行复杂的任务并运行相关的复杂算法。一辆汽车中的微处理器估计会使用上亿行代码，其中仅 GPS 和无线电模块就占了 2000 万行。

FPGA 是开始探索嵌入式系统设计空间的最佳选择，因为它们属于细粒度逻辑可编程的商业现货(Commercial Off The Shelf，COTS)器件，其一次性工程(Non Recurring Engineering，NRE)成本比目前能买到的基于单元(cell)的系统低得多。最新一代的 FPGA 电路板和设备支持在同一块板卡上使用软核、参数化或硬核微处理器来设计微处理器系统。这些板卡为许多设计提供了很棒的起点，因为板卡上还有大量的外围元件，如音频编解码器、视频 HDMI 连接器或 SD 卡。如果你使用的是非 FPGA 的标准商业现货微处理器系统，要把这些组件纳入项目中就会非常耗时。

在 21 世纪进入第二个十年之际，我们发现，可编程逻辑器件(Programable Logic Device，PLD)市场的两个领导者(Altera/Intel 和 Xilinx)的收入都超过了 20 亿美元。在过去的十年中，FPGA 获得了超过 20%的稳定增长，比 ASIC 和 PDSP 高出 10%。这是因为 FPGA 与 ASIC 有许多共同的特点，如减少了尺寸、重量和功耗；更高的吞吐量；增强了防止未经授权复制的安全性；更低的器件和库存成本，更少的电路板测试成本；在有些方面，FPGA 更胜过了 ASIC，如减少了开发时间(快速原型)、具有在线(重新)编程能力和更低的一次性工程成本，从而让少于 1000 个成品的解决方案的设计更加经济。硬件设计领域的另一个趋势是从图形设计输入(entry)向硬件描述语言(Hardware Description Language，HDL)进行迁移。人们发现，基于 HDL 的设计输入中的“代码重用”率要比基于图形的高得多。对 HDL 设计工程师的需求量很大，我们已经见到本科生课程在用 HDL 教授逻辑设计。如今，有两种流行的 HDL 语言。美国西海岸和亚洲更喜欢使用 Verilog，而美国东海岸和欧洲更经常使用 VHDL。尽管由于 VHDL-2008 支持定点和浮点数据类型，一些使用 VHDL 进行设计的例子更容易阅读一些，但对于使用 FPGA 进行的嵌入式微处理器设计，这两种语言似乎都很适合。其他限制因素可能包括个人偏好、EDA 库和工具的可用性、可读性、能力和使用编程语言接口(Programming Language Interface，PLI)的语言扩展以及商业、

业务和营销问题等，而以上因素也仅是略举几例。如今，工具供应商已经认识到这两种语言都必须得到支持，而本书也涵盖了这两种设计语言的例子。我们现在很幸运，“基准”FPGA 工具可以从不同的来源获得，并且对于教学用途来说基本上没有成本。我们在本书中利用了这个条件。书中包含了 Altera/Intel Quartus 15.1 Lite Edition 以及 Xilinx Vivado 2016.4 工具可用的代码，后者提供了一套完整的设计工具，涵盖从能感知内容的编辑器、微处理器配置器、编译器、模拟器到比特流生成器。本书展示的所有例子都用 VHDL 和 Verilog 编写，也容易移植到其他专有的设计输入系统中。

本书的结构安排如下。第 1 章首先简要介绍了当今主流的微处理器和基本的微处理器原理，特别是基于 FPGA 的微处理器。它还包括一个关于用 IP 块进行设计的概述和一个 PLL IP 核设计实例。第 2 章讨论了用于设计当前最先进的 FPGA 系统的器件、板卡和工具。还讨论了终极 RISC(URISC)微处理器的详细案例，包括模型讨论、编译步骤、仿真、性能评估、功率估计以及使用 Quartus 和 Vivado 进行的布局规划。这个案例研究是后续章节中许多其他设计实例的基础。第 3 章和第 4 章涉及微处理器设计中使用的 VHDL 和 Verilog 语言元素。第 5 章回顾了 ANSI C 语言，还讨论了调试方法以及与 C++的区别。第 6 章介绍了微处理器的软件工具开发，详述了使用 GNU Flex 的词法分析和使用 GNU Bison 的解析器实现。我们为 PICOBLAZE 微处理器设计了一个汇编器，为三地址机器设计了一个基本的和全功能的 C 语言编译器，还讨论了指令集模拟器和软件调试器。在第 7 章中，逐步开发了软核 PICOBLAZE，使其增加越来越多的架构特性。我们研究了循环控制和数据存储器的设计并用 HDL 来实现。第 8 章全面讨论了最流行的基于 FPGA 的 8 位微处理器对应的完整指令集。第 9 章和第 10 章讨论了 Altera/Intel 和 Xilinx 设备具有的两个最流行的参数化内核，分别称为 Nios II 和 MICROBLAZE。我们开发了一种自顶向下和自底向上的系统设计方法。我们通过为 Nios 添加浮点数协处理器和为 MICROBLAZE 添加 HDMI 解码器来演示如何将定制 IP 添加到微处理器中。我们也构建了 Tiny RISC 版本的处理器，名为 TRISC3N 和 TRISC3MB，它们支持精简指令集并且可以运行通过供应商 GCC 工具生成的基本程序。第 11 章讨论了最流行的 32 位商业现货硬 IP 处理器核 ARM Cortex-A9，它已经包含在 Altera/Intel 和 Xilinx 的最新设备中。我们再次展示了自顶向下和自底向上的设计，同时会开发一个 Tiny RISC 版本的 TRSC3A，从而展示 ARM Cortex-A9 为什么在架构上更有优势。我们定制了一个 IP 以加速 FFT 地址计算，并测量其速度提升水平。本书的附录 A 采用在线方式提供，其中包含了可以在 Quartus 和 Vivado 中使用的全部 5 个(小)处理器模型(URISC、TRISC2、TRISC3N、TRISC3MB 及 TRISC3A)对应的 Verilog 源代码和 `xsim` 仿真。附录 B 也采用在线方式提供，其中的缩写词列表可供读者快速参考。还有一些额外的文件、HDL 语言参考卡片以及实用工具将在 `GitHub`[1]和作者的个人网站[2]上发布，读者也可通过扫描本书封底的二维码下载。

在此要说明的是，本书采用黑白印刷，书中的彩图以在线方式提供，读者可通过扫描本书封底的二维码下载。

1 译者注：https://github.com/uwemeyerbaese。
2 译者注：https://web1.eng.famu.fsu.edu/~umb/。

致　　谢

本书的编写基于我自2003年以来在位于塔拉哈西市的佛罗里达农工大学——佛罗里达州立大学工程学院开设的嵌入式微处理器系统设计课程。在过去的17年里，我指导了60多位硕士和博士的论文项目。我希望感谢所有在实验室和会议上帮助我进行过关键性讨论的同事们以及用他们的设计想法启发过我的学生们，其中的一些设计想法会在第9、10和11章中讨论。

我特别感谢我在格拉纳达大学的朋友和同事，我从1998年起就有幸与他们一起工作，并在2014年夏天到西班牙拜访他们。他们最近帮助我的实验室开始了对FECG(胎儿心电图)分析的研究。特别感谢A. Garcia、E. Castillo、L. Parrilla、A. Lloris和J. Ramírez。

我想感谢2015年夏季我在德国卡塞尔大学进行研究访问时的同事们，感谢他们在Vivado Xilinx设计和ZyBo板方面对我提供的大力支持。特别感谢P. Zipf、M. Kumm和K. Möller。

特别感谢Francesco Poderico对我的信任，让我继续在他的PCCOMP编译器上提供对PICOBLAZE的支持。

我要感谢亚琛高等理工学院国际空间站的同事们，在2006年和2008年暑期我受洪堡奖资助赴德国研究期间，他们花了很多时间和精力教我进行LISA和C编译器设计。特别感谢H. Meyr、G. Ascheid、R. Leupers、D. Kammler和M. Witte。

我尤其要感谢来自马德里大学的Guillermo Botella和Diego Gonzalez，感谢他们在图像和视频处理应用中提供的帮助。

我想感谢来自Altera/Intel和Xilinx的Rebecca Nevin、B. Esposito、M. Phipps、A. Vera、M. Pattichis在软硬件方面提供的支持。我要感谢出版方(Spring-Verlag)的Charles Glaser和Merkle博士近年来给予我的持续支持和帮助。

如果读者发现任何错误或者有任何改进建议，请通过出版方联系我。

Uwe Meyer-Baese

美国佛罗里达州塔拉哈西市

2020年4月

目　录

第 1 章　嵌入式微处理器系统基础 …… 1
1.1　引言 …… 1
1.2　FPGA 上的嵌入式微处理器 …… 6
1.3　微处理器指令集设计 …… 7
1.4　FPGA 技术 …… 20
1.5　使用知识产权核的设计 …… 27
1.6　复习题和练习 …… 33

第 2 章　FPGA 器件、板卡和设计工具 …… 37
2.1　引言 …… 37
2.2　原型板卡的选择 …… 40
2.2.1　存储器 …… 42
2.2.2　基本 I/O 组件 …… 42
2.2.3　显示选项 …… 43
2.2.4　模拟接口 …… 45
2.2.5　通信 …… 48
2.3　FPGA 结构 …… 52
2.3.1　Xilinx FPGA 架构概述 …… 52
2.3.2　Altera FPGA 架构概述 …… 55
2.4　案例研究：名为 URISC 的 PSM …… 60
2.4.1　URISC 处理器模型 …… 60
2.4.2　Altera QUARTUS 设计编译 …… 65
2.4.3　Xilinx VIVADO 设计编译 …… 67
2.4.4　用于仿真的设计工具考虑 …… 68
2.4.5　QIVADO 布图规划 …… 74
2.4.6　VIVADO 布图规划 …… 75
2.4.7　时序估计和性能分析 …… 76
2.5　复习题和练习 …… 78

第 3 章　用 VHDL 设计微处理器组件 …… 81
3.1　引言 …… 81
3.2　词法元素 …… 82
3.3　运算符与赋值 …… 86
3.4　数据类型、数据对象和属性 …… 87
3.4.1　VHDL 数据类型 …… 87
3.4.2　转换函数 …… 89
3.4.3　属性 …… 89
3.4.4　数据对象 …… 90
3.5　VHDL 语句和设计编码建议 …… 90
3.5.1　组合逻辑编码建议 …… 92
3.5.2　基本时序电路编码：触发器和锁存器 …… 94
3.5.3　存储器 …… 95
3.5.4　有限状态机 …… 95
3.5.5　设计层次结构和组件 …… 97
3.5.6　VHDL 编码风格、资源和常见错误 …… 97
3.6　延伸阅读 …… 98
3.7　复习题和练习 …… 99

第 4 章　用 Verilog 设计微处理器组件 …… 111
4.1　引言 …… 111
4.2　词法元素 …… 112
4.3　运算符与赋值 …… 115
4.4　数据类型和值集 …… 117
4.4.1　Verilog 值集 …… 117
4.4.2　Verilog 数据类型 …… 117
4.5　Verilog 语句和设计编码建议 …… 119

4.5.1 组合逻辑编码建议……121
4.5.2 基本顺序电路编码：触发器和锁存器……123
4.5.3 存储器……123
4.5.4 有限状态机……125
4.5.5 设计层次和组件……126
4.5.6 Verilog 编码风格、资源和常见错误……126
4.6 延伸阅读……128
4.7 复习题和练习……129

第 5 章 用 C/C++进行微处理器编程……139
5.1 引言……139
5.2 词法元素……140
5.3 数据类型、数据属性和储存类…145
5.4 C 运算符和赋值……147
5.5 控制流构件……150
5.6 代码层级和 I/O……152
5.7 额外的考虑和推荐……157
5.8 延伸阅读……166
5.9 复习题和练习……167

第 6 章 嵌入式微处理器系统中的软件工具……175
6.1 引言……175
6.2 汇编器开发和词法分析……176
6.3 解析器开发……186
6.4 软件调试器和指令集仿真器……197
6.5 复习题和练习……198

第 7 章 PicoBlaze 软核微处理器的设计……203
7.1 引言……203
7.2 KCPSM6 指令集概况……206
7.3 初始 PicoBlaze 可综合架构……209
7.4 带有便签存储器的 PicoBlaze 可综合设计……214
7.5 带有链接控制的 PicoBlaze 可综合架构……216
7.6 复习题和练习……225

第 8 章 PicoBlaze 软核微处理器中的软件工具……229
8.1 引言……229
8.2 KCPSM6 汇编器……230
8.3 PicoBlaze 指令集仿真器……235
8.4 支持 PicoBlaze 的 C 编译器……236
8.5 复习题和练习……244

第 9 章 Altera Nios 嵌入式微处理器……247
9.1 引言……247
9.2 自顶向下的 Nios II 系统设计……250
9.3 自底向上的 Nios II 系统设计……254
9.4 定制指令的 Nios II 系统设计……262
9.5 深入了解：Nios II 指令集架构…282
9.6 复习题和练习……299

第 10 章 Xilinx MicroBlaze 嵌入式微处理器……307
10.1 引言……307
10.2 自顶向下的 MicroBlaze 系统设计……308
10.3 自底向上的 MicroBlaze 系统设计……310
10.4 定制指令的 MicroBlaze 系统设计……316
10.4.1 DVI 和 HDMI 显示选项实现……316
10.4.2 TMDS 编码和解码……317
10.4.3 HDMI 编码器……322

10.4.4　文本终端和字体设计……323
10.4.5　HDL 实现的 HDMI 编码器……327
10.4.6　HDMI 编码器的 CIP 接口……333
10.4.7　HDMI 编码器的综合结果……338
10.5　深入了解：MicroBlaze 指令集架构……338
10.5.1　HDL 实现和测试……350
10.5.2　综合结果和 ISA 经验教训……359
10.6　复习题和练习……360

第 11 章　ARM Cortex-A9 嵌入式微处理器……369
11.1　引言……369
11.2　自顶向下的 ARM 系统设计……371
11.3　自底向上的 ARM 系统设计……376
11.4　定制指令的 ARM 系统设计……381
11.5　深入了解：ARMv7 指令集架构……387
11.6　复习题和练习……410

——以下内容在线提供——

附录 A　Verilog 源代码和 Xilinx Vivado 仿真

附录 B　缩写词列表

参考文献

第 1 章

嵌入式微处理器系统基础

摘要

本章概述了本书中我们将使用的算法和技术。我们首先从嵌入式微处理器系统设计综述开始讨论，然后将重点讨论基于 FPGA 设计的系统。

关键词

嵌入式微处理器 • 指令集 • 现场可编程门阵列 • FPGA 技术 • 知识产权核(IP Core) • FPGA 测试基准 • 存储器架构 • 中央处理器(Central Processing Unit，CPU)• 常量编码可编程状态机 KCPSM • 寻址模式 • 数据流架构 • 软核 • 参数化核 • 硬核 • 锁相环(Phase-Locked Loop，PLL)• 硬件语言仿真器 ModelSim • 复杂指令集计算机(Complex Instruction Set Computer，CISC) • 精简指令集计算机(Reduced Instruction Set Computer，RISC) • 专用芯片 ASIC • 标准单元芯片 CBIC

1.1 引言

嵌入式系统通常具有的特征是，它们包括一个微处理器(μP)，但没有键盘、显示器或鼠标等计算机的典型组件[VG02, HHF08, Wol08]。

如今，大多数微处理器被用于嵌入式系统。只要看看你的周围，就会发现嵌入式系统无处不在。你可以在手机、数字时钟、GPS、录像机、互联网路由器，以及家用终端电子娱乐设备中找到它们[A08]。一辆现代汽车通常使用 50~100 个微处理器[C09]。一些嵌入式系统可能也有实时要求，例如，一定的计算必须在一定的期限内完成，就像汽车的 ABS 系统一样，但大多数不是这样。嵌入式系统通常受到价格、功耗、内存或存储资源的限制。一台通用计算机通常具有数百瓦功耗，而一台时钟或遥控器仅消耗微瓦级(μW)的功耗，以至于可以使用一节 AAA 电池运行一整年。虽然许多嵌入式系统要求具有低功耗，但其实现的算法，如 UMTS 手机中使用的 turbo 纠错码，对计算量却要求很高。总而言之，如今的嵌入式处理器正在执行复杂的任务并运行复杂的算法。汽车上的微处理器大约需要编写 1 亿行代码才能实现，仅 GPS 和无线电模块就占用了 2 千万行代码[C09]。典型的嵌入式微处理器系统的设计目标可以总结如下。

- 针对功耗、性能、内存、成本进行优化
- 兼容性

- JTAG 调试支持
- 架构风格：微控制器、可编程数字信号处理器(Programmable Digital Signal Processor，PDSP)、SIMD(单指令多数据流)、MIMD(多指令多数据流)
- 超标量：指令级并行、功能单元数目、协处理器
- 存储器层次结构
- 指令集(CISC、RISC、控制器)

考虑到嵌入式系统的广泛应用范围，毫不奇怪的是，没有一个微处理器能够满足所有这些硬件(Hardware，HW)和软件(Software，SW)的需求，所以系统定制是不可避免的。这正是现代嵌入式系统工程师的工作描述：对硬件(即微处理器及其外设)和软件(即算法和用诸如汇编或C/C++等计算机语言进行编码)有深入的了解。要掌握如此复杂的系统设计需要具有不同的技能，涵盖从数字逻辑、微处理器、计算机架构、编译器设计、编程语言(VHDL/Verilog/C/C++)、数字信号处理(Diyital Signal Processing，DSP)、图像处理、数学算法，到要解决的任务的背景知识，如图 1.1 所示。

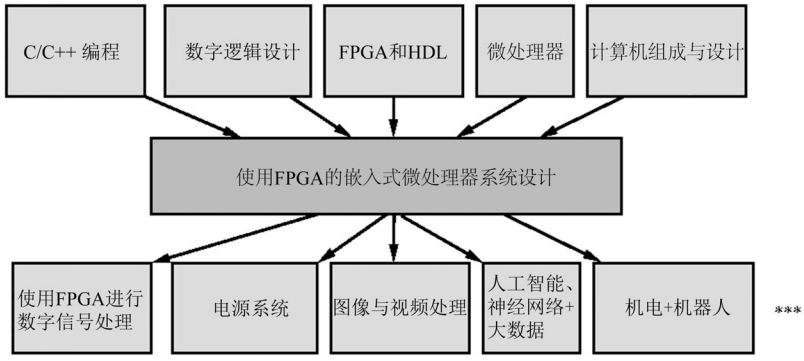


图 1.1　嵌入式微处理器的必要预备知识和可能应用领域的概述

微处理器历史上的几个里程碑都促成了今天嵌入式系统的成功。从 20 世纪 60 年代集成电路(Integrated Circuit，IC)的发明，到早期出现的英特尔微处理器，以及后来一再被证实的摩尔定律(即每 18 个月晶体管数量翻一番)，这些事件使我们今天能够以低成本制造数百万个门电路。20 世纪 80 年代 RISC 处理器的发明使得可以构建功耗只有通用处理器(General Purpose Processor，GPP)零头的高性能微处理器(μP)。最后，同样重要的是，如今 FPGA 允许我们采用商业现货(Commercial Off The Shelf，COTS)器件来设计各种定制的微处理器，而在这过程中无需参与到专用芯片的代工制造。但是，在详细讨论基于 FPGA 构建的 μP 系统之前，让我们简要回顾一下微处理器的重要架构选择。

微处理器的基础概况

通常情况下，微处理器分为三大类：通用或 CISC 处理器、精简指令集处理器(Reduced Instruction Set Processor，RISC)以及微控制器。现在让我们简单了解一下这些类别的微处理器的发展历程和它们之间存在的主要区别。

1968 年，人们使用的典型的通用小型计算机是 16 位架构，在一块电路板上使用约 200 颗 MSI 芯片。每颗 MSI 芯片上大约有 100 个晶体管。当时一个流行的问题[Maz95]是：我们是否也可以(只)用 150、80 或 25 个芯片制造单个 CPU?

大约在同一时间，原来在 Fairchild 公司工作的罗伯特 • 诺伊斯(Robert Noyce)和戈登 • 摩尔(Gordon Moore)创办了一家新公司，最初名为 NM 电子公司，后来改名为英特尔，其主要产品是内存芯片。1969 年，日本的一家计算器制造商 Busicom 请英特尔为其新的可编程计算器系列设计一套芯片。英特尔没有足够的人力来制造 Busicom 要求的 12 种不同的定制芯片，因为当时优秀的 IC 设计师是很难找到的。作为替代方案，英特尔的工程师泰德 • 霍夫(Ted Hoff)建议建造一套更为通用的 4 芯片组，它可以从一片存储芯片中获取指令。一个带有存储器的可编程状态机(Programmable State Machine，PSM)便诞生了，如今我们称之为*微处理器*。在 F. Faggin 的帮助下，9 个月后，霍夫小组交付了英特尔 4004，它是一款 4 位 CPU，可用于 Busicom 计算器中使用的 BCD 算术运算。4004 使用 12 位程序地址和 8 位指令，执行一条指令需要花费五个时钟周期。一个最基本的可以工作的系统(包括 I/O 芯片)可以只用两个芯片来构建：4004 CPU 和一个程序 ROM。1 MHz 的时钟频率允许以每个数字 80 ns 的速度进行多位 BCD 数字的加法运算[FHM96]。

霍夫当时的设想是将 4004 的用途从计算器扩展到数字秤、出租车计价器、加油泵、电梯控制、医疗仪器、自动售货机等应用领域。因此，他说服英特尔管理层从 Busicom 获得授权，可以同时向其他公司出售芯片。1971 年 5 月，英特尔向 Busicom 公司做出了价格让步，同时作为交换获得了将 4004 芯片用于非计算器应用的销售权。当时的一个担忧是，4004 的性能无法与当时最先进的小型机竞争。但来自同为英特尔公司的 Dov Frohamn-Bentchkovsky 的另一项发明 EPROM，帮助 4004 开发系统进行了市场推广。有了它，程序不需要经过 IC 工厂生成 ROM，从而避免了由此带来的长期开发延迟。如果有必要，EPROM 可以由开发者多次编程和重新编程。

英特尔的一些客户要求有一款更强大的 CPU，于是，一款 8 位 CPU 被设计了出来，它可以处理 4004 的 4 位 BCD 算术运算。英特尔决定制造 8008，它也支持标准的 RAM 和 ROM 器件；而且它不再像 4004 的设计那样需要定制存储器。8008 设计中的一些缺点在 1974 年的 8080 设计中得到了修正，该设计当时使用了大约 4500 个晶体管。1978 年，第一款 16 位的 μP 推出了，即 8086。1982 年，80286 紧随其后：一个 16 位的 μP，但性能大约是 8086 的六倍。1985 年推出的 80386，是第一个支持多任务的 μP。在 80387 中，增加了一个数学协处理器，以加快浮点运算的速度。然后在 1989 年，80486 推出，它带有指令缓存和指令流水线，以及一个用于浮点运算的数学协处理器。1993 年，奔腾系列问世，它有两条流水线用于执行指令，即超标量架构。1997 年推出的下一代奔腾 II 增加了多媒体扩展(Multimedia Extension，MMX)指令，可以执行一些类似于 MAC 的并行矢量操作，最多有四个操作数。随后的奔腾 3 和奔腾 4 具有更先进的功能，如超线程和 SSE 指令，以加快音频和视频处理。2006 年出现的最大规模的处理器是英特尔安腾，它拥有两个处理器内核和高达 5.92 亿个晶体管，单是它的 L3 高速缓存就有 9 MB。英特尔处理器的完整系列见表 1.1。

表 1.1　英特尔微处理器系列

名称	推出年份	MHz	IA/比特数	制程/μm	晶体管数目
4004	1971	0.108	4	10	2300
8008	1972	0.2	8	10	3500
8080	1974	2	8	6	4500
8086	1978	5~10	16	3	29 K
80286	1982	6~12.5	16	1.5	134 K
80386	1985	16~33	32	1	275 K
80486	1989	25~50	32	0.8	1.2 M
奔腾	1993	60~66	32	0.8	3.1 M
奔腾 II	1997	200~300	32	0.25	7.5 M
奔腾 3	1999	650~1400	32	0.25	9.5 M
奔腾 4	2000	1300~3800	32	0.18	42 M
至强	2003	1400~3600	64	0.09	178 M
安腾 2	2004	1000~1600	64	0.13	592 M

IA 指令集架构

纵观半导体公司的收入，英特尔主要凭借微处理器这一种产品，多年来一直保持领先地位，这仍然令人印象深刻。其他以微处理器开发为主的公司，如德州仪器、摩托罗拉/飞思卡尔或 AMD 的收入要低得多。而其他顶级公司，如三星或东芝则以存储器技术为主导，但仍然没有获得英特尔那么高的收入，英特尔多年来一直处于领先地位。各顶级半导体公司的收入详见图 1.2。

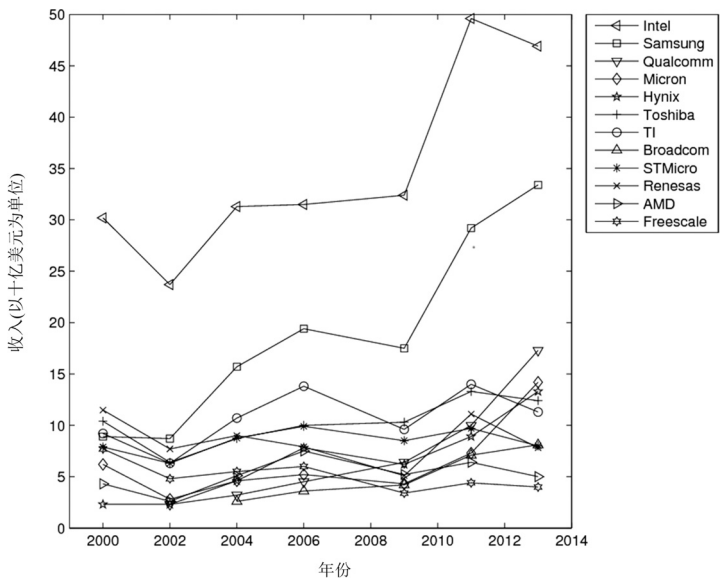


图 1.2　顶级半导体公司的收入

刚才讨论的英特尔架构有时被称为复杂指令集计算机(Complex Instruction Set Computer，CISC)。从早期的 CPU 开始，后续的设计就试图做到兼容，即能够运行相同的程序。随着数据和程序位宽的扩大，你可以想象，这种兼容性是以牺牲性能为代价的。仰仗着摩尔定律，这种 CISC 架构不断增加新的组件和功能(如数值计算协处理器、数据和程序缓存、MMX 和 SSE 指令以及支持这些新增功能以提高性能的相应指令)，从而大获成功。英特尔 μP 的特点就是有许多种指令并支持许多寻址模式。有关 μP 的手册其篇幅常常超过 700 页。

1980 年左右，加州大学伯克利分校(Patterson 教授)、IBM(后来称为 PowerPC)和斯坦福大学(Henessy 教授，其致力于后来称为无互锁流水线级微处理器的研究，即 MIPS μP 系列)的研究分析了 μP 并得出结论，从性能角度来看，CISC 机器存在若干问题。在伯克利，这种新一代的 μP 被称为 RISC-1 和 RISC-2，因为它们最重要的特征之一是具有有限的指令和寻址模式，从而被命名为精简指令集计算机(Reduced Instruction Set Computer，RISC)。现在我们简单回顾一下(至少是早期)RISC 和 CISC 机器具有的一些最重要的特征。

- CISC 机器的指令集很丰富，而 RISC 机器通常支持少于 100 条指令。
- RISC 指令的字长是固定的，通常为 32 位。在 CISC 机器中，字长是可变的。在英特尔机器中，指令的长度为 1~15 字节。
- CISC 机器支持丰富的寻址模式，而在 RISC 机器中只支持很少的寻址模式。典型的 RISC 机器只支持立即数寻址和寄存器基址寻址。
- CISC 机器中 ALU 操作的操作数可以来自指令字、寄存器或存储器。在 RISC 机器中，没有直接的存储器操作数可用，只允许向/从寄存器加载/存储，因此 RISC 机器被称为加载/存储架构。
- 在子程序中，参数和数据在 CISC 机器中通常通过栈链接；而 RISC 机器有大量的寄存器，用于将参数和数据链接到子程序。

到 20 世纪 90 年代初，显而易见的是，无论是 RISC 还是 CISC 架构，在其最纯粹的形式下，都不适合所有的应用。CISC 机器，尽管仍然支持许多种指令和寻址模式，但利用了更多的 CPU 寄存器和深度流水线。而今天的 RISC 机器，如 ARM、MIPS、PowerPC、SUN Sparc 或 DEC alpha，会有数百条指令，有些指令的执行需要花费多个时钟周期，几乎不再适配精简指令集计算机的称号。

虽然现代 32 位或 64 位 RISC 和 CISC 处理器可以高速执行复杂的任务，但我们发现，特别是在汽车和家电领域，还有另一套要求。在这些场景中，我们不需要实现高性能，也就是说，一个 8 位的处理器以几 kHz 或几 MHz 的速度运行就足够了，要使其运行过程消耗最少的电力(即电池寿命长)，还要是一个“完整的系统”，要具有片上程序 ROM 存储器、多输入/输出、最少的外部元件，有时还要在片上集成 ADC 和/或 DAC。这样的微处理器通常被称为微控制器，这些微小的工作机器以每年数十亿的数量被制造出来。8 位机已经成为最受欢迎的微控制器。每年的销售量约为 30 亿个，而 4 位或 16/32 位控制器的销售量为 10 亿个。4 位处理器通常不具备所需的性能，而 16 位或 32 位控制器对于许多应用来说通常过于昂贵或耗电。例如，在汽车中，

只有少数高性能微控制器需要用于音频或发动机控制；其他 50 多个微控制器用于诸如电动后视镜、安全气囊、速度表和门锁等功能，而许多这样的功能都可以由 8 位处理器来完成。

有时，RISC 和微处理器之间的区别变得有点模糊，因为微控制器也有一个小的称为“精简”的指令集。 然而，微控制器通常没有高流水线延迟，没有较大的寄存器文件，但有非常低的功耗，并有 PROM 和外围组件，如片上 ADC 或 DAC，这通常是 RISC 机器所没有的。

1.2 FPGA 上的嵌入式微处理器

当提及目前最先进的微处理器时，你可能会想到拥有 5.92 亿个晶体管的英特尔 Itanium 处理器。用 FPGA 来设计这种微处理器对今天的 FPGA 来说要求也许太高了。显然，今天的 FPGA 尚无法利用其可编程资源实现这样顶级的微处理器。尽管现在的一些 FPGA 系列有高性能的 RISC 硬核处理器，如 ARM Cortex-A9，但它仍然是一个嵌入式处理器，资源占用还是相当小的。不过，在许多应用中，功能有限的微处理器可以起到很大的作用。记住，同硬接线解决方案相比较，微处理器是舍弃性能换取了更高的门效率。在软件中，或者当设计为 FSM 时，像卷积这样的算法可能运行得比较慢，但通常需要用到的资源要少得多。因此，我们用 FPGA 构建的微处理器更像是微控制器类型，而不是功能齐全的现代英特尔奔腾或 TI VLIW PDSP。我们将在后面讨论一个典型的应用，该应用对一个简单游戏进行微控制器设计。现在你可能会争辩说，这可以用 FSM 来完成。是的，这是真的，我们基本上认为所用的 FPGA 微处理器设计不外乎是一个 FSM，再加上一个程序存储器，其中包含了 FSM 将执行的操作，如图 1.3 所示。事实上，Xilinx PICOBLAZE 处理器的早期版本被称为 Ken Chapman 可编程状态机(Ken Chapman Programmable State Machine，KCPSM)[Xil05a]。基于 FPGA 的嵌入式微处理器中存在的重点约束(我们将在后面的章节中详细讨论)可总结如下。

- 比起性能强大的通用处理器，嵌入式处理器的产量会是其 100 倍
- 由小型团队设计的嵌入式处理器，满足上市时间的硬性要求
- 硬性的性能/成本/上市时间问题
- 工具的一致性：架构设计→编译器开发→FPGA 实现流程中，建议使用高层次抽象工具
- 嵌入式处理器，通常更接近可编程 FSM→但并不适用于 Itanium(580M)
- 基于应用的指令集设计
- μP 使用 FPGA 的(小型)片上块存储器→经常用汇编语言编程
- 使用平面存储器以获得高性能
- 使用 C→HDL 编译器或自定义指令以提高性能

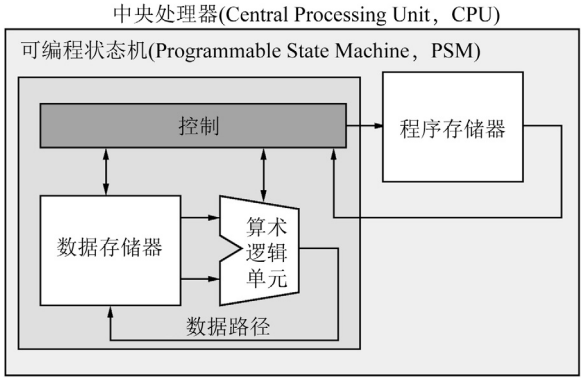


图 1.3　典型的基于 FPGA 的微处理器：Xilinx KCPSM 即 PicoBlaze

一套完整的微处理器设计通常包含多个步骤，例如，架构探索阶段、指令集设计，以及开发工具。涉及微处理器的指令集和 μP 功能细节方面的设计决定可能包括：

- 数据路径和程序字长
- 寄存器的数量和大小
- 定点/浮点支持
- 寻址模式及其生成方式
- 流水线级

我们将在下一节详细讨论这些选择。我们鼓励读者另外学习计算机体系结构方面的书籍；现在有很多这方面的书籍，因为这是大多数本科计算机工程课程的标准主题[BH03, HJ04, HP03, MH00, PH98, Row04, Sta02]。

1.3　微处理器指令集设计

微处理器(μP)的指令集描述了 μP 可以执行的动作集合。通常，设计者首先要问的是，在我所用的微处理器的应用中，我需要用到哪种算术操作？例如，对于数字信号处理(Digital Signal Processing，DSP)的应用，就要特别注意支持(快速)加法和乘法运算。在需要进行大量 DSP 处理的场合，由一系列较小移位加法完成的乘法可能不会是好选择。除了这些 ALU 操作，我们还需要用到一些数据移动指令和一些程序流类型的指令，如分支或 goto。

指令集的设计主要取决于底层的 μP 架构。如果不考虑硬件要素，就不能完整地定义我们所用的指令集。指令集的设计是一项复杂的任务，所以把开发分成几个步骤是个好主意。我们将通过回答以下问题来进行设计：

1. μP 支持哪些寻址模式？
2. 底层的数据流结构是什么，即一条指令涉及多少个操作数？
3. 我们在哪里可以找到每个操作数(如寄存器、内存、端口)？

4. 支持哪些类型的操作？

5. 在哪里可以找到下一条指令？

寻址模式

寻址模式描述了操作数的定位方式。CISC 机器可能支持许多不同的模式，而 RISC 或者 PDSP 等为性能服务的设计则要求限制最常用的寻址模式。现在让我们看看最常被支持的模式。

隐式寻址 在隐式寻址模式中，操作数来自或去往一个隐含的位置，而不是由指令显式定义的位置，见图 1.4。一个例子是栈机(stack machine)中使用的 ADD 操作(没有列出操作数)。栈机中的所有算术运算都是使用栈的两个顶层元素进行的。另一个例子是 PDSP 的 ZAC 操作，它清零了 TMS320 PDSP 中的累加器和乘积寄存器[TI83]。表 1.2 列出了不同微处理器中采用隐式寻址的一些例子。

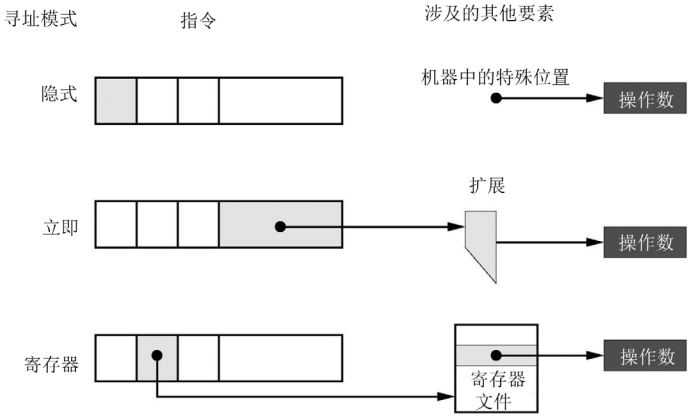


图 1.4 隐式、立即和寄存器寻址

表 1.2 不同微处理器中采用隐式寻址的示例

指令	描述	μP
RETURN	pc 加载了栈寄存器的值	PicoBlaze
bret	将 b 的状态复制到状态寄存器中，并在 pc 中加载 ba 寄存器的值	Nios II
imm 0	将下一条指令中立即操作数寄存器的高 16 位设置为 0	MicroBlaze
eret	异常返回，即把 lr 加载到 pc	ARM

立即寻址 在立即寻址模式中，操作数(即常量)就包含在指令本身之中。图 1.4 中展示了这一点。此处存在一个小问题，即一个指令字中提供的常量通常比 μP 中使用的(完整的)数据字的位数小。以下几种方法可以解决这个问题。

(a) 使用符号扩展，因为程序中使用的大多数常量(如增量或循环计数器)都很小，反正不需要用到完整位长。我们应该使用符号扩展(即复制 MSB 到字的高位)，而不是零扩展，

这样负值(如−1)就能被正确扩展。

(b) 使用两条或更多的独立指令，一个接一个地加载常量的低位和高位部分，并将这两部分串联成一个全长的字(例如，ARM 中的 MOV 和 MOVT)。

(c) 使用双字长指令格式来访问长常量。如果我们将默认的指令大小延长一个字，这通常可以提供足够的比特位来加载一个全精度的常量。这通常是由所谓的伪指令完成的，这些指令被映射到低/高指令。

(d) 使用操作数的桶形(barrel)移位对齐，将常量按所需方式对齐。

表 1.3 列出了不同微处理器中采用立即寻址的一些例子。

表 1.3　不同微处理器中采用立即寻址的示例

指令	描述	μP
LOAD s1, 05	将数据 5 加载到寄存器 1 中	PicoBlaze
roli r6, r6,1	将寄存器 6 循环移动 1 位	Nios II
lwi r3, r19, 4	将 r3 中的值存储到 r19+4 指向的内存地址	MicroBlaze
subs r3, r3, #1	寄存器 3 的值减 1	ARM
ldr r2, [r6, r8, lsl #4]!	将 r6+r8<<4 指向的内存的数据加载到寄存器 2 中	ARM

为了避免总是要进行两次内存访问，也可以将方法(a)符号扩展和(b)高/低位寻址结合起来。然后，我们只需要确保在低地址字的符号扩展后再进行高地址字的加载。

寄存器寻址　在寄存器寻址模式中，操作数从 CPU 内部的寄存器中访问，不发生外部存储器访问，见图 1.4。表 1.4 列出了不同微处理器中采用寄存器寻址的一些例子。

表 1.4　不同微处理器中采用寄存器寻址的示例

指令	描述	μP
AND sA, sB	sA 和 sB 进行按位逻辑 AND 运算，结果存到 sA 寄存器中	PicoBlaze
addk r19, r1, r0	将 r1 和 r0 的和存入 r19，并保留进位标志	MicroBlaze
xor r6, r7, r8	寄存器 7 和 8 进行 XOR 并把结果存到寄存器 6	Nios II
mul r2, r4, r5	寄存器 4 和 5 相乘并把结果放到寄存器 2	ARM

由于在大多数机器中，寄存器访问比常规的内存访问要快得多，消耗的功率也更少，因此这是 RISC 机器中经常使用的一种模式。事实上，RISC 机器中的所有算术运算通常只通过 CPU 寄存器完成；内存访问只允许通过单独的加载/存储操作来进行。

存储器寻址　为了访问外部存储器，通常使用直接、间接和组合模式。在直接寻址模式中，指令字的一部分指定了要访问的存储器地址，见图 1.5。这里出现了和立即寻址相同的问题：指令字中提供的用于访问存储器操作数的位数太小，无法指定完整的存储器地址。完整的地址长度可以通过一个显式或隐式指定的辅助寄存器来构建。如果将辅助寄存器的值加到直接存储器

地址上，则称作基址寻址，见下面的 ldbu 例子。如果辅助寄存器的值只是用来提供缺失的 MSB，这被称为分页寻址，因为辅助寄存器支持我们指定一个可以访问数据的页面。如果需要访问页面之外的数据，我们需要首先更新页指针。由于寄存器在基址寻址模式下代表一个全长的地址，因此也可以在没有直接存储器地址的情况下使用该寄存器。这就是所谓的间接寻址，见下面的 `str` 例子。表 1.5 列出了不同微处理器中采用的典型存储器寻址模式的例子。

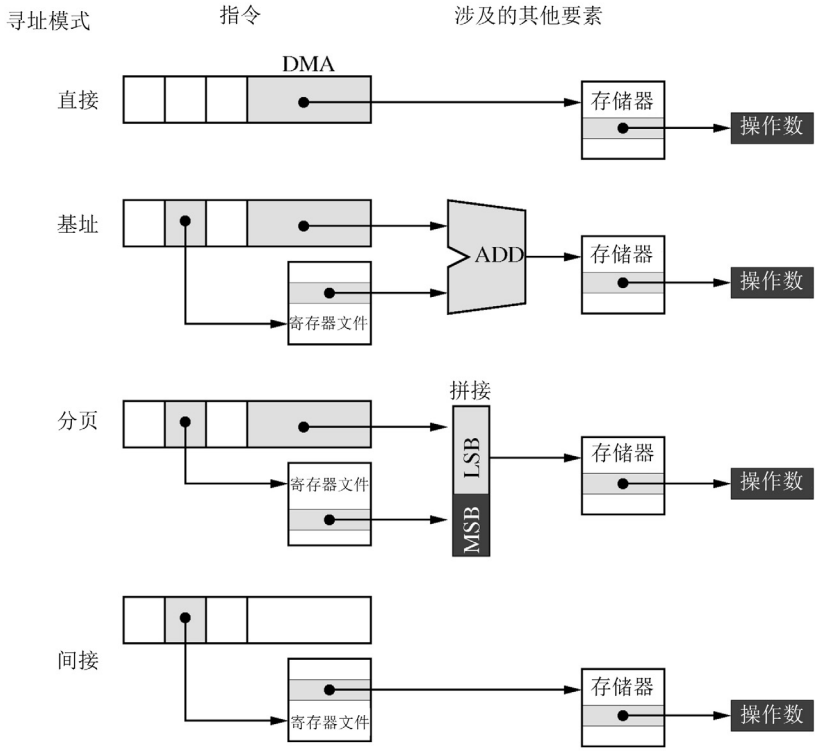


图 1.5　存储器寻址：直接、基址、分页和间接

表 1.5　不同微处理器中采用典型存储器寻址的示例

指令	描述	μP
`FETCH s4, 3F`	将便笺内存位置 `3F` 的数据读入寄存器 `4`	PicoBlaze
`ldbu r6, 100(r5)`	将寄存器 `r5` 中的地址加上 `100` 得到目标地址，将该地址中的数据读入寄存器 `r6`	Nios II
`str r1, [r0]`	将寄存器 `r1` 中的值存储到由寄存器 `r0` 指定的地址	ARM

由于间接寻址模式通常只能指向有限的索引寄存器，而这通常会缩短指令字数，因此它也是 RISC 中最流行的寻址模式。在 RISC 机器中，基址寻址也更受青睐，因为它允许指定基地址和数组元素的偏移量来方便地访问数组，见上面的 `ldbu` 例子。

数据流架构

一条典型指令的汇编编码首先会列出操作码，然后是操作数。一个典型的 ALU 操作需要用到两个操作数，如果我们还想指定一个单独的结果位置，自然的方法是让指令字有一个操作码，跟着是三个操作数，这就让程序员易于编程。然而，三个操作数的设计可能需要构建一个很长的指令字。一个现代的 CPU 在寻址 4 GB 的数据时需要用到 32 位的地址，而直接寻址的三个操作数至少需要构建一个 96 位的指令字，且不计操作码的位数。因此，限制操作数可以减少指令字的长度，从而节省资源。在这方面，使用零地址或栈机是完美的。另一种方法是使用寄存器文件，而不是直接访问内存，只允许加载/存储单个操作数，这是 RISC 机器的典型做法。对于一个有 8 个寄存器的 CPU 来说，只需要使用 9 个比特来指定三个操作数。但是我们还需要使用额外的操作码来加载/存储寄存器文件中的数据存储器数据。在下面的章节中，我们将讨论在指令字中允许使用 0~3 个操作数时对硬件资源和指令集的影响。

栈机：零地址的 CPU　你可能会问，零地址的机器，这怎么可能工作？我们需要回顾一下寻址模式，操作数可以隐式地在指令中指定。例如，在 TI PDSP 中，所有的乘积都存储在乘积寄存器 P 中，这不需要在乘法指令中指定，因为所有的乘法结果都会存入 P。同样，在零地址或栈机中，所有的双操作数运算都是用栈的两个顶层元素进行的[Koo89]。顺便说一下，栈可以被认为是一个后进先出(Last In First Out，LIFO)队列。当我们使用 `POP` 操作时，弹出来的第一个元素是我们用指令 `PUSH` 放在栈顶上的。让我们简单地分析一个类似于这样的表达式

$$e=a-b+c\times d \tag{1.1}$$

在栈机上进行运算。表 1.6 的左边是指令，右边是四元素栈中包含的内容。栈顶在左边。

表 1.6　表达式在栈机上的运算过程

指令	栈			
	顶	2	3	4
`push a`	*a*	—	—	—
`push b`	*b*	a	—	—
`sub`	*a*−*b*	—	—	—
`push c`	*c*	*a*−*b*	—	—
`push d`	*d*	*c*	*a*−*b*	—
`mul`	*c*×*d*	*a*−*b*	—	—
`add`	*c*×*d*+*a*−*b*	—	—	—
`pop e`	—	—	—	—

可以看出，所有的算术运算(`ADD`、`SUB`、`MUL`)都使用了隐式指定的栈顶和第二个元素作为操作数，这实际上是零地址指令的操作。然而，内存操作 `PUSH` 和 `POP` 需要使用一个操作数。

栈机的代码被称为后缀(或逆波兰)运算，因为首先指定了操作数，然后进行运算。如式(1.1)中所示的标准算术符号被称为中缀记法，例如，我们有如下两个等价的表示法。

$$\text{中缀：}a-b+c\times d \leftrightarrow \text{后缀：}ab-cd\times+ \tag{1.2}$$

在练习(1.45-1.47)中，展示了应用这两种不同算术模式时的更多例子。有些人可能记得，后缀记法与 HP41C 袖珍计算器所要求的编码方式完全相同。HP41C 也使用了一个包含 4 个值的栈。图 1.6a 展示了栈机的机器架构。

累加器机：单地址的 CPU　现在我们在 CPU 上增加一个累加器，并把这个累加器作为单个操作数的源以及结果的目标。算术运算的形式为

$$\text{accu} \leftarrow \text{accu} \square \text{op1} \tag{1.3}$$

其中□描述了一个 ALU 操作，如 ADD、MUL 或 AND。TI TMS320 [TI83]系列 PDSP 的底层结构就是这种类型，如图 1.6b 所示。例如，在 ADD 或 SUB 运算中，会指定一个操作数。上一节所举例子对应的表达式(1.1)在 TMS320C50[TI95]汇编器代码中对应的编码如表 1.7 所示。

表 1.7　式(1.1)在汇编器代码中对应的编码

指令	描述
ZAP	;清零 accu 和乘法寄存器
ADD DAT1	;将 DAT1 加到 accu
SUB DAT2	;从 accu 减去 DAT2
LT DAT3	;将 DAT3 加载到 T 寄存器
MPY DAT4	;将 T 和 DAT4 相乘存入 P 寄存器
APAC	;将 P 寄存器加到 accu
SACL DAT5	;存储 accu (低位) 到地址 DAT5

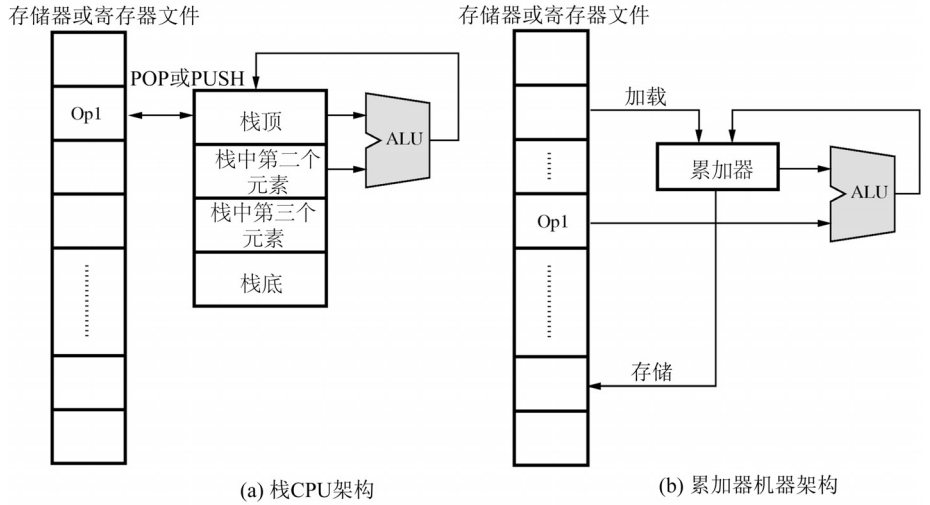


图 1.6　栈机与累加器机的机器架构

这个例子假设变量 a-e 被映射到数据内存字 DAT1-DAT5。对比栈机和累加器机，我们可以得出以下结论：

- 指令字的大小没有改变，因为栈机也需要使用包含了操作数的 POP 和 PUSH 操作。
- 编码一个代数表达式所需的指令数量没有从本质上减少(累加器机为 7 条，栈机为 8 条)。

当我们使用双操作数机器时，编码代数表达式所需的指令数量预计会显著减少，这一点将在接下来讨论。

双地址 CPU　在双地址机器中，允许我们在算术运算中独立指定两个操作数，目的操作数也是第一个操作数，即操作的形式为

$$op1 \leftarrow op1 \square op2 \tag{1.4}$$

其中□描述了一个 ALU 操作，如 SUB、DIV 或 AND。Xilinx 公司的 PICOBLAZE[Xil02a, Xil05a]和 Altera 公司的第一代 Nios 处理器[Alt03a]就使用了这种数据流。基本的数据流结构如图 1.7a 所示。对两个操作数的限制使得在这些情况下可以使用 16 位指令字格式。我们列出的式(1.1)对应的编码在采用了 PICOBLAZE[1]风格的微处理器的汇编程序中的表示如表 1.8 所示。

表 1.8　式(1.1)对应的编码在采用了 PICOBLAZE 风格的微处理器的汇编程序中的表示

指令	描述
LOAD sE, sC	;将寄存器 C 的值存入寄存器 E
MUL sE, sD	;将寄存器 D 的值和寄存器 E 的值相乘
ADD sE, sA	;将寄存器 A 的值加到寄存器 E
SUB sE, sB	;从 E 中减去寄存器 B 的值

为了避免出现乘积的中间结果，有必要对操作进行重排。请注意，PICOBLAZE 没有单独的 MUL 操作，因此该代码仅用于演示两个操作数的原理。PICOBLAZE v6 使用了 2×16 个寄存器，每个寄存器位宽都是 8 位。有了两个操作数和 8 位的常数值，我们可以将操作码和操作数或常数放在一个 16 位或 18 位的数据字中。你还可以看到，与栈机或累加器机相比，双操作数编码实质上减少了操作的数量。

三地址 CPU　三地址机器是所有机器中最灵活的。两个操作数和目标操作数可以来自或去往不同的寄存器或内存位置，也就是说，操作的形式为

$$op1 \leftarrow op2 \square op3 \tag{1.5}$$

大多数现代 RISC 机器，如 ARM、PowerPC、MicroBlaze 或 Nios II，都偏爱使用这种类型的编码[Alt03b, Xil02b, Xil05b]。而操作数通常是寄存器操作数，或者最多只有一个操作数可以来自数据存储器。数据流架构如图 1.7b 所示。

1 现在的 PICOBLAZE 使用 18 位指令字，在其 ISA 中仍然没有乘法运算。

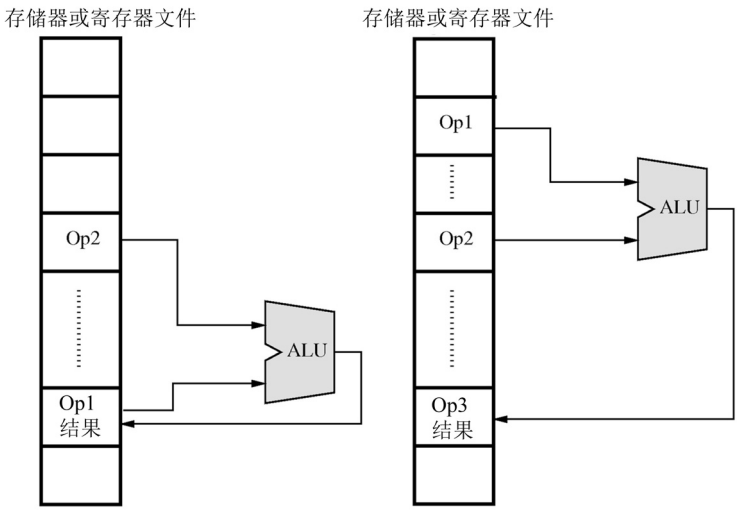


图 1.7　双地址和三地址 CPU 架构

用汇编语言对三操作数机器进行编程是一项直截了当的任务。我们列出的式(1.1)在 Nios II 机器上对应的编码看起来如表 1.9 所示。

表 1.9　式(1.1)在 Nios II 机器上对应的编码

指令	描述
`sub r5, r1, r2`	;将 r1 寄存器的值减去 r2 的值并存入 r5
`mul r6, r3, r4`	;将寄存器 r3 和 r4 的值相乘并存入 r6
`add r5, r5, r6`	;将 r5 和 r6 的值相加并存入 r5

假设寄存器 r1-r5 保存了变量 a 到 e 的值，这是迄今为止我们讨论过的所有 4 种机器中长度最短的代码。付出的代价则是具有较长的指令字。在硬件实现方面，我们不会看到双操作数机和三操作数机之间有太大的区别，因为无论如何，寄存器文件都需要使用单独的复用器和解复用器。

零、单、双和三地址 CPU 的比较

为了了解 4 种 CPU 架构之间的区别，我们先做一个简短的并列对比，要计算式(1.1)，即 e=a-b+c×d，我们假设存在以下指令集。

- 0-AC(栈)：PUSH Op1，POP Op1，ADD，SUB，MUL，DIV
- 1-AC(累加器)：LA Op1，STA Op1，ADD Op1，SUB Op1，MUL Op1，DIV Op1
- 2-AC：LD Op1, M；ST Op1, M；ADD Op1, Op2；SUB Op1, Op2；MUL Op1,Op2；DIV Op1, Op2

- 3-AC：LD Op1, M；ST Op1, M；ADD Op1, Op2, Op3；SUB Op1, Op2, Op3；MUL Op1, Op2, Op3；DIV Op1, Op2, Op3

我们假设 2-AC 和 3-AC 的操作数“a”到“d”已经被加载到寄存器中，但是最终结果应该被存储到存储器当中，而 2-AC 和 3-AC 机器中原来的寄存器值应该被保留下来。表 1.10 是对这 4 种机器进行的并列比较。

表 1.10　零、单、双和三地址 CPU 架构的并列比较

0-AC：栈	1-AC：累加器	2-AC：(a=sA;b=sB;…)	3-AC：(r1=a;r2=b; …)
PUSH a	LA a	LD sE,sC	SUB r5,r1,r2
PUSH b	SUB b	MUL sE,sD	MUL r6,r3,r4
SUB	STA t	ADD sE,sA	ADD r5,r5,r6
PUSH c	LA c	SUB sE,sB	ST r5,e
PUSH d	MUL d	ST sE,e	
MUL	ADD t		
ADD	STA e		
POP e			

下面总结一下我们的发现。

- 栈机有最长的程序和最短的单条指令。
- 即使是栈机，也需要使用一条单地址指令来访问存储器。
- 三地址机的代码最短，但每条指令需要的比特数最多。
- 寄存器文件可以减少指令字的大小。一般来说，在三地址机中，支持使用两个寄存器和一个存储器操作数。
- 加载/存储机器只允许在存储器和寄存器之间移动数据。任何 ALU 操作都是通过寄存器文件完成的。
- 大多数设计假定寄存器访问比存储器访问快。虽然在基于单元的 IC(Cell Based IC，CBIC)或使用了外部存储器的 FPGA 中确实如此，但在 FPGA 内部，寄存器文件访问和嵌入式存储器访问时长处于相同的范围，这就提供了用嵌入式(三端口)存储器来实现寄存器文件的选择。

上述发现并不令人满意，但似乎没有最好的选择，每一种风格在实践中都会被使用，正如我们举出的编码例子所显示的那样。那么问题来了：为什么没有哪一种特定的数据流类型脱颖而出成为最佳选择？这个问题的答案并不简单，因为需要考虑许多因素，如编程的难易程度、代码的长短、处理的速度和硬件要求。下面我们根据不同的设计目标来比较不同的设计，其总结见表 1.11。

汇编程序编码的难易程度与指令的复杂度成正比。三地址的汇编代码比我们在栈机汇编编

码中见到的许多 PUSH 和 POP 操作要易读易写得多。另一方面，针对栈机来设计简单的 C/C++ 编译器要简单得多，因为它易于采用后缀操作，而后缀操作易于用解析器分析。对于编译器来说，以高效的方式管理寄存器文件是一项非常困难的任务。而流水线化则进一步增加了高效使用寄存器文件的复杂度。算术操作中包含的代码字数对于双地址和三地址操作来说要短得多，因为中间结果很容易被计算出来。指令的长度与操作数的个数直接成正比。这一点可以通过使用寄存器而不是直接的内存访问来简化，但更少的操作数仍然会带来短得多的指令长度。可以存储的立即操作数的大小取决于指令的长度。在指令字数较短的情况下，可以嵌入指令的常数也较短，这样我们就可能需要使用多条加载或双字长的指令，见上面的存储器寻址部分。如果涉及的操作数较少，那么取操作数及解码的速度就会更快。由于栈机总是使用栈的两个顶层元素，因此不会出现由寄存器文件的 MUX 或 DEMUX 而导致的长时间延迟。硬件大小主要取决于寄存器文件。三操作数的 CPU 要求最高，栈机最小；ALU 和控制单元的大小则相似。

表 1.11　从零到三操作数 CPU 的不同设计目标的比较

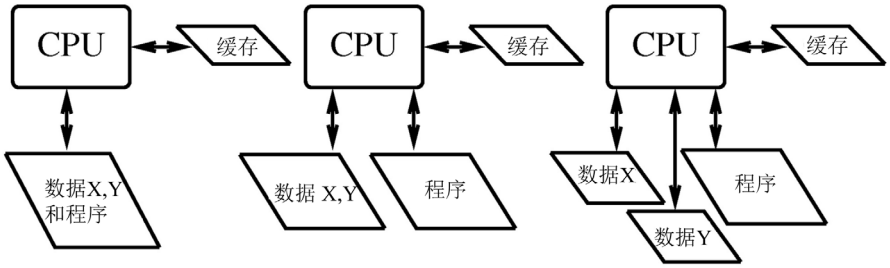
目标	操作数个数			
	0	1	2	3
汇编难易程度	最差	…	…	最好
简单的 C/C++编译器	最好	…	…	最差
代码字数	最差	…	…	最好
指令长度	最好	…	…	最差
立即数范围	最差	…	…	最好
操作数取数和解码速度	最好	…	…	最差
硬件大小	最好	…	…	最差

总之，我们可以说每个特定的架构都有其优缺点，它们必须与设计者的工具、技能、尺寸/速度/功率方面的设计目标以及开发工具(如汇编器、指令集模拟器或 C/C++编译器)相匹配。

寄存器文件和存储器架构

在计算机出现的早期，当存储器很昂贵的时候，冯诺依曼提出了备受赞誉的一项创新：将数据和程序放在同一个存储器中，见图 1.8a。当时的计算机程序通常是在 FSM 中硬连接的，只有数据存储器使用 RAM。现在的技术约束则不同：存储器很便宜，但是对于典型的 RISC 机器来说，其访问速度仍然比 CPU 的寄存器慢得多。因此，在三地址机中，我们需要考虑三个操作数应该来自哪里。是否应该允许所有的操作数来自主存储器，或者只允许其中的两个或一个，或者我们应该实现一个加载/存储架构，只允许在寄存器和存储器之间进行单次传输，但要求所有的 ALU 操作都通过 CPU 寄存器完成？在这方面，VAX PDP-11 一向被誉为冠军，它支持多存储器和多寄存器操作。对于 FPGA 设计，存在一个额外的限制条件，即指令字的数量通常数以千计，冯诺依曼的方式并不是一种好选择。如果我们使用独立的程序和数据存储器，就可以避免

由于数据和程序字复用所造成的时间浪费。这就是所谓的哈佛架构，见图 1.8b。对于 PDSP 的设计，如果能够使用三个不同的存储器端口，那就更好了(想想矢量乘积)：系数和数据来自两个独立的数据存储器位置 *x* 和 *y*，而累加的结果则保存在 CPU 寄存器中。还需要用到第三个存储器作为程序存储器。由于许多算法使用短循环，一些处理器试图通过实现一个小型缓存来节省第三条总线。在循环第一次运行后，指令就存储在缓存中，程序存储器则可以作为第二个数据存储器使用。这种三总线结构如图 1.8c 所示，通常被称为超级哈佛架构。



(a) 冯・诺依曼机器(GPP)　(b) 哈佛架构，数据和地址总线分离　(c) 超级哈佛架构，有两条数据总线

图 1.8　存储器架构

像英特尔的奔腾或 RISC 机器这样的 GPP 机器通常使用存储器层次结构为 CPU 提供连续的数据流，但也允许人们使用较便宜的存储器来处理主要数据和程序。这样的存储器层次结构从非常快的 CPU 寄存器开始构建，然后是一级、二级数据和/或程序缓存，再到主 DRAM 存储器，以及 CD-ROM 或磁带等外部介质。这种存储器系统的设计要比我们在 FPGA 内的设计复杂得多。

从硬件实现的角度来看，CPU 的设计可以分成如下三个主要部分。

- 控制路径，即有限状态机
- ALU
- 寄存器文件

在这三部分中，尽管寄存器文件设计起来并不困难，但在用标准逻辑资源实现时，往往却是成本最高的一个模块。从这些高的实现成本来看，我们似乎需要在使用更多寄存器以使 μP 易用和更大的文件(如 32 个寄存器)带来的高实现成本之间进行折中。当设计一个 RISC 寄存器文件时，通常需要实现更多的寄存器。一种用于节省寄存器文件逻辑资源的选项是使用两个嵌入式双端口存储器块作为寄存器文件。我们可在两个存储器中写入相同的数据，并可以从存储器的另一个端口读取源数据。这一思路已经在 Nios μP 中采用，可以大大减少所需的逻辑资源。然而从时序要求来看，现在出现了一个问题，即 Block RAM 是同步存储器块，我们不能从两个端口的同一时钟沿来加载并存储存储器地址和数据，也就是说，在同一时钟沿不可以用当前的解复用器值替换同一个寄存器的值。不过，我们可以先在上升沿指定要加载的操作数地址，然后在下降沿存储新值并置位写使能。

为了避免因使用间接寻址(偏移量为 0)或清零一个寄存器(两个操作数都为 0)或寄存器移动指令而出现所需的额外指令，有时将第一个寄存器永久地设置为 0，在 MicroBlaze 和 Nios II

中就是这样。这对于一个没有多少寄存器的机器来说似乎是一种很大的浪费，但却从根本上简化了汇编代码，如表 1.12 中的例子所示。

表 1.12　简化后的汇编代码示例

指令	描述
`add r0, r0, r0`	;NOP，即什么也不做
`add r3, r0, r0`	;将寄存器 r3 的值设为 0
`add r4, r2, r0`	;将寄存器 r2 的值存到寄存器 r4
`ldbu r5, 100(r0)`	;从地址 100 加载数据到 r5

注意，以上伪代码只有在第一个寄存器 r0 的值为 0 的前提下才能完成任务。

操作支持

大多数机器至少使用以下三类指令中的一类：算术/逻辑单元(Arithmetic Logic Unit，ALU)、数据移动，以及程序控制。下面我们简单回顾一下每个类别对应的一些典型例子。底层的数据类型通常是字节的倍数，即 8、16 或 32 位的整型数据类型；一些更复杂的处理器使用 32 或 64 位的 IEEE 浮点数据类型。

ALU 指令　ALU 指令包括算术、逻辑和移位操作。其支持的典型双操作数算术指令有加(ADD)、减(SUB)、乘(MUL)或乘积累加(MAC)。对于单个操作数，绝对值(ABS)和符号反转(NEG)是最低限度也要提供支持的指令。除法操作通常由一系列移位-减法-比较指令完成，因为一个阵列除法器可能相当大。

移位操作的用处在于，在 b 位整型算术运算中，每次乘法运算后位数增长到 $2b$。移位器可以像 TI 公司的 TMS320 PDSP 那样是隐含的，或者作为单独的指令提供。通常会支持逻辑和算术运算(即正确的符号扩展)以及循环移位。在块浮点数据格式中，指数检测(即确定符号位的数量)也是必须支持的操作。

表 1.13 列出了不同微处理器具有的算术和移位操作。

表 1.13　不同微处理器具有的算术和移位操作

指令	描述	μP
`XOR s3, FF`	将 s3 和 FF 按位 XOR，即翻转 s3	PicoBlaze
`div r3, r2, r1`	r2 除以 r1，并把商存到寄存器 r3 中	Nios II
`mul r2, r4, r5`	r4 和 r5 相乘；把乘积的 32 位 LSB 存入 r2	MicroBlaze
`lsl r2, r5, #3`	将 r5 左移 3 位并填 0，也就是乘以 8，结果存入 r2	ARM

对于像 PICOBLAZE 这样字长较短的 μP 来说，了解进位输入、进位输出和零位标志如何被指令使用和更新也很重要。这些标志是用来建立大于原始字长的长字算术运算的。尽管逻辑运

算不像算术运算那样常用，但较长的 IF 条件求值和一些使用了密码学或纠错算法的更复杂的系统需要用到一些基本的逻辑运算，如 AND、OR、NOT 或 XOR。如果指令条数很关键，我们也可以使用单一的 NAND 或 NOR 运算，所有其他的布尔运算都可以从这些通用函数中衍生出来。

数据移动指令　由于地址空间的增大和性能方面的考量，大多数机器更接近于典型的 RISC 加载/存储架构，而不是采用 VAX PDP-11 那种允许指令的所有操作数都来自存储器的统一方式。在加载/存储哲学中，只支持在存储器和 CPU 寄存器间或不同的寄存器间使用数据移动指令——存储器地址不可以成为 ALU 操作的一部分。通常会使用寄存器间接寻址方式来访问存储器，可能还支持额外的操作前或者操作后减/增。

表 1.14 列出了不同微处理器具有的数据移动操作。

表 1.14　不同微处理器具有的数据移动操作

指令	描述	μP
`FETCH s2, 03`	将暂存器位置 3 处的数据加载到寄存器 s2	PicoBlaze
`ldw r4, 4(r6)`	从内存位置 r6+4 处加载数据到寄存器 r4	Nios II
`str r2, [r4]`	将 r2 中的值存储到 r4 指定的存储器地址	ARM

程序流程指令　控制流相关的指令组中包括支持用于实现循环、调用子程序或跳转到特定程序位置的指令。我们也可以将 μP 设置为空闲状态，等待中断的发生，这表明有新的数据到达，需要进行处理。

更新的 μP 还支持更长的循环和几级的嵌套循环。在大多数 RISC 机器应用中，循环通常不像 PDSP 那样短，循环开销也不那么关键。此外，RISC 机器使用延迟分支间隙来避免流水线机器中出现的 NOP。函数和过程调用通常需要使用一个数据栈来保存寄存器、返回 PC 值和参数值。在开始进行软件开发之前，程序员应该知道嵌套调用的级别。

表 1.15 列出了不同微处理器具有的数据移动操作。

表 1.15　不同微处理器具有的数据移动操作

指令	描述	μP
`JUMP NZ, loop`	如果零标志没有置位，则跳到标签 loop	PicoBlaze
`CALL func`	调用名为 func 的函数	PicoBlaze
`braid 4b4`	无条件分支到十六进制地址 4b4；由于下一条指令操作位于延迟间隙中，因此它会被执行	MicroBlaze
`beq r5, r0, no_button`	比较寄存器 r5 和 r0，如果值相等跳到标签 no_button	Nios II
`bne wait`	如果前一个操作的相等(即 0)标志是 0，则跳到标签 wait	ARM

下一个操作的位置

理论上，我们可以提供包含了下一条指令字地址的第四个操作数来简化下一个操作的计算。

但由于几乎所有的指令都是一条接一条地执行的(除了跳转类指令)，因此这个地址大多会是冗余信息，至今，我们还没有发现有使用这种设计理念的商用微处理器。

只有当设计一个单指令类型的处理器，也就是终极 RISC(Ultimate RISC，URISC) 机器时，我们才需要在指令字中包括下一个地址，甚至更进一步，包括与当前指令相比的偏移量[Par05]。

1.4 FPGA 技术

VLSI 电路的分类如图 1.9 所示。FPGA 可归为现场可编程逻辑器件(Field Programmable Logic Device，FPLD)。FPLD 可定义为包含了小逻辑块和元素[1]的可编程器件，见图 1.10。可以认为，FPGA 也是一种 ASIC 技术，因为 FPGA 是特定于应用程序的 IC。然而，人们普遍认为经典 ASIC 的设计需要额外的半导体处理工序，已经超越了 FPLD 所需。额外的工序提供了更高阶的 ASIC 的性能和功耗优势，但也提高了其一次性工程成本(Non Recurring Engineering，NRE)。在 40 纳米条件下，NRE 的成本约为 400 万美元，见[BB08]。另一方面，门阵列通常由“NAND 门海”组成，其功能由客户在“线路表”中提供。在制造过程中使用线路表以实现最终金属层的明确定义。可编程门阵列解决方案的设计者则可完全控制实际的设计实现，而不需要利用任何物理 IC 制造设施及产生的相关延迟。更详细的 FPGA/ASIC 比较可以在有关 FPGA 的竞争技术一节中找到。

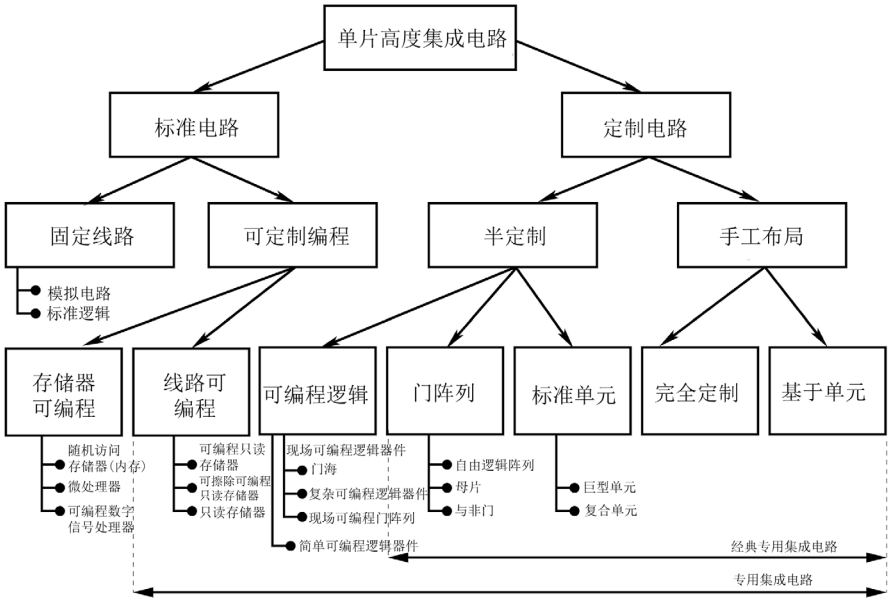


图 1.9 VLSI 电路分类

1 Xilinx 称为切片(slice)或者可配置逻辑块(Configurable Logic Block，CLB)，Altera 称为逻辑单元(Logic Cell，LC)，逻辑元素(Logic Element，LE)，或者自适应逻辑模块(Adaptive Logic Module，ALM)。

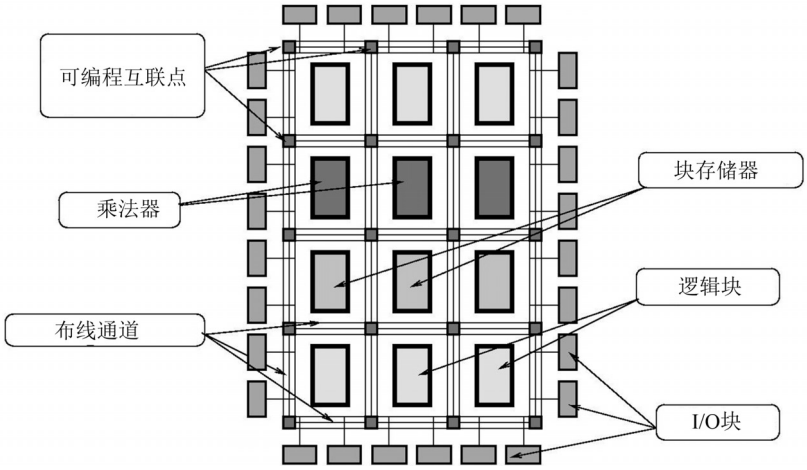


图 1.10　FPGA 架构

按粒度分类

逻辑块大小与器件的粒度(*granularity*)相关，而器件的粒度又与完成块之间的连接(布线通道)所需的工作量相关。通常，可以按粒度将器件分为三类。细粒度器件(*fine-grain device*)最初由 Plessey 授权而后由摩托罗拉授权，由 Pilkington 半导体公司提供。基本逻辑单元由一个 NAND 门和一个锁存器组成。因为使用 NAND 门可以实现任何二进制逻辑函数(见练习 1.39)，所以 NAND 门被称为万能(*universal*)函数。这种技术仍然被用于门阵列设计中，并且与经过认可的逻辑合成工具(如 ESPRESSO)一同被采用。门阵列 NAND 门之间的布线通过使用附加的金属层来达成。对于可编程架构来说，这将成为瓶颈，因为连接逻辑功能所需的路由资源将非常大。双输入 NAND 对 PLD 而言并不是一个好的选择。此外，要构建一个简单的电路，需要使用大量的 NAND 门。例如，一个快速的 4 位加法器要使用大约 130 个 NAND 门。这使得细粒度技术在实现大多数微处理器组件时缺乏吸引力。

最常见的 FPGA 架构如图 1.10 所示。现代中等粒度 FPGA 器件的具体例子如图 2.16 和 2.19 所示。基本逻辑块通常是小表(通常有 4~8 位输入表，1~2 位输出)，或者用专用的多路复用器(multiplexer，MPX)逻辑来实现，Actel ACT-2 器件中就使用了这种逻辑[GHB93]。路由通道选择范围从短到长。还有一个带有触发器的可编程 I/O 块附加在器件的物理边界上。

大粒度器件，如复杂可编程逻辑器件(Complex Programmable Logic Device，CPLD)，其特点是将几个所谓的简单可编程逻辑器件(Simple Programmable Logic Device，SPLD)结合在一起。SPLD 由一个用 AND/OR 阵列实现的可编程逻辑阵列(Programmable Logic Array，PLA) 和一个通用 I/O 逻辑块组成，该逻辑块包括一个存储元素，如锁存器或触发器。CPLD 中使用的 SPLD 通常有 8~10 个输入，3~4 个输出，并支持大约 20 个乘积项。在这些 SPLD 块之间，可以使用具有短延迟的宽总线(Altera 称为可编程互连阵列(Programmable Interconnect Array，PIA))。通过将总线和固定的 SPLD 时序相结合，可以在 CPLD 上提供可预测且短的引脚到引脚(pin-to-pin)延迟。SPLD 中所用的解码器通常小而快，而算术电路即使是中等位宽也需要用到 CPLD 的许多乘法

项，使得 SPLD 在实现大多数微处理器时不具有吸引力。

按技术分类

FPLD 在几乎所有的存储器技术中都可用：SRAM、EPROM、E^2PROM 和反熔丝[RGS93]。具体的技术定义了器件是可重新编程还是一次性可编程。大多数 SRAM 器件可以通过单比特流进行编程，这能降低布线要求，但也增加了编程时间(通常在毫秒范围内)，可能会引起 IP 窃取的担忧。SRAM 器件是 FPGA 的主导技术，它基于静态 CMOS 存储技术，并可以在系统内(in system)重新编程。不过，它们需要使用一个外部“引导”器件进行配置。电可编程只读存储器(Electrically Programmable Read Only Memory，EPROM)器件通常使用一次性 CMOS 可编程模式，因为需要使用紫外线擦除。CMOS 电可擦可编程只读存储器(Electrically Erasable Programmable Read Only Memory，E^2PROM)可用于系统内重编程。EPROM 和 EEPROM 的优点是设置时间短。因为编程信息没有被“下载”到器件上，所以可以更好地防止未经授权的使用。最近一项基于 EPROM 技术的创新被称为“闪存”存储器。这些器件通常被视为具有较小物理单元的“按页”系统内可编程系统，等效于 E^2PROM 器件。最后，表 1.16 总结了不同器件技术具有的重要优缺点。

表 1.16　FPLD 技术的属性

技术	SRAM	EPROM	E^2PROM	反熔丝	闪存
可重新编程	是	是	是	否	是
在系统内可重新编程	是	否	是	否	是
易失性	是	否	否	否	否
复制保护	否[a]	是	是	是	是
例子	Xilinx Zynq	Altera MAX5K	AMD MACH	Actel ACT	Xilinx XC9500
	Altere Cyclone	Xilinx XC7K	Altera MAX7K		Cypress Ultra 37K

a　一些现代 SRAM 使用 DES/AES 加密来保护配置文件

FPLD 的基准测试

为 FPLD 提供客观基准并非易事。性能通常取决于设计人员具有的经验和技能，以及设计工具的特性。为了建立有效的基准，Xilinx [Xil93]、Altera [Alt93]和 Actel [Act93]创办了可编程电子性能合作组织(Programmable Electronic Performance Cooperative，PREP)，后来扩展到十多个成员。PREP 为 FPLD 制定了 9 个不同的基准，列于表 1.17。基准测试的核心思想是每个供应商使用自己的器件和软件工具在指定的器件中尽可能多地实现基本块，同时试图将速度最大化。同一个逻辑块在一个器件中的实例化数量称为重复率(*repetition rate*)，是所有基准测试的基础。对于嵌入式微处理器的比较，表 1.17 中所示的小型和大型 FSM 设计的基准测试 3 和 4 是最相关的。

表 1.17　FPLD 的 PREP 基准测试

编号	基准测试名称	说明
1	数据通路	8 个 4 对 1 多路复用器驱动一个带并行加载的 8 位移位寄存器
2	定时器/计数器	两个 8 位的数值由 8 位数值寄存器给出时钟并加以比较
3	小状态机	具有 8 个输入和 8 个输出的 8 状态机(见图 3.14)
4	大状态机	具有 40 个状态转换、8 个输入和 8 个输出的 16 位状态机
5	算术电路	4×4 无符号倍增器和 8 位累加器
6	16 位累加器	一个 16 位累加器
7	16 位计数器	可加载二进制递增计数器(见图 3.15)
8	16 位同步预分频计数器	带异步复位的可加载二进制计数器(见图 3.15)
9	存储映射器	将 16 位地址空间解码为 8 个区域(见图 3.16)

在图 1.11 中，以频率为重复率的单位，报告了目前在大学开发板中常用的 Altera(A_k)和 Xilinx(X_k)FPGA 及 CPLD 器件。这些并不总是可用的最大器件，但所有器件都得到了基于 Web 的设计工具版本的支持。Xilinx 似乎实现了更快的速度，而 Altera FPGA 的重复率数字则更大。与 PLDC 相比，可以说，现代 FPGA 提供了最好的嵌入式微处理器资源和最快的速度。这是由于现代器件提供了延迟小于 0.1 ns / bit 的快速进位逻辑，它支持具有宽位的快速加法器，而不需要使用昂贵的“超前进位”解码器。尽管 PREP 基准测试在比较等效门数和最大速度时很有用，但对于具体应用，附加属性也很重要。它们包括：

- 片上大块 RAM 或 ROM
- 外部存储器支持 ZBT、DDR、QDR、SDRAM
- 嵌入式硬连线处理器系统(Hardwired Processor System，HPS)。例如，32 位 ARM Cortex-A9
- 阵列乘法器(例如，18×18 位，18×25 位)
- 封装如 BGA、TQFP、PGA
- 配置数据流加密，DES 或 AES
- 片上快速模数转换器(Analog to Digital Converter，ADC)
- 引脚到引脚延迟
- 内部三态总线
- 回读或边界扫描解码器
- 摆率或电压可编程的 I/O
- 功率耗散
- 硬 IP 块，支持×1、×2 或×4 PCIe

功耗是 FPLD 具有的另一个重要特性，尤其是在移动应用中。我们发现，CPLD 通常有较高的“待机”功耗。对于更高频率的应用，可以预期 FPGA 有更高的功耗。详细的功率分析示例见[MB14，第 1.4.2 节]。

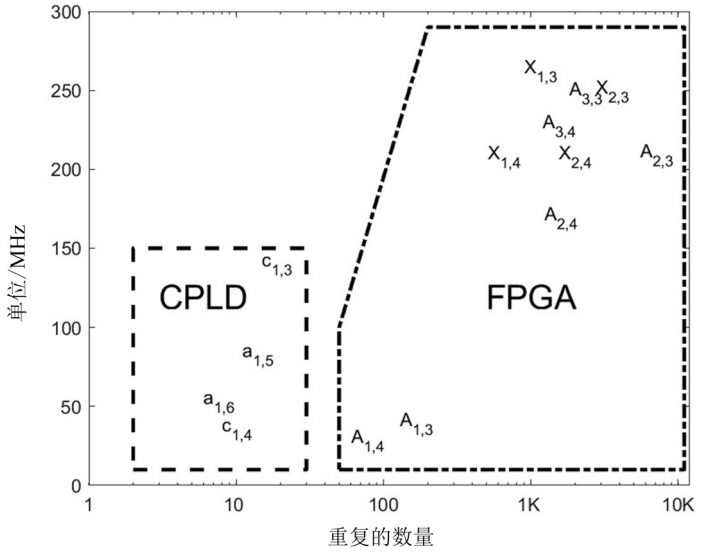


图1.11　Digilent的FPLD提供的PREP基准3和4(即第二个下标)平均值和TERASIC开发板：UP2的A_1 = FLEX10K；DE2的A_2 =Cyclone；DE2-115的A_3 =Cyclone IV；UP2的a_1 = EPM7128；Nexys的X_1 = Spartan 3；Nexys III的X_2 = Spartan 6；Atlys的X_3 = Spartan 6 LX45；以及c_1 = CoolRunner II CPLD

最新的 FPGA 系列和特性

这些特性中的一部分(取决于特定的应用)与嵌入式微处理器的设计更加相关。我们分别在表 1.18 和表 1.19 中总结了 Xilinx 和 Altera 的一些关键特性的可用性。下面列举了器件的系列名称以及其与大多数嵌入式系统应用相关的特性：

1. 器件系列的名称。
2. 到 LUT 的地址输入(扇入)数量。
3. 嵌入式阵列乘法器的大小。
4. 片上块 RAM 的大小，以千(1024)位来衡量。
5. 嵌入式微处理器：当前 Xilinx ZYNQ 和 Altera 器件上的 32 位 ARM Cortex-A9。
6. Xilinx 器件的片上(Virtex 6: 10 位，0.2 MSPS；7 系列: 12 位，1 MSPS)快速 ADC。
7. 器件系列的目标价格和可用性。不再被推荐用于新设计的器件被归为成熟器件，以 m 表示。低成本器件用一个$表示，中等价格器件用两个$$表示，而高价器件用三个$$$表示。
8. 器件系列推出的年份。
9. 工艺技术，以纳米计。

本书编写时，Xilinx 支持 4 个器件系列：性能和容量领先的 Virtex 系列、DSP 密集应用和低成本的 Kintex 系列，以及成本最低用于取代 Spartan 系列的 Artix 系列。此外，还介绍了名为 ZYNQ 的以嵌入式微处理器为中心的系列。包含一个或多个 IBM PowerPC RISC 处理器的 Virtex-II、Virtex-4-FX 或 Virtex-5-FXT 系列不再推荐用于新设计。Xilinx 器件有 18×18 位或 18×25 位嵌

入式乘法器。目前大多数器件提供容量为 18 Kbit 或 36 Kbit 的内存。第六代 Virtex 增加了 0.2 MSPS 10 Kbit 快速片上 ADC。第七代包括一个 12 位 1 MSPS 双通道 ADC，带有额外的电源传感器和片上温度传感器，ADC 支持多达 17 个传感器源，见图 1.12 b。请记住，在开发软件对应的 Web 版本中，仅支持大量的 Spartan 系列；大多数其他器件则需要使用 Xilinx ISE 软件的订阅版本。

表 1.18　最新的 Xilinx FPGA 系列微处理器系统的特性

FPGA 系列	特性							
	LUT 扇入	嵌入式乘法器大小	BRAM 大小/Kbit	快速 A/D	嵌入式 μP	成本/成熟度	年份	工艺/nm
Spartan3	4	18×18	18	无	–	m	2003	90
Virtex 4	4	18×18	36	无	PPC	m	2004	90
Virtex 5	6	18×18	36	无	PPC	m	2006	65
Spartan 6	6	18×18	18	无	–	$	2009	45
Virtex 6	6	18×25	36	有	–	$$	2009	40
Artix 7	6	18×25	36	有	–	$	2010	28
Kintex 7	6	18×25	36	有	–	$$	2010	28
Virtex 7	6	18×25	36	有	–	$$	2010	28
Zynq 7Ks	6	18×25	36	有	1×ARM	$$	2011	28
Zynq 7K	6	18×25	36	有	2×ARM	$$	2011	28
Zynq CG	6	18×25	36	有	2+2ARM	$$$	2015	20
Zynq EV	6	18×25	36	有	2+4ARM	$$$	2015	16

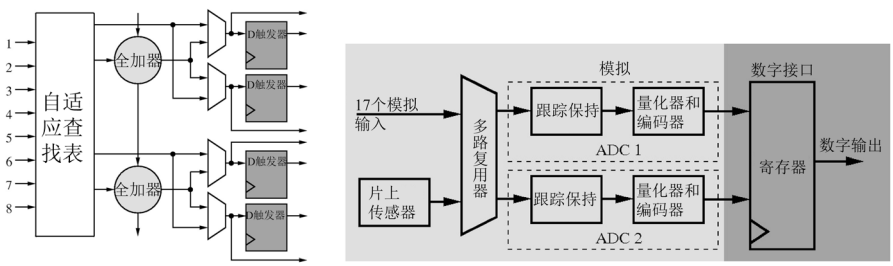
表 1.19　Altera FPGA 系列微处理器系统的特性

FPGA 系列	特性							
	LUT 扇入	嵌入式乘法器大小	BRAM 大小/Kbit	快速 A/D	嵌入式 μP	成本/成熟度	年份	工艺/nm
FLEX10K	4	无	4	无	–	m	1995	420
Cyclone	4	无	4	无	–	$	2002	130
Cyclone II	4	18×18	4	无	–	$	2004	90
Cyclone III	4	18×18	9	无	–	$	2007	65
Cyclone IV	4	18×18	9	无	–	$	2009	60
Cyclone V	8	18×19	0.640，10	无	2×ARM	$	2011	28
Cyclone 10	8	18×19	0.64，20	无	–	$	2017	20
Arria	8	18×18	576	无	–	$	2007	90
Arria II	8	18×18	9	无	–	$	2009	40

(续表)

FPGA 系列	特性							
	LUT 扇入	嵌入式乘法器大小	BRAM 大小/Kbit	快速 A/D	嵌入式 μP	成本/成熟度	年份	工艺/nm
Arria V	8	27×27	10	无	2×ARM	$$	2011	28
Stratix	4	18×18	0.5，4，512	无	–	$$	2002	130
Stratix II	8	18×18	0.5，4，512	无	–	$$	2004	90
Stratix III	8	18×18	9，144	无	–	$$	2006	65
Stratix IV	8	18×18	9，144	无	–	$$	2008	40
Stratix V	8	27×27	20	无	–	$$	2010	28
Stratix 10	8	18×19	4500，20，0.64	无	4×ARM	$$$	2013	14

Altera 提供了 3 种主要的 FPGA 器件：Stratix 高性能器件系列、Arria 中档器件和 Cyclone 器件。其中 Cyclone 器件具有最低成本、最低功率、最低密度和最低性能的特点。如今，器件的逻辑块大小已经从 4 个输入 LUT 增加到最大 8 个不同的输入，例如，允许构建三输入加法器，其速度几乎与双输入加法器相同。在物理结构上，ALM 有两个触发器，两个全加法器，两个四输入 LUT 和四个三输入 LUT，以及许多多路复用器，支持实现一般的六输入功能，见图 1.12。Altera 器件中的嵌入式乘法器大小从 9×9 位、18×18 位到 27×27 位不等。以降低速度为代价将这些块组合在一起，可以构建更大的乘数器。从第五代开始，三个 9×9 位块被组合成一个快速的 27×27 位乘法器。存储器种类繁多，从 0.5K、M4K、M9K、M10K、M144K、到 M512K 位存储器。请记住，只有少数 Cyclone 器件可以在 Web 版本上也就是 Quartus 开发软件的 Prime Lite 版本上运行；Arria 和 Stratix 器件则需要使用订阅版本的软件。



(a) Altera的ALM块　　(b) Xilinx 7系列高速片上ADC

图 1.12　最新 FPGA 系列中使用的新架构特性

与 FPGA 竞争的技术

按厂商划分的 PLD 市场份额如图 1.13 所示。自从 PLD 在 20 世纪 80 年代初问世以来，前 20 年中的营收以每年 20%的数量稳步增长，超过 ASIC 增长数量的 10%以上。2001 年全球微电子衰退降低了 ASIC 和 FPLD 的营收增长速度。自 2003 年以来，我们再次看到两家市场领导者的营收大幅增长(每年增长约 10%)。Actel 于 2010 年 11 月成为 Microsemi Inc.的一部分。Altera 自 2015 年起成为英特尔的一部分。多年来，FPLD 的表现优于 ASIC，其原因似乎与 FPLD 能提

供许多类似 ASIC 具有的优点有关，例如：

- 减小尺寸、重量和功耗
- 更高的吞吐量
- 更好地防止未经授权的复制
- 降低器件和库存成本
- 降低板测试成本

而没有 ASIC 的许多缺点，例如：

- 将开发时间(快速原型)缩短至原来的三分之一到四分之一
- 在线可重编程
- 更低的 NRE 成本，对 1000 台以下微处理器的设计方案更经济

CBIC ASIC 用于高端、高出货量的应用(约超过 1000 份)。与 FPLD 相比，CBIC ASIC 在相同尺寸的裸晶(die)上会有十倍的门数。解决该问题的一种尝试是使用所谓的硬连线 FPGA(Altera 称为 HardCopy ASIC，而 Xilinx 现在将其称为 EasyPath FPGA)，在这种设计中，门阵列用于实现经过验证的 FPGA 设计。

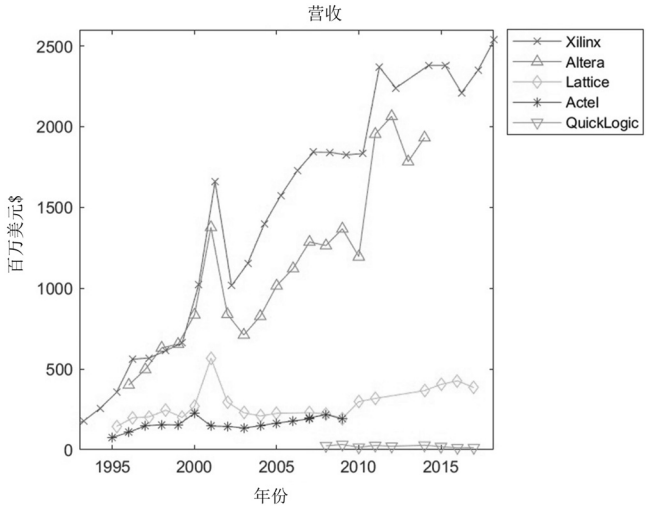


图 1.13　PLD/FPGA/CPLD 市场上营收最高的五家厂商

1.5　使用知识产权核的设计

尽管 FPGA 以其支持快速原型开发的能力而闻名，但这只适用于 HDL 设计已经可用并经过充分测试的情况。像 PCI 总线接口、带流水线的 FFT、FIR 滤波器或微处理器这样的复杂模块可能需要花费几周甚至几个月的开发时间。有一种选择可以让我们从根本上缩短开发时间，那就是使用所谓的知识产权(Intellectual Property，IP)核。这些是预先开发的(较大的)模块，其中典型的标准模块如微处理器、锁相环(Phased Locked Loop，PLL)、FIR 滤波器或 FFT 可直接从 FPGA

供应商处获得，而更专业的块(如 AES、DES 或 JPEG 编解码器、I2C 总线或以太网接口)可从第三方供应商处获得。虽然有些模块在 QUARTUS 软件包中是免费的，但更大更复杂的模块价码可能就高了。然而，只要该块能满足你的设计要求，使用这些预定义的 IP 块往往更合算。现在让我们快速了解一下不同类型的 IP 块，并讨论每种类型具有的优缺点[H00, VG02, CMG07]。通常，IP 核分为三种主要形式，如下所述。

软核 *软核*是一种需要使用 FPGA 供应商工具进行综合的组件行为描述。该模块通常以硬件描述语言(Hardware Description Language，HDL)的形式提供，如 VHDL 或 Verilog，让用户能够轻松修改，甚至可以针对特定供应商或器件在进行综合之前增加或删除功能。缺点在于，IP 块可能需要做更多的工作来满足所需的尺寸/速度/功耗要求。很少有 FPGA 供应商模块以这种形式提供，例如，Altera 公司开发的第一代 Nios 微处理器，或是第 7 和第 8 章讨论的 Xilinx 公司发布的 PICOBLAZE 微处理器。FPGA 供应商要对软核进行 IP 保护是可能的[CMG07]，但很难实现，因为该模块是作为可综合的 HDL 提供的，很容易用于竞争平台的 FPGA 工具/器件套件或基于标准单元的 ASIC。以 HDL 方式提供的第三方 FPGA 模块的价格通常比接下来要讨论的价格较为适中的参数化核的价格高得多。

参数化核 *参数化核*或固核是对组件结构的描述。设计的参数可以在进行综合之前改变，但通常不提供 HDL。Altera 和 Xilinx 提供的大多数核都属于这种类型。它们允许具有一定的灵活性，但会禁止其他 FPGA 供应商或 ASIC 代工厂使用该核，因此为 FPGA 供应商提供了比软核更好的 IP 保护。Altera 和 Xilinx 提供的参数化内核的例子包括 PLL、FIR 滤波器编译器、FFT(并行和串行)，以及嵌入式处理器，例如，Altera 的 Nios II 和 Xilinx 的 MicroBlaze。参数化内核的另一个优点是，它通常提供资源估计(LE、乘法器、块 RAM)，且其期望准确度会在百分之几以内，这使得在进行综合之前就可以在尺寸/速度/功耗要求方面快速探索设计空间。HDL 中的测试平台(用于 MODELSIM 模拟器)允许进行时钟级精确的建模，同时 C/C++模型或 MATLAB 脚本支持行为级精确的仿真，这也是参数化内核具有的标准功能。代码生成通常只需要花费几秒钟。在本章的末尾，我们将研究一个小型的参数化 PLL 核，随后在第 9 章和第 10 章将研究 Altera Nios II 和 Xilinx MicroBlaze 这种大规模参数化内核的例子。

硬核 *硬核*(固定网表核)是 FPGA 内的物理描述或硬接线核。当需要低功耗、高速度或硬性实时约束时，内核通常被优化并可用于特定的器件(系列)，例如，PCI 总线接口。设计的参数是固定的，如 16 位 256 点 FFT。对于一些核的行为，会提供 HDL 描述，让它在一个更大的项目中进行仿真和集成。大多数来自 FPGA 供应商的第三方 IP 核、Xilinx 的几个免费 FFT 核和 ARM Cortex-A9 嵌入式处理器都属于这种类型的核。由于布局是固定的，因此所提供的时序和资源数据是精确的，并且不依赖于综合结果。但缺点是不能改变参数，所以如果 FFT 需要有 24 位输入数据的话，就不能使用 16 位 256 点 FFT 硬核 IP 块。在后面的第 11 章和第 12 章中，我们将研究 Altera 的 SoC Cyclone V 以及 Xilinx Zynq 7K 器件中包含的 ARM cortex A9。

IP 核的比较和挑战

如果我们现在比较不同的 IP 块类型，就必须在设计的灵活性(软核)和快速得到结果及数据

的可靠性(硬核)之间做出选择。软核更灵活，比如，我们容易改动它的系统参数或器件/工艺技术，但可能会需要较长的调试时间。硬核已经在硅片上验证过。硬核减少了开发、测试和调试时间，但没有 VHDL 代码可供查阅。参数化核往往是灵活性和可靠性之间的最佳折中方案。然而，目前的 IP 块技术面临两个主要挑战，即块的定价和与之密切相关的 IP 保护。由于核是可重复使用的，因此供应商的定价必须依赖于客户将使用的 IP 块的单元数量。这是一个多年来在专利权方面众所周知的问题，通常需要签订长期的许可协议，并且在客户滥用的情况下需要支付高额罚款。FPGA 供应商提供的参数化块(以及设计工具)有非常适度的定价，因为供应商的获利方式是：首先客户在许多器件中使用该 IP 块，并且这些器件也往往必须从同一供应商处购买。这与第三方 IP 块的供应商不同，他们没有第二种收入来源。这种情况下，许可协议，特别是软核的许可协议，在起草时必须非常仔细。

为了保护参数化内核，FPGA 供应商使用基于 FlexLM 的密钥来启用/禁用单个 IP 内核的生成。通过使用有时间限制的编程文件或要求通过 JTAG 在上位 PC 和电路板之间保持连接，可以支持对参数化内核进行直至硬件验证的评估，让设计者在购买许可证之前就可以对器件进行编程并在硬件上验证设计。例如，Altera 的 OPENCORE 评估功能允许设计者在目标系统内模拟 IP 核功能的行为，以验证设计的功能，并快速、轻松地评估其大小和速度。当你对 IP 核功能完全满意并希望将设计投入生产时，可以购买一个许可证，从而允许生成不受时间限制的编程文件。QUARTUS 软件会自动从 Altera 的网站下载最新的 IP 核。许多第三方 IP 供应商也支持 OPENCORE 评估流程，但你必须直接与 IP 供应商联系，以启用 OPENCORE 功能。

对软核的保护更为困难。有人建议对 HDL 进行修改，使其难以读取，也就是使用混淆技术[MCB11]，或者在最小化额外硬件的同时将水印嵌入高层设计中[CMG07]。水印应该足够健壮，也就是说，水印中如果有一个位发生变化就足以破坏所有者的认证。

表 1.20 显示了对三种类型的 IP 核最重要特性的比较。

表 1.20　IP 核比较

特性	IP 核类型		
	软核	参数化核	硬核
灵活性	++	++	-
性能	+	+	++
IP 成本	-	+	+
开发时间	-	+	++
不依赖供应商	++	-	-
电源效率	+	+	++
例子	PICOBLAZE、Nios	FIR 编译器、FFT 编译器、Nios II、MICROBLAZE	PCI、IEEE FP、ARM Cortex-A9

图例：++=优秀，+=良，-=差

基于 IP 核的 PLL 设计实例

最后，我们以一种典型的开发板设计为例，评估 IP 块生成并将其集成到项目中的设计过程。开发板有一个高精度的固定在 50 MHz 频率的振荡器；而我们所用的嵌入式微处理器系统可能需要用到一个高达 2.5 倍的时钟，即 125 MHz。这样升高了的时钟基本上不可能用同步数字逻辑产生。于是，大多数现代 FPGA 使用锁相环(Phase Locked Loop，PLL)来产生不同的时钟或相位延迟。图 1.14 展示了 FPGA 中使用的典型 PLL 的整体结构。PLL 是一种混合的模拟/数字电路，其核心是一个相位/频率检测器(Phase Frequency Detector，PFD)、一个压控振荡器(Voltage Controlled Oscillator，VCO)和一个反馈计数器。在稳定状态下，我们会注意到两个 PFD 信号(F_{REF} 和 F_{FB})是同相的(完全一样)。环路是稳定的也就是说已锁定，即反馈频率 $F_{FB}=F_{VCO}/M$ 与输入频率为 $F_{REF}=F_{IN}/N$ 的参考信号同相，假定参考频率来自输入分频器 N。重新排列使输出频率 F_{VCO} 在左边，我们选择 M=5 和 N=2，期望的输出频率为:

$$F_{VCO} = M \times F_{IN}/N = 5 \times 50\ \text{MHz} / 2 = 125\ \text{MHz} \tag{1.6}$$

额外的后缩放分频器 C 可以用来产生基于 F_{VCO} 的额外分频，或产生不同的相位延迟。

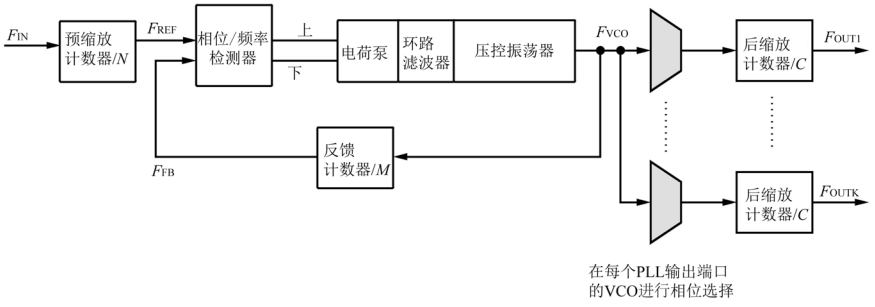
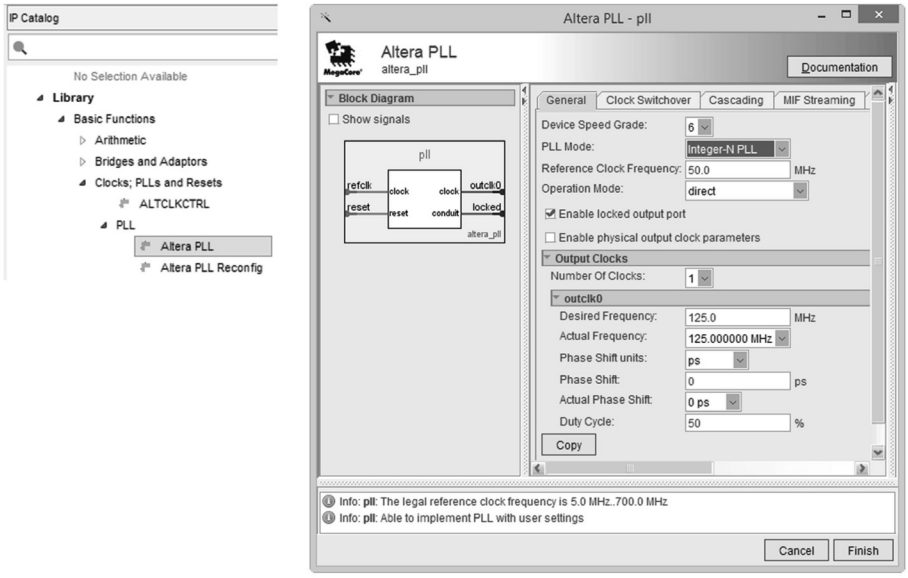


图 1.14　FPGA PLL 的架构

你可以在 Altera Quartus IP 目录中找到各种 PLL 架构，包括零输出延迟和直接选项。为了进行简单的评估，我们把 IP 核命名为我们开发的项目，即 `PLL`。在 QUARTUS 中建立一个新的文件目录并打开一个新的项目，命名为 `PLL` 并选择 `Tools→IP Catalog`。第一步，在 IP Catalog 的 `Basic Functions→Clocks; PLLs and Resets→PLL→Altera PLL` 窗口下找到 `PLL` 块，见图 1.15a。然后选择所需的输出格式(AHDL、VHDL 或 Verilog)，并指定我们的工作目录。在 IP GUI 工具弹出后(见图 1.15b)，我们就可以访问 PLL 文档并可以开始对该模块进行参数化。可以将 PLL 配置成专门的模式，如零延迟(参考输入和输出之间只需要相位延迟)，不过我们使用了基本的配置，称为 `direct`，即直接操作模式。由于我们想用上面例子中的 50 MHz 参考信号产生 125 MHz 的输出信号，因此在 GUI 中分别输入 `Desired Frequency` 期望频率和 `Reference Clock Frequency` 参考时钟频率这两个数据。我们保持 50%的占空比，不需要任何相移。信息窗口会立即给我们提供反馈，告诉我们选择的频率是否可以通过 FPGA PLL 实现。由于 PLL 是一个混合的模拟/数字电路，FPGA 通常在 VCO 信号的上下界会进行限制，这点可以从 PLL 参数窗口的 `Info` 输出中看到。我们选择的 125 MHz 当然是在 5~700MHz 的允

许范围内的。



(a) IP Catalog元素选择　　(b) 按照对参考时钟输入进行2.5倍频对PLL IP核参数化

图 1.15　PLL IP 设计

在对选择的参数感到满意后，我们单击 Finish 完成设计，然后模块和所有支持文件就会立即生成。除了设计文件，还提供了 ALDEC、CADENCE、MENTORGRAPHICS 和 SYNOPSYS 的仿真脚本。表 1.21 中列出了生成的一些最重要的文件的概况。

表 1.21　为 IP PLL 核生成的最重要的文件列表

pll.vhd	定制 IP 核功能的 VHDL 顶层描述
pll.cmp	IP 核功能变种的 VHDL 组件声明
pll.bsf	IP 核功能变种的 QUARTUS 符号文件
pll_sim/memtor/msim_setup.tcl	用于为 MODELSIM 编译库模型和 IP 组件的测试平台
pll.vho	VHDL IP 功能性仿真模型
pll.qip	QUARTUS 项目信息文件

我们看到，VHDL 文件与其对应的组件文件是一起生成的，既支持组件的图形实例化，又支持 HDL 实例化。VHDL 组件文件 pll.cmp 如下所示：

VHDL 文件：PLL IP 组件

```
1 component pll is
2           port(
3                refclk    : in std_logic := 'X'; -- clk
```

```
4                rst       : in std_logic := 'X'; -- reset
5                outclk_0  : out std_logic;       -- clk
6                locked    : out std_logic        -- export
7            );
8      end component pll;
```

我们决定直接使用这个 IP 核作为顶层设计输入，因此避免了在另一个设计中实例化该模块并连接输入和输出。通过检查顶层 VHDL 文件 `pll.vhd` 或组件文件，我们注意到，该 ENTITY 具有预期的模块输入 `refclk` 和输出 `outclk_0` 信号，但有一些额外的有用的控制信号，即 `rst` 和 `locked`，这两个信号的功能无需解释。然后我们启动 QUARTUS 运行全编译，以生成用于时序仿真的 `pll.vho` 文件，该文件可以在`/simulation/modelsim` 中找到，我们用这个生成的文件来替换目录 `pll_sim` 中的功能文件。

为了对设计进行仿真，我们使用生成的 TCL 脚本来编译设计和模型库。启动 MODELSIM 仿真器并切换到目录 `pll/pll_sim/mentor`，然后在命令窗口输入 `do msim_setup.tcl`。启动脚本后，它显示了一个有可能执行的步骤对应的菜单。我们选择编译(compilation)，然后选择详述(elaboration)。几个库和设计文件会被编译，但不会进行仿真。要进行仿真，我们使用以下步骤：

ModelSim：命令行提示输入序列

```
1 type> com
2 type> elab
3 type> add wave *
4 type> force rst 1 0ns, 0 50ns
5 type> force refclk 0 0ns,1 10ns -r 20ns
6 type> run 500ns
```

我们放大到前几个时钟周期，看到 PLL 锁定(大约两个时钟周期后)和高时钟信号，如图 1.16 所示。对于 `refclk` 输入的两个时钟周期，我们有 5 个输出时钟周期，即实现了期望的 2.5 倍高频。我们可能会注意到 IP 块仿真中存在的一个小问题，这个问题无法在 GUI 设置中得到解决。通常，FPGA 板使用低电平有效来进行复位，因为大多数按钮在按下后会归零。在软核中，我们可以改变设计的 HDL 代码来实现低电平有效复位，但在参数化的核中，我们没有这个选项。然而，可以通过在 PLL 复位的输入端附加一个反相器来解决这个问题。这是使用参数化内核的典型经验：内核可以节省 90%或更多的设计时间，但有时为了满足项目的严格要求，需要做一点额外的设计工作。

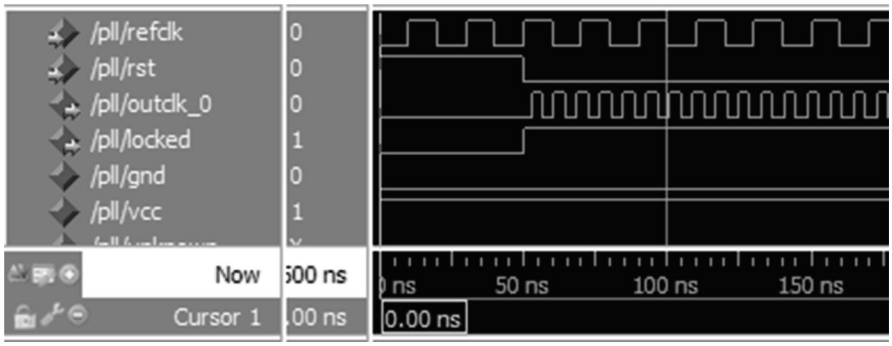


图 1.16　PLL IP 设计的测试台。通过时序仿真进行验证

1.6　复习题和练习

简答题

1.1. 为什么 Plessey FPLD 器件没有取得商业成功？
1.2. 我们应该使用奔腾处理器来设计嵌入式系统吗？
1.3. 在微处理器指令解码过程中会发生什么？
1.4. 列举并解释五种 μP 寻址模式。
1.5. 画出 0~3 地址的 CPU 具有的数据流结构。
1.6. 将十进制数字 123、1000、255、4681 分别转换成二进制、八进制和十六进制。
1.7. 十六进制数 A、123、ABCD、FFF 的十进制表示是什么？
1.8. 对于存储器来说，1 KB(Kilo Byte)、1 MB 或 1 GB 的精确位数是多少？
1.9. 假定每个像素需要 16 比特，存储一幅 640 行、480 列的图像需要多少比特？
1.10. 解释冯诺依曼、哈佛和超级哈佛架构之间的区别。
1.11. 根据以下特性对 0~3 地址的 CPU 进行排序：指令长度、汇编的难易程度、代码字数、硬件大小(最好的排在前面)。
1.12. 你什么时候会选择使用 FPGA，什么时候会选择使用基于单元的 ASIC 进行设计？
1.13. FPLD 基准测试是如何工作的？
1.14. 什么是 FPLD 的粒度？
1.15. 按电路速度对以下技术排序：基于单元的 ASIC、FPGA、完全定制和 PLD(速度最慢的器件排在前)。
1.16. 哪些 FPGA 硬件资源对 μP 设计最重要？(从最重要的开始列举)。
1.17. PLL 是如何工作的？什么时候需要用到 PLL？

填空题

1.18. Cyclone V 器件是_____的产品。
1.19. Zynq 7K 器件是_____的产品。
1.20. 希望的 FPLD 技术特性是_____。
1.21. 不想要的 FPLD 技术特性是_____。
1.22. ASIC 是_____的首字母缩写。
1.23. CPU 是_____的首字母缩写。
1.24. RISC 是_____的首字母缩写。
1.25. KCSPM 是_____的首字母缩写。
1.26. NRE 是_____的首字母缩写。
1.27. PLA 是_____的首字母缩写。

判断题

1.28. _____ EPROM FPLD 提供“复制保护”，但 E^2PROM FPLD 不提供。
1.29. _____ SRAM FPLD 是可重复编程和易失性的。
1.30. _____ FPGA 的功耗与频率和在此频率下运行的逻辑单元数量的乘积成正比。
1.31. _____ PREP 基准测试 3 和 4 是 FSM 类型的设计。
1.32. _____在 PREP 基准中，CPLD 的重复率比 FPGA 高。
1.33. _____ Cyclone V 器件有嵌入式乘法器、BRAM 和一个 PPC 微处理器。
1.34. _____ Zynq 器件包括一个 ARM 微处理器。
1.35. _____在过去的十年中，FPGA 市场营收方面的领导者是 Lattice 和 Actel 公司。
1.36. _____在过去的十年中，FPGA 的营收每年增长 20%，而 ASIC 每年增长 10%。
1.37. _____ 现代 FPGA 中使用的 BRAM 存储器是异步存储器。
1.38. _____ 现代 FPGA 通常有 PLL。

项目和挑战

1.39. 只用 NAND 门来实现 NOT、AND 和 OR，以证实双输入 NAND 的通用性。
1.40. 只用双输入 NAND 门来实现一个全加器：
s = a XOR b XOR c_{in}；c_{out} =(a AND b) OR(c_{in} AND(a OR b))
1.41. 只用 NOR 门来实现 NOT、AND 和 OR，以证实双输入 NOR 门的通用性。
1.42. 只用双输入 NOR 门来实现一个全加器：
s = a XOR b XOR c_{in}；c_{out} =(a AND b) OR(c_{in} AND(a OR b))
1.43. 只用双输入复用器来实现 NOT、AND、OR，以证实双输入复用器 f(x,y,s)= x AND(NOT s)OR(y AND s)的通用性。

1.44. 只用双输入复用器 f(x,y,s)= x AND(NOT s)OR(y AND s)来实现一个全加器：

s = a XOR b XOR c_{in}；c_{out} =(a AND b) OR(c_{in} AND(a OR b))。

1.45. 将下面的中缀算术表达式转换为后缀表达式，见式(1.2)。

(a) a + b−c−d + e

(b) a + b × c

(c) (a−b) ×(c + d) + e

(d) (a−b) ×((c−d × e) × f)/g)× h

1.46. 将下列后缀算术表达式转换为中缀表达式，见式(1.2)。

(a) ab + cd /

(b) ab/cd ×−

(c) ab + c ^(用^幂符号)

1.47. 以下哪一对后缀表达式是等价的(见式(1.2))?

(a) ab + c +和 abc + +

(b) ab −c −和 abc−−

(c) ab×c +和 cab× +

(d) abc +×和 ab×bc×+

1.48. 通过编写尽可能最短的程序进行以下计算来比较 0~3 地址编码(Address Coding，AC)机器：

$$h=(a-b)/((c+d)\times(f-g))$$

指令集如下。

(a) 0-AC(栈)：PUSH Op1，POP Op1，ADD，SUB，MUL，DIV

(b) 1-AC(累加器)：LA Op1，STA Op1，ADD Op1，SUB Op1，MUL Op1，DIV Op1

(c) 2-AC：LD Op1, M；ST Op1, M；ADD Op1, Op2；SUB Op1, Op2；MUL Op1, Op2；DIV Op1, Op2

(d) 3-AC：LD Op1, M；ST Op1, M；ADD Op1, Op2, Op3；SUB Op1, Op2, Op3；MUL Op1, Op2, Op3；DIV Op1, Op2, Op3

假设 2-AC 和 3-AC 的所有操作数都已经加载到寄存器中，但最后的结果应该存储在存储器中。另外，对于 2-AC 和 3-AC，寄存器的值不需要保留，也就是说，你可以覆盖寄存器的初始值！

1.49. 对下面的算术表达式重复练习 1.48。

$$f=(a-b)/(c+d\times e)$$

如果有必要，你可以重新排列这个表达式。

1.50. 对下面的算术表达式重复练习 1.48。

$$f=(a-b/c)/(b+d\times e)$$

如果有必要，你可以重新排列该表达式。

1.51. 对下面的算术表达式重复练习 1.48。

$$f=(a+b\times c)\ /(d-e)$$

如果有必要，你可以重新排列该表达式。

1.52. 对下面的算术表达式重复练习 1.48。

$$g=(a\times b-c)\ /(d+e/f)$$

如果有必要，你可以重新排列该表达式

1.53. 对以下算术表达式重复练习 1.48。

$$g=(a-b\times c)\ /(d/e+f)$$

如果有必要，你可以重新排列这个表达式。

1.54. 访问 FPLD 供应商的网页，找到 Xilinx 和 Altera/Intel FPGA 的最新营收。

1.55. 解释一下软核、硬核和参数化核之间的区别。每种类型的优点和缺点是什么？

1.56. 检查你所用工具库中的 IP 核，用一句话总结其功能。这些核的成本是多少？在你的列表中，哪一个 IP 核可以用在 μP 设计中？

第 2 章

FPGA 器件、板卡和设计工具

摘要

本章详细介绍了对嵌入式系统设计特别有帮助的 SoC 型 FPGA、原型电路板卡以及设计工具。嵌入式软件设计师在首次阅读本书时也可以先跳过本章。

我们将仔细研究两个特殊的器件，Altera EP5CSEMA5F31C6 和 Xilinx Zynq-Z-7010，它们将用于我们提出的更大的设计示例。我们将研究芯片综合、时序分析、布图(floorplan)和功耗。最后，在一个更大的案例研究中，终极精简指令集计算机(Ultimate Reduced Instruction Set Computer，URISC)是用 HDL 语言设计、综合和编程的。

关键词

现场可编程门阵列(Field Programmable Gate Array，FPGA) • 现场可编程逻辑器件(Field Programmable Logic Device，FPLD) • Quartus • Vivado • Altera FPGA • Xilinx FPGA • TerASIC • 视频图形阵列(Video Graphics Array，VGA) • 高清多媒体接口(High Definition Multimedia Interface，HDMI) • 布图 • URISC • ZyBo • DE1-SoC • TimeQuest • I2C 总线 • 通用异步收发器(Universal Asynchronous Receiver Transmitter，UART) • 7 段显示 • 模数转换器(Analog to Digital Converter，ADC) • 音频编解码 • CAD 设计循环

2.1 引言

VLSI 设计中用到的技术细节涵盖了从全定制 ASIC 的几何布局到使用所谓的机顶盒进行的系统设计。表 2.1 给出了一项调查。而 FPGA 设计工作中并不包含布局和电路级的任务，因为它们的物理结构是可编程的，但是固定的。最优的器件利用通常是在门的层次上使用寄存器传输级设计语言。上市时间要求结合迅速增加的 FPGA 复杂性，正迫使方法论转向使用知识产权(Intellectual Property，IP)宏单元(macrocell)或巨核单元(mega-core cell)。宏单元为设计者提供了一系列预定义功能，如微处理器或 UART。因此，设计师只需要指定选定的特性和属性(例如，精度)，综合器将为目标解决方案生成硬件描述代码或原理图。

表 2.1　VLSI 设计级别

对象	目标	实例
系统	性能规格	计算机、磁盘单元、雷达
芯片	算法	μP、RAM、ROM、UART、并口
寄存器	数据流	寄存器、算术逻辑单元、计数器、多路复用器
门	布尔方程	与门、或门、异或门、触发器
电路	差分方程	三极管、电阻、电感、电容
布局	无	几何形状

因此，FPGA 技术的一个关键点在于它是强大的设计工具，具有以下作用：

- 缩短设计周期。
- 提供良好的器件利用率。
- 提供综合器选项，即在速度与设计尺寸的优化之间进行选择。

应用于嵌入式系统 FPGA 设计流程的 CAD 工具分类如图 2.1 所示。系统需求和层次组织应该在进入 FPGA 设计流程之前就有据可查。

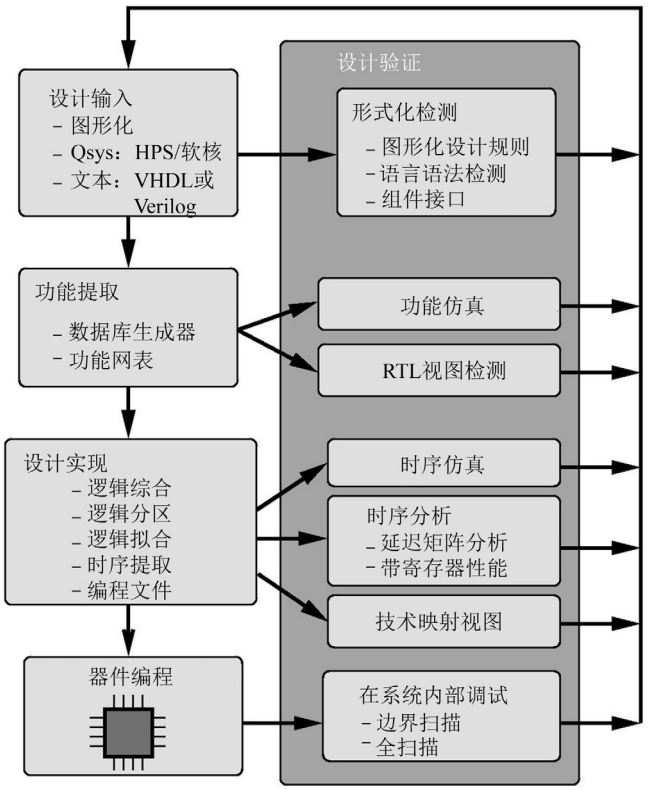


图 2.1　基于嵌入式 μP FPGA 的 CAD 设计流程

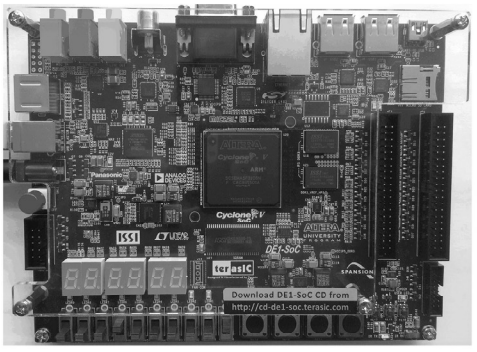
设计输入(Design Entry)可以是图形化的、基于文本的，或是基于软核和 HDL 组件构建的高层次系统工具。进行后续步骤之前，应该执行形式化检测以消除语法错误或图形设计规则错误(例如，连线开路)。预定义块的分配，包括通过 QSYS[1](Altera) 实现的嵌入式硬连接处理器系统(Hardwired Processor System，HPS)，也会在设计的第一步完成。多种多样的微处理器测试程序也应该与 μP 设计并行开发(图中未展示)。在功能提取(Function Extraction)中，会从设计中提取基本设计信息，并将其写入功能网表(Functional netlist)中。该网表支持对电路进行第一次功能仿真(Functional Simulation)(又名 RTL 级仿真)，并允许构造一个称为测试台的示例数据集，用于在以后使用时序信息来测试设计。功能网表还支持电路的 RTL 视图，从而可快速总览 HDL 所描述的电路。如果 RTL 视图验证或功能仿真没有通过，我们将再次从设计输入开始进行设计。如果功能测试满足要求，我们将继续进行设计实现(Design Implementation)，这通常需要几个步骤才能完成。如果组件是软核或参数化核而不是硬核，则需要花费比功能提取更多的编译时间。在设计实现的最后，微处理器电路在 FPGA 内完全布线，这提供了精确的资源数据，从而支持使用所有时序信息来进行仿真(又称门级仿真)，并进行性能测量。一些综合工具还提供电路的技术映射视图，用于展示 HDL 元素如何映射到 LUT、存储器和嵌入式乘法器。如果所有这些实现数据都符合预期，我们就可以继续进行实际 FPGA 的编程；如果不符合，我们就得从设计输入重新开始，并在设计中进行适当的更改。使用现代 FPGA 的 JTAG 接口，我们也可以直接监控 FPGA 上的数据处理：我们可以只读出 I/O 单元(称为边界扫描(boundary scan))，或者读回所有内部触发器(称为全扫描(full scan))。现代软件工具(例如，监控程序)也支持单步跟踪程序，同时检查存储器和寄存器值。如果在系统内部调试失败，我们也需要返回到设计输入。

一般来说，若要做出使用固定的、参数化的还是软的微处理器内核的决定，取决于个人品位、先前的经验和系统要求。系统的图形化表示可以强调基于组件来构建层次结构，从而支持模块化设计方法和测试。如果设计不需要使用很多关于算法控制设计的大型组件，并且允许更广泛的设计风格，则通常首选 HDL 文本环境，如后面的案例研究所演示的那样。具体来说，对于基于 HDL 的设计，通过文本设计，似乎可以赋予设计更特殊的属性和更精确的行为。在基于文本的 HDL 设计中，如果文本是 VHDL 或 Verilog，我们可以使用以下三种设计策略中的任何一种来进行设计。

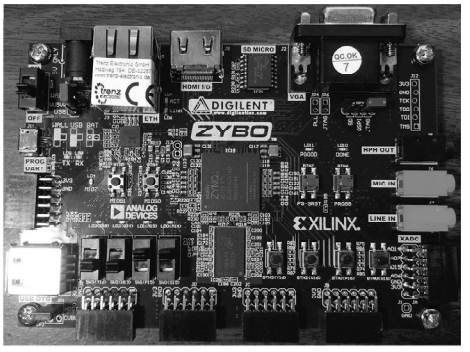
结构化风格(通过组件实例化)非常类似于图形化的网表设计，并且只能用于大型(预定义的或重用的)组件，而不能用于门级，因为这会使代码非常难以阅读、验证和维护。数据流风格(即并发语句)在用到大量的门或多路复用类型电路，而编码中没有任何存储器元素时使用。当与门或多路复用器一起设计时，综合工具通常不识别锁存器或触发器，并且将使用标准 LUT，而不是 FPGA 内提供的物理触发器或锁存器。当需要用到存储器元素时，可使用 `PROCESS` 模板(VHDL)或 `always` 块(Verilog)进行顺序化设计。当然，这些策略也可以进行组合使用，只是要注意，在这些块中求值是顺序执行的，而不是像代码的数据流部分那样是并发执行的。关于 HDL 语言的详细信息可以在第 3 章(VHDL)和第 4 章(Verilog)中找到。

1 Intel 最近已将 QSYS 更名为 PLATFORM DESIGNER。

所有 HDL 代码都从定义 I/O 端口开始，随后是内部网络。在 VHDL 中，需要在端口声明之前指定所使用的库。VHDL 具有比 Verilog 更大、更复杂的库和数据类型概念。然后才是实际的电路描述。可以采用一种编码风格，也可以采用上面讨论的三种风格。关于如何综合和仿真电路(根据图 2.1 中所示的流程)的详细研究将在 2.4 节中针对 URISC 处理器模型来进行介绍。在 CAD 工具流程的末尾，我们将得到一个编程文件，可以将其下载到硬件板卡(如图 2.2 所示的原型板)。接下来，我们继续对设备进行编程，并可能使用回读方法执行额外的硬件测试。



(a) Cyclone V DE1-SoC Altera/TerASIC 板卡[Ter14]



(b) Xilinx Zynq 板卡(ZyBo)[Dig14]，带有板载的 ADC 及 DAC 编解码器

图 2.2 低成本原型板卡

2.2 原型板卡的选择

如果我们想选择一个支持使用软的、参数化的和硬核处理器系统的适当平台，那么可能需要用到的原型板数量就会减少到嵌入了 HPS 的少数几个。我们还需要提供足够的资源(LE、嵌入式乘法器、块 RAM、锁相环和引脚数量)来承载我们计划的最大设计任务。另一方面，我们可能希望选择使用 Altera 或 Xilinx 大学计划提供的(低成本)电路板。

Xilinx 提供的直接板级支持非常有限，举例来说，大学计划中提供的所有电路板都来自第三方。然而，其中一些板的价格如此之低，似乎是非营利的设计。Xilinx 板的主要供应商是 Digilent Inc.，在撰写本书时，它提供 ZedBoard、ZyBo、Nexys 4 DDR 或 Basys 3。即使对非大学的顾客来说，这些板也很便宜。另一个目标可能是使用 VIVADO 网络版软件支持的电路板。目前，VIVADO 网络版支持 Zynq 评估和开发板(ZedBoard)，还支持 Zynq 板(ZyBo)。Nexys 和 Basys 需要用到 ISE 工具。表 2.2 给出了一些流行的 Xilinx 板的概述。Digilent Inc.提供的 Zynq 板(ZyBo)是一款适用于嵌入式系统的低成本板，具有 HPS(带片上 BRAM、乘法器、音频编解码器、VGA、开关、按钮和 LED)，价格仅为 189~299 美元，见图 2.2b。ZyBo 上有一颗 Zynq XC7Z010-1CLG400C FPGA、128Mb 闪存、512MB DDR、5 个 LED、4 个开关和 6 个按钮[Xil18, Dig14]。对于 DSP 和视频实验，我们可以利用片上的 A/D、音频编解码器和 VGA 端口。

表 2.2　流行的 Xilinx 板卡、FPGA 及板上元器件的概览

板卡名称	ZedBoard	ZyBo[1]/ZyBo-Z7-10	ZyBo-Z7-20
FPGA 设备	Zynq-7K SoC XC7Z020-CLG484	Zynq-7K XC7Z010-1CLG400C	Zynq-7K XC7Z020-1CLG400C
I/O/LUT/片上 RAM	200/53 K/612 KB	100/17 K/270 KB	100/53 K/630 KB
微处理器	2x ARM Cortex-A9	2x ARM Cortex-A9	2x ARM Cortex-A9
开关/按钮/LED	8/5/8	4/6/5	4/6/5
片外闪存/DRAM	256 MB/512 MB	–/512 MB	–/1024 MB
显示选项	OLED-VGA-HDMI	VGA-HDMI-0/1RGB LED	HDMI-Pcam
A/D 选项	编解码器和 XADC	编解码器和 6/5Pmod	编解码器和 6Pmod
价格	$495	$189/$199	$299

1　最近，ZyBo 已被 ZyBo Z7 板卡取代。主要的升级在于使用了一个额外的(单向) HDMI 口代替 VGA，以及第二版带有更大 FPGA 的 ZyBo-Z7-20[Dig19]。

Altera 支持多种开发板，其中包含大量有用的原型组件，包括快速 A/D、D/A、音频编解码器、DIP 开关、单段和七段 LED 以及按钮(见表 2.3)。这些开发板可直接从 Altera(大学客户)或 TerASIC(标准客户)获得。Altera 提供 Stratix 和 Cyclone 板，价格在 199 美元至 24,995 美元之间，这些板不仅在 FPGA 尺寸上有所不同，而且在额外的特性方面也有所不同，比如 A/D 通道的数量、精度和速度，以及内存块。对于大学来说，低成本的 DE1-SoC Cyclone V 板是一个很好的选择，它虽然比许多数字逻辑实验室中使用的 UP2 或 UP3 板更贵，但具有双通道编解码器、FPGA 之外的大型内存块，以及许多其他有用的端口(USB、VGA、PS/2、以太网、七段 LED、开关、按钮等)，见图 2.2a。

表 2.3　流行的 Altera 板卡、FPGA 及板上元器件

板卡名称	DE2-115	DE10 标准	DE1 SoC
FPGA 设备	Cyclone IV EP4CE115F29C7	Cyclone V SoC 5CSXFC6D6F31C6N	Cyclone V SoC 5CSEMA5F31
I/O/LUT/片上 RAM	529/115 K/497 KB	499/111 K/707 kB	457/85 K/487 KB
微处理器	无 HPS	2x ARM Cortex-A9	2x ARM Cortex-A9
开关/按钮/LED	18/4/27	10/4/10	10/4/10
片外闪存/DRAM	2 MB/128 MB	–/64 MB+1 GB	–/64 MB+1 GB
显示选项	8×7 段/LCD/VGA/TV 解码器	6×7 段/LCD/VGA/TV 解码器	6×7 段/VGA/TV 解码器
A/D 选项	24 比特双通道编解码器	8×12 比特 500KSPS	24 比特双通道编解码器 8×12 比特 500KSPS
价格	$595	$350	$249

现在，我们简要介绍一下目前在电路板上已有并将在以后的项目中使用的常见外围设备。

2.2.1 存储器

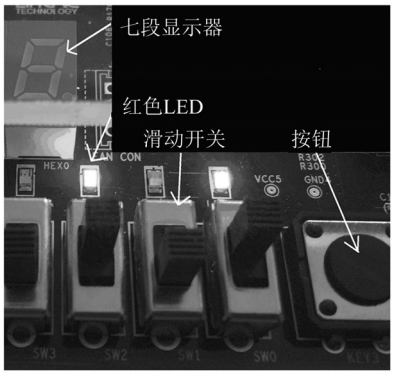
FPGA 嵌入式系统原型板通常在 FPGA 之外有几种不同类型的存储器。让我们先从 FPGA 配置存储器说起。由于基于 SRAM 的 FPGA 不稳定，它们需要在开始时一次性将配置文件下载到设备上，其中包含 FPGA 电路编程信息以及初始片上寄存器和块 RAM 值。在开发过程中，我们通过串行接口(如 USB)下载这些文件。通常，这种通信是使用 JTAG 协议完成的，这样就允许对其进行部分重新配置和回读。开发之后，我们可能会想在不用主机的情况下对板卡进行配置，板卡也常常提供某种 E^2PROM 闪存来存储 FPGA 的编程信息。这些闪存配置 ROM 仅用于编程，通常不能作为整个微处理器系统设计的一部分来加载或存储数据。

第二种类型的存储器是 SRAM 或 DRAM 存储库(bank)，大小从几 MB 到上 GB，远远大于片上的块 RAM 资源。SRAM 通常具有较短的访问时间，而 DRAM 通常提供较大的内存。如果我们使用 RAM 作为微处理器存储器，则需要注意存储库的组织。除了 FPGA 中的块 RAM，这些外部存储库在配置中是固定的。例如，如果我们准备构建一个 32 位微处理器系统，但有一个 16 位存储器接口，就需要使用两个时钟周期来加载一个字。快速的 16 位宽 SRAM 可能会比“较慢”的 32 位宽 DRAM 慢，因为 DRAM 数据只需一个时钟周期就可用。

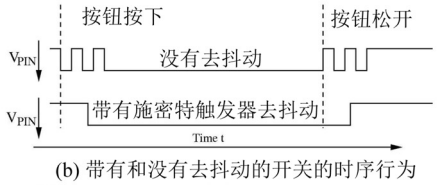
嵌入式系统板上的第三种外部 FPGA 存储器通常通过 micro SD 卡插槽提供。虽然访问速度常比 S(D) RAM 慢，但存储容量只受购买的 SD 卡大小的限制(通常不提供)。这将允许你编译和配置标准 PC 的操作系统或复制大量数据(图像、视频等)到 SD 卡上，然后使用 FPGA 访问这些 micro SD 上存储的文件。

2.2.2 基本 I/O 组件

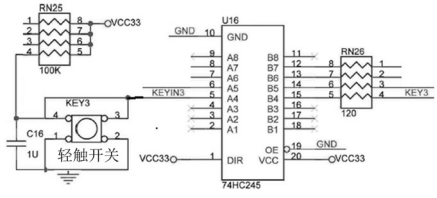
FPGA 板卡上通常有 4 种基本的 I/O 元素：LED、七段显示器、按钮和开关，见图 2.3a。



(a) LED、按钮、开关和七段显示器



(b) 带有和没有去抖动的开关的时序行为



(c) DE1 SoC去抖动逻辑的电路细节

图 2.3 基本 I/O

所有电路板都会带有几个红色、绿色或双色的单个 LED。当设置为逻辑 1 时，它们会亮起；当设置为逻辑 0 时，它们会关闭。空间和 I/O 引脚资源较富余的 FPGA 板卡可以增加一个或多个七段显示器。不同于单个 LED，七段显示器通常是低电平有效，即当输入设为 0 时亮起，因为 FPGA 或 CMOS 通常可以在 GND 电平处比 Vcc 电平抽取更大电流。由于一块板上的多个七段显示器需要用到大量的 FPGA 引脚，我们有时会看到(例如，Digilent Nexys)多个七段显示器共享数据线，并使用一个选择信号在多路复用模式下亮起。小型 ZyBo 没有七段显示器，而 DE1 SoC 有 6 个非复用的数字显示，其中小数点未使用。

FPGA 的基本输入外设还包括开关和按钮。开关“南”通常用作输入逻辑 0，而“北”输入逻辑 1 到 FPGA。ZyBo 有 4 个滑块开关，而 DE1 SoC 有 10 个。这些按钮使我们能够产生比滑动开关更短的 FPGA 输入使能信号。在使用开关或按钮时，状态切换的 2~3 ms 内倾向于在 0 和 1 之间切换，如图 2.3b 所示的开关抖动。因此，在某些板卡上，这些按钮是“去抖动”(debounced)的，也就是说，当按钮被按下或释放时，产生的是个可靠的上升或下降边缘过渡。可以使用外部组件或内部 FPGA 资源以不同的方式实现去抖动。我们可以使用具有迟滞的线路驱动器，即所谓的施密特触发器，如 IC 74HC245。0→1 会比 1→0 转换具有更高的阈值。此外，DE 板卡使用一个电阻(100 KΩ)和一个较大的电容(1 μF)。电容器并联于按钮，以确保具有更慢更稳定的行为。按钮释放后，电容器以 RC 时间常数充电

$$U_C = \left(1 - e^{-t/(RC)}\right) Vcc \tag{2.1}$$

当 $0.5=(1-e^{-t/(RC)})$时，电容电量将达到 50%，对于我们所用的 R=100KΩ 和 C = 1μF, 切换按钮之间的最小时间为 $T=R\,C\,\ln(2)\approx 70$ ms，这绝对是安全的。1→0 会短得多，因为这只受限于通过按钮进行的电容放电。从图 2.3c 中我们也可以看到，DE1 上的按钮是低电平有效，即按下时将输入逻辑 0，而 ZyBo 上的按钮按下时其逻辑为 1。ZyBo 按钮，或者 ZyBo 或 DE 板上的开关没有去抖动，因此不应在计数单个事件时使用，例如，在单步执行微处理器程序时。如果需要，我们应该添加一个 FPGA 去抖动电路，每 10ms 左右对输入进行“采样”(使用触发器和带有输出使能的计数器)，这样抖动不会导致在后续模块中进行多次检测。

2.2.3　显示选项

FPGA 开发板(见图 2.3a)通常带有一个或多个基本显示选项，如 LCD、LED 和七段显示器。此外，板卡通常支持图形输出，如 VGA 或 HDMI 的图像和视频。例如，Xilinx ZyBo 配备了视频图形阵列(Video Graphics Array，VGA)接口和用于高清多媒体接口(High Definition Multimedia Interface，HDMI)的双角色(源/接收器)端口，可用于视频和音频。Altera DE2 SoC 具有 PAL、SECAM 和 NTSC 电视解码器，并支持 VGA。VGA 接口是经典标准之一，大多数 PC 仍然支持将其作为在 CRT 显示器[1]上显示系统信息的默认启动接口。基本分辨率为 640 列 480 行，刷新率

1 译者注：如今，CRT 显示器已经很少见了，但许多显示器仍支持 VGA。

为 60 Hz。我们将使用这个基本的 VGA 模式，因为大多数显示器都支持它，即使是液晶显示器通常也有 VGA 输入。这允许我们直接将 ZyBo 和 DE1 SoC 开发板连接至 CRT 显示器使用，而无须购买任何新的显示器硬件。

在 DE1 SoC 板上使用的基本 VGA 系统设置如图 2.4 所示。FPGA 提供必要的控制和 RGB 数据信号。DE1 上使用的 VGA 芯片是 Analog Devices ADV7123KSTZ140，包括三个高速 10 位 DAC，可以以高达 140 MHz 的像素率运行[AD98]。因此，我们可以使用 VGA(640×480)、SVGA(800×600)、XGA(1024×768)或 SXGA(1280 ×1024)显示模式。注意，DE1 只用了 ADV7123 三 DAC IC 的 10 位数据中的高 8 位，因为处理 10 位数据太麻烦了，而且大多数图像都具有 8 位 RGB 分辨率。另一边的 ZyBo 板没有专用的 VGA 芯片。它使用 R-2R 梯状阵列(ladder array)将 RGB 数字数据直接转换为模拟信号。绿色用 6 位，红色和蓝色各用 5 位，总共有 16 位或 65K 种不同的颜色。HSYNC 和 VSYNC 信号由 FPGA 产生。

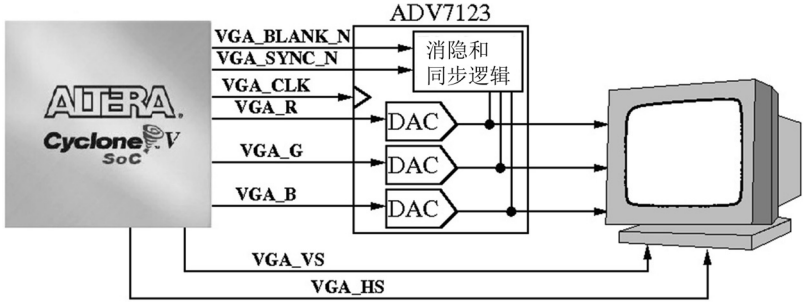


图 2.4 DE1 SoC 板卡上的总体 VGA 配置

表 2.4 展示了显示模式和所需的像素时钟(总行数×列数×帧速率)、像素数(以 M10K 块为单位)和灰度图像内存要求，即行×列×8。彩色图像需要三倍的存储空间。注意，即使是 640×480 分辨率的黑白图片，也需要超过 30 个 M10K 大小的嵌入式内存块来将图像存储在芯片上。彩色或较大的图像很可能需要进行额外的片外存储。VGA 信号的总体时序由视频数据和额外的同步时序组成，因为我们需要让 CRT 的电子束返回行开头或第一行。此处假设使用外部同步，而不是使用嵌入在绿色视频通道中的同步。然后，VGA 中的行信号以一个低电平有效的水平同步脉冲开始。让我们简要地看看 VGA 信号在刷新率为 60 Hz 时的时序。同步时长为 3.8 μs，在 25 MHz 的像素时钟下为 96 个时钟周期。接下来是 1.9 μs 或 48 个周期的水平后沿。然后 640 个时钟周期即 25.4 μs 时，视频信号出现，接着是 0.6 μs 即 16 个时钟周期的前沿信号，让 CRT 电子束返回到行的起点。总的来说，一行视频信号需要 96+48+640+16=800 个时钟周期。垂直时序也有类似规格。我们还是观察默认的具有 480 行的 VGA 信号。垂直同步脉冲需要 2 行，然后是 33 行的垂直后沿。总计需要 2+33+480+10=525 行周期来完成图像的显示。

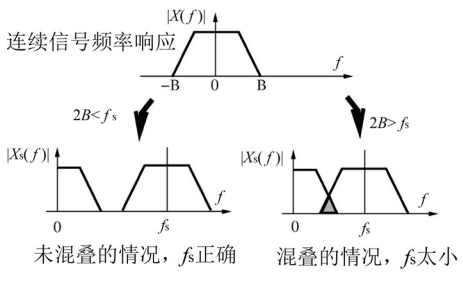
表 2.4　刷新率为 60 Hz 的 VGA 配置信息

模式	分辨率 列 x 行	总列 x 行	像素时钟 /MHz	像素/10K	大小、8 位灰度/Mbit	大小、3x8 彩色/Mbit
VGA	640×480	800×525	25.175	30.72	2.4	7.37
SVGA	800×600	1056×628	40.0	48	3.84	11.52
XGA	1024×768	1344×806	65.0	78.64	6.29	18.87
SXGA	1280×1024	1688×1066	108.0	131.07	10.49	31.46

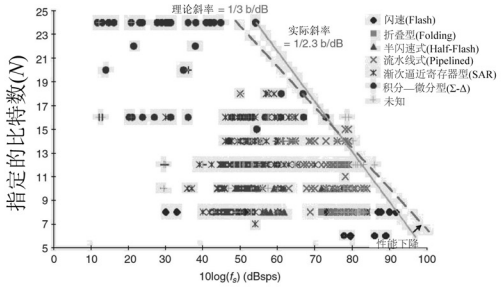
2.2.4　模拟接口

嵌入式系统通常用于监视、控制或处理模拟数据，例如，视频或音频信号、电压、温度、光强、速度、加速度计等。因此 FPGA 板卡通常会有模数转换器(Analog to Digital Converter，ADC)和数模转换器(Digital to Analog Converter，DAC)。一些第六代和第七代 Xilinx FPGA 甚至有片上快速 ADC。Xilinx Zynq-7K 有两个带有额外温度和电源传感器的片上 ADC(12 比特，1 MSPS)。该 ADC 还有一个 17 通道输入模拟多路复用器，支持同时监控许多模拟信号，见图 1.12b。

在开始讨论 ADC 和 DAC 的技术之前，让我们简要回顾一下处理模拟信号转换时的两个重要设计考虑因素：*采样率*和要使用的*比特数*。归功于香农定理，确定适当采样率的工作直截了当。我们知道，当对模拟信号进行采样时，其傅里叶频谱随采样频率 f_S 而有了周期性。因此，我们应该使用比输入信号的最高频率分量高两倍的采样率；否则，频谱会重叠，发生所谓的混叠(aliasing)，见图 2.5a。



(a) 香农采样理论的图形化解释



(b) ADC原理概览

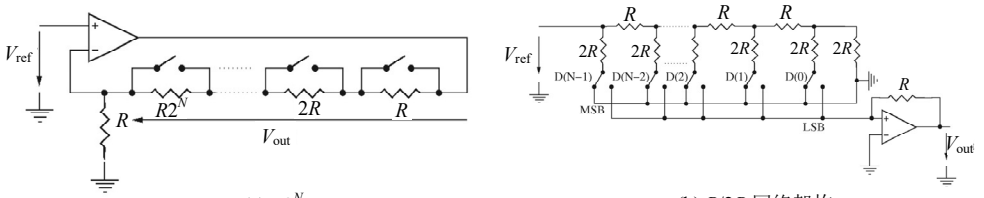
图 2.5　香农采样理论的图形化解释与 ADC 原理概览

因此，如果我们想监测 60 Hz 的电力线信号，按照香农定理，采样频率最低要 120 Hz。我们通常会使用高几个 Hz 的采样频率，因为具有 0~60 Hz 通带和开始于 60+ Hz 的阻带的抗混叠滤波器不能以零过渡带构建。一个可实现的滤波器至少要有几赫兹的过渡带，所以要达到足够的抑制，我们的采样率应该至少是阻带频率的两倍。而所需的抑制可以通过所用的第二个主要 ADC 参数，即比特数来确定。遗憾的是，比特数不能像采样率那样直接用数学方法来确定，因

为这可能取决于生理机能。例如，我们的耳朵对噪音非常敏感，因此需要较高的信噪比(Signal-to-Noise，S/N)我们才会感到音频信号可接受。因此，典型的音频信号被处理为 16 比特或更多；至于我们的眼睛，则对量化噪声不太敏感，通常 8 比特就足够了，精打细算的情况下用于颜色的比特位就更少。注意到，每个比特对总体信噪比的质量贡献约为 6 dB(见练习 2.46)。因此，一个好的音频抗混叠滤波器应该有 16×6=96dB 的抑制，而图像处理抗混叠滤波器可能只需要 8×6=48 dB。因此，我们应该使用的采样频率，至少两倍于混叠滤波器达到输入频谱时所需的抑制频率。

现在，我们有了两个系统设计参数：采样率和比特数，接下来我们应该开始寻找合适的 ADC 和 DAC；常用 ADC 架构概览见图 2.5b。由于 DAC 通常更容易理解，因此让我们先来看看其典型设计。这需要用到一些电路方面的知识，这些知识应该是现代高中物理课程的一部分，如果你不熟悉，可以从电子学教科书的介绍中查找有关 R/L/C 网络和运算放大器(Operational Amplifier，OP)的基础知识。

$R2^N$ DAC 设计实现了一个电阻值递增的网络，见图 2.6a。具有零输入电流的 OP 将保证电流恒定在 $I = V_{ref}/R$。因此，开关的关闭或打开将为输出电压 V_{out} 增加一个等效电压值，因为 OP 确保通过 $R2^N$ 网络的电流保持恒定。然而，从 VLSI 的角度来看，对于更大的比特宽度，这不是一种非常有效的设计方案，因为构建电阻网络所需的 VLSI 区域与网络大小成正比，因此电阻 $R2^N$ 将需要 2^N 倍于电阻 R 的面积，这就是为什么图 2.6b 中显示的 $R/2R$ 网络可能看起来更复杂，却通常给出一种更有效的设计方案。与开关位置无关，所有输出(MSB…LSB)从 OP 连接到虚地。并联电路中 $2R||2R = R$，$R + R = 2R$，因此每级电流除以 2。输出电压因此与电流的和成正比，而电流反映了输入数据字 $D(N–1…0)$的编码。转换器的速度可以非常高，只取决于开关的速度。ADC 通常需要用到比 OP 和几个寄存器更多的硬件资源，下文详述。



(a) $R2^N$　　(b) $R/2R$ 网络架构

图 2.6　$R2^N$ 与 $R/2R$ 网络架构

从图 2.5b 中给出的技术概述，我们得出的结论是，如今只有几种不同类型的 ADC 仍被使用。纵轴表示比特数，横轴表示最大采样率。从这些技术概览中可以看出，通常有三种不同类型的 ADC 被使用：*渐次逼近寄存器*(Successive Approximation Register，SAR)、*闪速*(Flash)和*积分微分型*(ΣΔ)ADC。

在速度和分辨率方面处于中间位置的是渐次逼近寄存器型模数转换器。该转换器使用刚刚讨论过的 DAC 来渐次减小输入与输出的差值，输入通常使用采样保持(Sample and Hold，S&H)电路或跟踪保持(Track and Hold，T&H)来采样，而输出是 DAC 的输出。这是个迭代过程，可能

需要花费几个时钟周期才能找到采样值和 DAC 输出之间的最佳匹配，但有利的一面是，所需用到的资源很少。这种来自凌力尔特公司的转换器(LTC2308)被用于 DE1 SoC 上，能够提供一个 500 kSPS、8 通道、12 比特的转换器，见图 2.7。

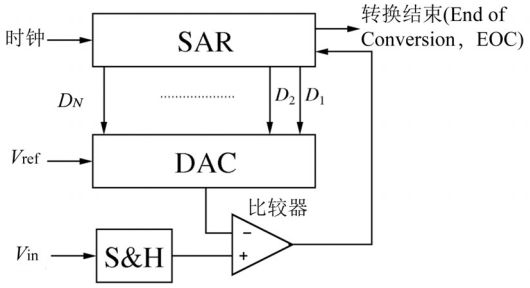
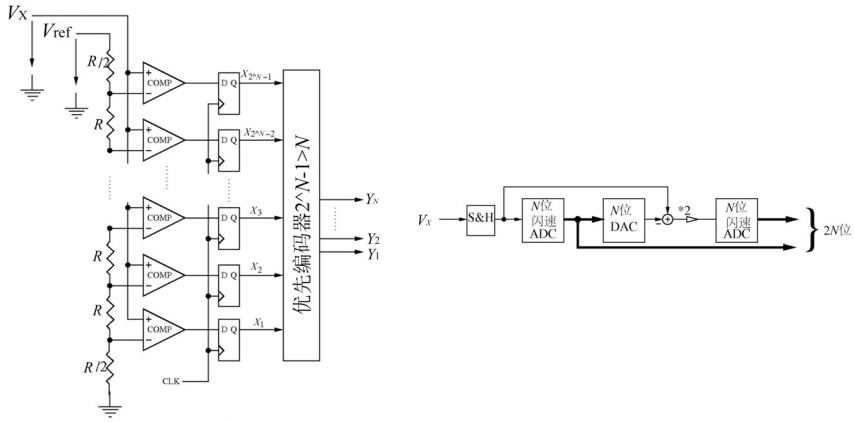


图 2.7　DE1 SoC 中使用的渐次逼近寄存器(SAR)型模数转换器

高速、低分辨率并具有 8~10 位精度和超过 1GSPS 的采样率的转换器，只有用闪速转换器或半闪速转换器才能实现，通常用于视频信号处理或高速数字示波器。这些转换器由一系列电阻/运算放大器组成，使用运算放大器在比较器模式下“测量”输入，即具有开放的反馈环路。如果输入电压大于指定的 **R** 阵列分压器值，OP 放大器的输出将切换，同样会导致其下面更低的 OP 参考电压继续此操作。我们将从所有输入中找到具有最高优先级的 OP，它将决定输入电压。对于 8 比特数据，我们需要使用 256 个运算放大器，从而将这种方法限制为低位宽。半闪速架构会使用两个较小的闪速转换器，并通过减去第一次粗量化的结果来进行第二次细量化转换。因此，对于 8 位转换器，我们需要使用 2×16＝32 而不是 256 个转换器，见图 2.8b。

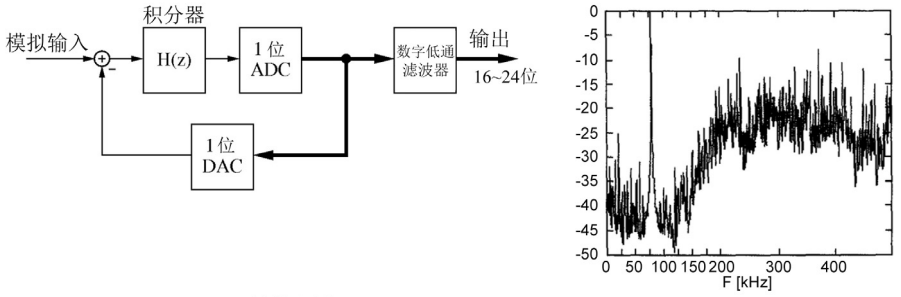


(a) 高速、低分辨率闪速转换器　　(b) 半闪速转换器中联合的两个闪速转换器

图 2.8　闪速转换器

另一个极端是 ΣΔ 型的高分辨率低速转换器。在这里，输入以非常高的速率进行采样，这些采样在反馈回路中累积在积分器中，这样积分器就会跟随输入，而+1 和−1 脉冲的数量表示输入模拟信号的大小。然后用带下采样器的数字滤波器对 1 位高速值序列进行滤波，产生高位宽输

出数据。积分器回路还将“整型”噪声频谱，将量化噪声移到更高的频率，从而实现非常高的分辨率，见图 2.9b。这种转换器是 DE1 SoC 上的 Wolfson WM8731[Wol04]音频编解码器(Coder/Decoder，CODEC) 和 ZyBo 上的 Analog Device SSM2603 的一部分。这些器件是 24 位双(立体声)ADC 和 DAC，可编程采样率为 8~96 kHz。两个板卡都有标准 3.5 mn 立体声音频插孔，能够直接连接音频信号到编解码器。这些设备的配置和数据流通过串行 I^2C 通信协议完成，该协议允许设置设备状态寄存器，从而改变采样率、数据格式(16/24/32 位，立体声/单声道)，以及输入增益，如图 2.12 所示。



(a) ΣΔ转换器原理

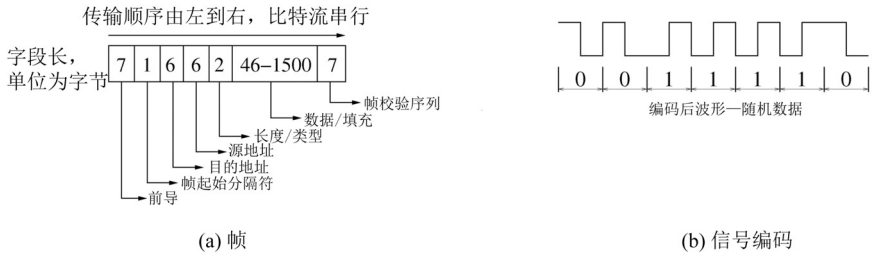
(b) ΣΔ上对于一个77.5 kHz的信号下采样前的频谱[Hut92]

图 2.9 ΣΔ 转换器原理与一个频谱示例

2.2.5 通信

目前在嵌入式系统中使用各种各样的低速和高速通信协议，我们所用的板卡支持最流行的格式。对于低带宽协议，如 UART、CAN、I^2C、PS2、SPI 和 IR 发射器/接收器，由于其数据速率不太高，协议也不太复杂，大部分协议处理可以使用 FPGA 片上资源完成。对于高带宽通信，如 USB 1/2/3 或 10/100/1000 Mbit/s 的以太网，除了特殊的电气或光学连接器，现在还需要对其进行更复杂的(帧)处理，通常由外部物理 IC(PHY)完成。ARM cortex A9 还通过片上自定义 IP 支持多个标准：2×SPI、2×I^2C、2×CAN、2×UART、2×USB 和 2×Ethernet。不过，56 引脚多路复用器 I/O(Multiplex I/O，MIO)用于这些 IP 块，这样在一个系统配置中通常一次只有两个 IP 可以连接到 I/O 引脚。下面简要回顾一下在这两个电路板上使用的最重要的协议。

以太网通信是由 IEEE 802.3 标准化的极其快速的通信协议，广泛用于局域网(Local Area Network，LAN)技术。在两块板卡上，它支持 10 Mbit/s(称为 10Base-T)、100 Mbit/s(100 Base-TX)和 1 Gbit/s(1000 Base-T)三种传输速率。它最初由 DEC、Intel 和 Xerox 提出，作为带冲突检测的载波监听多路访问(Carrier Sense Multiple Access Collision Detection，CSMA/CD)协议，现在用于 80%以上的局域网中。在个人电脑中，则通过标准 CAT-5 无屏蔽双绞线进行同步串行传输。双电缆差分线 TRD+和 TRD-由一对 LED 增强，用于指示导线上的链接速度(10/100/1000)和活动/无活动状态。为了避免出现基线漂移，信号采用曼彻斯特编码；每个比特具有半时钟长度的 0 或 1 值，即传输的 0 为时钟中间的下降沿，1 为时钟中间的上升沿，见图 2.10。



(a) 帧　　(b) 信号编码

图 2.10　典型以太网电缆中的以太网帧使用的物理信号

以太网部分，DE1 板卡上使用了一片 Micrel KSZ9021RN PHY 芯片，而在 ZyBo 上由 Realtek RTL8211E-VL PHY 来实现 10/100/1000 以太网选项。典型以太网电缆中的以太网帧使用的物理信号如图 2.10 所示。前导(Preamble，PRE)有 7 个字节，由交替的 1/0 组成，用于同步。帧起始分隔符(Start of Frame Delimiter，SFD)长一个字节，由交替的 0/1 加上结尾的两个 1 组成。目的地址(Destination Address，DA)与源地址(Source Address，SA)长度相同，为 6 字节，即 46 位[应为 48 位]。用于表示数据块长度的字段占用 2 字节，但实际允许的最大数据块为 1500 字节，这些数据块本身紧随在其长度字段之后。当长度字段超过 1536 字节时表示可选的类型帧。帧校验序列(Frame Check Sequence，FCS)是一个 32 位的循环冗余校验字，用于校验 DA、SA、长度和数据。

通用串行总线即 USB 是广泛使用的 PC 标准，数以百万计的闪存驱动器，以及我们提到的 FPGA 编程电缆，都使用 USB。除了提供各种标准，USB 还能供电，有时足以为整个 FPGA 板供电，如 Xilinx Nexys。为了避免损坏 USB 组件，它使用不同的主和从连接器。最初的 USB 1 支持 12 Mbit/s 的全速数据速率，USB 2 标准支持 480 Mbit/s 的数据传输速率，而最新的 USB 3 所支持的数据传输率最高可达 10 Gbits/s。

DE1 SoC 的 USB 使用了 SMSC USB3300 控制器，ZyBo 使用 FTDI FT2232HQ USB 桥接器以及支持 USB 2.0 标准的 Microchip USB3320 收发芯片，因此我们将在下面简要讨论 USB 2.0。USB 2.0 有 4 个引脚：5V 的 VCC、接地、差分信号 Data-和 Data+，差分信号大多数时候呈现相反的电压。信号编码在 USB 中是用非归零抑制(Non Return to Zero Inhibit，NRZI)码，0 为跳变而 1 则表示不跳变，编码示例见图 2.11。只有在包的末端才使用全零信号，称为单端零。

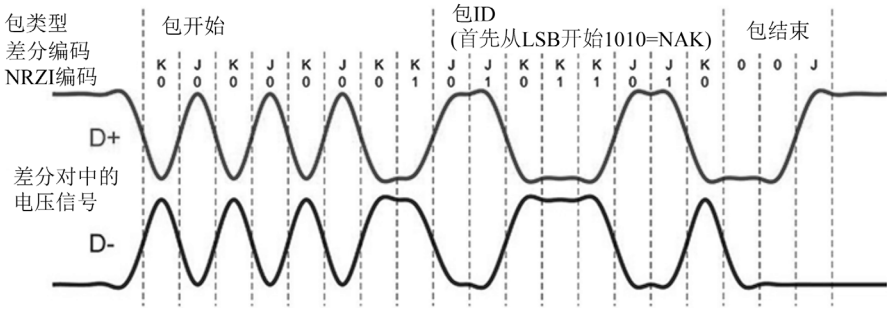
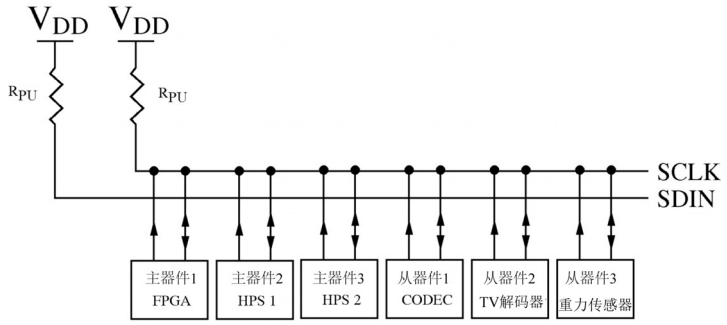


图 2.11　USB 信号示例，即握手包、同步序列；含未被接受的 PID 编码数据序列及包结束标识

单个位被分组成包。USB 用到握手、令牌和数据包类，共 16 种包类型，由 8 位传输标识符标识。在可以发送数据块之前，我们需要使用指定了方向(到设备或到主机)和设备号(7 位和 11 位地址和 5 位 CRC)的地址令牌(IN/OUT)来设置传输方式。数据包以同步字段 00000001_2 或 KJKJKJKK 开始，如图 2.11 所示，后面跟着 8 位 PID。接下来是 0~1024 字节的数据，接着是 16 位的 CRC 校验和 EOP 信号，详见[Axe14]的完整细节。

与音频编解码器(DE1 SoC 上的 Wolfson WM8731 编解码器和 ZyBo 上的 Analog Device SSM2603 编解码器)的通信是通过两块板上的 I^2C 协议完成的。因此，我们应该更详细地研究这个标准。DE1 有三个 I^2C 主器件(HPS1、HPS2 和 FPGA)，除 CODEC 外，还有两个从器件，电视解码器(w/r 地址 0×40 和 0×41)和重力传感器(w/r 地址 0xA6 和 0xA7)都使用 I^2C 进行通信。I^2C 是一种串行协议，主要使用两根线：单向时钟 SCLK 和双向数据信号 SDIN(见图 2.12)。时钟和数据线均采用线与(Wired-OR)或开漏信号，并配有上拉电阻，DE1 SoC 的 R= 2.2 K，ZyBo 的 R = 1.5 K。主控器以 ND 为逻辑零，以高阻 1 为逻辑 1，并通过上拉电阻得到逻辑 1。MODELSIM 中，高阻值用蓝色线显示在零电平和逻辑 1 电平之间的 50%处，见下面的图 2.13。



(a) I²C 总线架构

寄存器	B8	B7	B6	B5	B4	B3	B2	B1	B0	默认值
R0 左线路进	LRIN BOTH	LIN MUTE	0	0	LINVOL[4:0]					0_1001_0111
R1 右线路进	RLIN BOTH	RIN MUTE	0	0	RINVOL[4:0]					0_1001_0111
R2 左耳机出	LRHP BOTH	LZCEN	LHPVOL[6:0]							0_0111_1001
R3 右耳机出	RLHP BOTH	RZCEN	RHPVOL[6:0]							0_0111_1001
R4 模拟音频通路控制	0	SIDEATT		SIDE TONE	DAC SEL	BY PASS	INSEL	MUTE MIC	MIC BOOST	0_0000_1010
R5 数字音频通路控制	0	0	0	0	HPOR	DAC MU	DEEMPH		ADC HPD	0_0000_1000
R6 断电控制	0	PWR OFF	CLK OUT PD	OSC PD	OUT PD	DAC PD	ADC PD	MIC PD	LINE INPD	0_1001_1111
R7 数字音频接口格式	0	BCLK INV	MS	LR SWAP	LRP	IWL[1:0]		FORMAT[1:0]		0_1001_1111
R8 采样控制	0	CLKO DIV2	CLKI DIV2	SR[3:0]				BOSR	USB/ NORM	0_0000_0000
R9 激活控制	0	0	0	0	0	0	0	0	Active	0_0000_0000
R15 复位	Reset[8:0]									没有复位

(b) DE1 SoC CODEC寄存器赋值

图 2.12　单向时钟 SCLK 和双向数据信号 SDIN

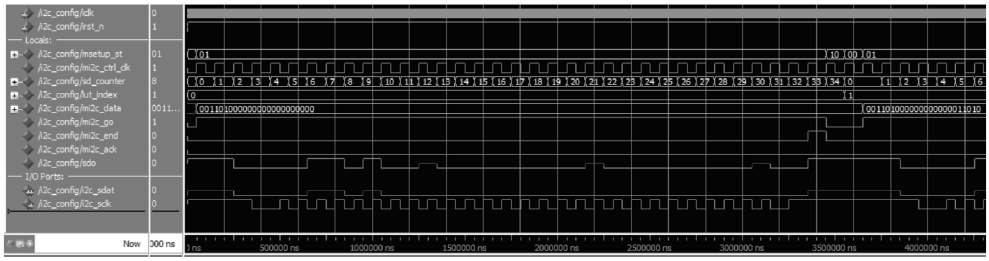


图 2.13　将第一个 CODEC 的寄存器值设置为 0

允许使用多个主器件，其中第一个拉低了数据信号 SDIN 的主器件会拥有总线。而 DE1 有一个额外的多路复用芯片(TI: TS5A23157)，以避免三个主器件出现任何冲突，特别是在时钟线上。I^2C 一般只用于编解码器系统的配置，对于用来与 ADC 和 DAC 上的编解码器所包含的整个数据进行通信的时钟、数据和左/右通道，也提供了选择信号。

图 2.13 展示了 DE1 与编解码器进行通信以访问第一个寄存器的示例。通信中包含 9 个时钟周期、7 位数据/地址、一个 R/$\overline{W}$位和从器件的应答位。DE1 编解码器是只写，所以 R/$\overline{W}$总是低电平。在 HDL 语言中，首先我们将传入的 50 MHz 时钟分频为更适合有线 OR 处理的速率，例如，25 kHz。传输开始的时序为主器件首先拉低 SDIN，然后拉低 SCLK。要设置一个寄存器值，我们需要 1 字节地址(0 × 34)，然后是 7 位寄存器号和 9 位寄存器值，如图 2.12b 所示，总共是 24 位。这 24 位会被附加 3 个应答位(所有应答都应该被从机拉低)；参见 sd_counter 时钟 12、21 和 30。停止符号对应的编码时钟 SCLK 为高电平，然后 SDIN 为高电平。接着，三个应答信号的组合决定 FSM 是否可以继续处理下一个寄存器值(lut_index+1)，还是需要再次进行传输。从 sd_counter 可以看出，整个传输过程花费了低频时钟(25 kHz)的 35 个时钟周期。

通用异步收/发器(Universal Asynchronous Receiver/Transmitter，UART)是最古老的 PC 标准之一，也是异步标准之一。由于其使用+/−12 V 信号电平，所以被认为更安全。在旧的 PC 上，此 UART 将通过 RS232 串行 COM 端口进行通信。许多现代 PC 不再有 RS232 连接器或±12V 电源。如今我们开发所用的电缆也通常以 USB 类型连接，为了支持这些遗留的标准，FPGA 嵌入式系统板提供了 UART→USB 桥接器，它使用了 USB mini-B 型连接器。缺点是，当使用 USB 协议而不是 RS232 电压时，我们牺牲了更长的长电缆和更高的信噪比保护。ZyBo 采用了飞特帝亚有限公司(Future Technology Device International Ltd，FTDI)的 FT2232HQ 桥接器，DE1 用了 FTDI FT232R。更复杂的 ZyBo 桥接器(64 和 32 引脚)使用 4 引脚 JTAG 来实现 USB-JTAG 电路。TXD 和 RXD 用了 LED 来显示两条数据线上的活动。

UART 使用两线串行端口，用于发送 TXD 和接收 RXD，ARM Cortex-A9 中的某个 UART 核在 MIO 配置正确时负责管理协议。典型的默认设置是符号速率 f_s = 115,200 波特(得名于法国通信工程师埃米尔 • 博多(Emile Baudot))、1 停止位、无奇偶校验和 8 位字符长度。因此，比特速率为 $Rate=f_s \times N$ = 1,152,000 bit/s。传输是异步的，因此接收方会等待下降沿到来(起始位)，然后是数据位(首先是 LSB) 和停止位。由于没有使用时钟，接收机通常以 8~16 过采样来监测数据线的下降沿，并在比特周期的中间采样或在比特尾使用“积分转储(integrate-and-dump)”方法

采样数据，以最大限度地减少干扰。图 2-14 给出了用于传输 ASCII 字符 S 的 10 位时钟周期序列。

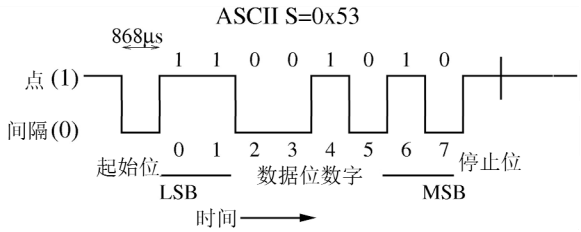


图 2.14　UART 字节传输

2.3　FPGA 结构

到了 21 世纪初，FPGA 器件系列具备了几个有吸引力的功能来实现嵌入式微处理器。这些器件提供快速进位逻辑，可以实现速度超过 300 MHz 的 32 位(非流水线)加法器[Dil00, Xil93, Alt96]、嵌入式 18×18 位乘法器、大内存块、锁相环和低偏移时钟网络。

2.3.1　Xilinx FPGA 架构概述

Xilinx FPGA 是基于早期 XC4000 系列构建的基本逻辑块，其最新的衍生产品被称为 Kintex(低成本)和 Virtex(高性能)以及 Zynq(嵌入式 SoC) (见图 2.15)。Xilinx 设备具有 FPGA 典型的大范围路由层级。

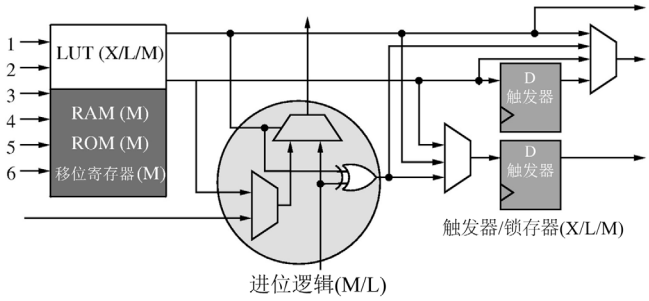


图 2.15　Zynq-7K 的四分之一，即 Artix-7 切片。L 切片有 LUT、两个触发器，以及快速进位逻辑。M 切片有所有的特性

由于 Zynq-7K 设备 XC7Z010-1CLG400C 是 Digilent Inc.提供的一款流行的 ZyBo 嵌入式系统板的一部分(见图 2.2b)，因此我们将仔细研究这个 FPGA 系列。XC7Z010 的器件整体平面图如图 2.16 所示。