第1章
绪　　论

本章概述将要在本书中研究的算法和技术。首先简要介绍一下数字信号处理技术，然后重点讨论 FPGA 技术。最后研究 Altera 的 EP4CE115F29C7 片和一个包含了芯片合成、时序分析、平面布局和功耗分析的大型设计示例。

1.1　数字信号处理技术概述

长期以来，信号处理技术一直用于转换和产生模拟或数字信号。其中最常见的应用就是信号的滤波，第 3 章和第 4 章将讨论这一问题。此外，从数据通信、语音、音频、生物医学信号处理到检测仪器仪表、机器人技术等诸多领域中，都广泛地应用了数字信号处理(Digital Signal Processing，DSP)技术。表 1-1 给出了 DSP 技术的一些应用概况[6]。

表 1-1　数字信号处理的应用

应 用 领 域	DSP 算法
通用领域	滤波和卷积、自适应滤波、检测和校正、谱估计和傅立叶变换
语音处理	编码和解码、加密和解密、语音识别和合成、扬声器识别、回声消除、人工耳蜗的信号处理
音频处理	hi-fi 编码和解码、噪声消除、音频平衡、环境声学仿真、混音和编辑、声音合成
图像处理	压缩和解压缩、旋转、图像传输与分解、图像识别、图像增强、人工视网膜的信号处理
信息系统	语音信箱、传真、调制解调器、移动电话、调制器/解调器、线路均衡器、数据加密和解密、数字通信和局域网、延拓频谱技术、无线局域网、广播和电视、生物医学信号处理
控制	伺服控制、磁盘控制、打印机控制、发动机控制、定向和导航、振动控制、电力系统监控器、机器人
仪表设备	波束成型、波形发生器、瞬态分析、稳态分析、科学仪器设备、雷达和声呐

数字信号处理(DSP)已经发展成为一项成熟的技术，并且在许多应用领域逐步取代了传统的模拟信号处理系统。DSP 系统具有如下几项优点：数字元器件对温度变化、老化以

及对元件容差不敏感。在过去，模拟芯片设计可以生产出很小体积的芯片，可是发展到今天，随着现代亚微米设计所带来的噪声，使得数字设计在集成度方面可以比模拟设计做得更好。紧凑、低功耗并且低成本的数字设计产品就应运而生了。

有两个事件加速了 DSP 技术的发展。其一是 Cooley 和 Tuckey(1965 年)揭示了一种计算离散傅立叶变换(Discrete Fourier Transform，DFT)的有效算法。第 6 章将详细讨论这类算法。另一个里程碑就是可编程数字信号处理器(Programmable Digital Signal Processor，PDSP)在 20 世纪 70 年代后期的引入，第 9 章将详细讨论该内容。这种 PDSP 能够在仅仅一个时钟周期内完成(定点数)“乘-累加”的计算，与同一时代以“冯·诺伊曼”(Von Neuman)式微处理器为基础的系统相比有着本质上的改进。现代的 PDSP 可以包含更加复杂的功能，例如：浮点数乘法器、桶式移位器、存储器组以及零架空的 A/D 和 D/A 转换器接口。EDN 每年都会出版一份有关可用的 PDSP 的详细综述[7]。在研究了 FPGA 的体系结构之后，第 2 章和第 9 章将继续研究 PDSP。

图 1-1 给出了一个借助于数字信号处理系统实现过去由模拟系统实现的典型应用示例。模拟输入信号通过一个模拟抑混叠滤波器进入系统，该滤波器的阻带起始于采样频率 f_s 的一半，用于抑制采样过程中出现的异常镜像频率。随后是模拟数字转换器(Analog-to-Digital Converter，ADC)，通常由采样和保持以及量化(和编码)电路构成。接下来由数字信号处理电路执行与模拟系统中相同的处理过程。然后就可以进一步处理或存储(例如在 CD 上)数字化处理的数据，也可以通过数字模拟转换器(Digital-to-Analog Converter，DAC)生成等价于模拟系统输出的模拟输出信号(如音频信号)。

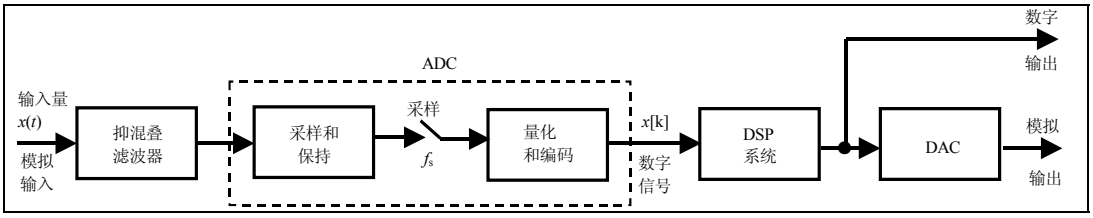


图 1-1 一个典型的 DSP 应用示例

1.2 FPGA 技术

VLSI(Very Large Scale Integration，超大规模集成)电路可以按图 1-2 进行分类。FPGA 属于现场可编程逻辑(Field-Programmable Logic，FPL)器件。FPL 被定义为如下可编程器件：包含可反复使用字段的小规模逻辑模块和元件的可编程器件[1]。鉴于 FPGA 是专用的集成电路，所以该技术可以认为是一种专用集成电路(Application Specific Integrated Circuit，ASIC)技术。但是，通常假设典型 ASIC 电路的设计需要额外的半导体处理步骤，而 FPL 则不需要这些步骤。这些额外的处理步骤能够为更高阶的 ASIC 提供性能和能耗优势，但

1. Xilinx 称之为片或可配置逻辑模块(Configurable Logic Block，CLB)，Altera 称之为逻辑单元(LC)、逻辑元素(LE)或自适应逻辑模块(ALM)。

同时也带来了高额的一次性工程成本(Non Recurring Engineering，NRE)。在 40 纳米级别，NRE 成本大约是 400 万美元[8]。另一方面，门阵列通常由“NAND 门海”构成，用户在“网表”(wire list)中定义其功能属性。在整个制造过程中都要使用这一网表，以便获得最终明确清晰的金属层布线。但可编程门阵列解决方案的设计人员可以完全控制设计的实现过程，而不需要任何实际的集成电路制造设备，也不会因为后者而延缓设计进度。1.3 节给出了 FPGA/ASIC 更详细的比较。

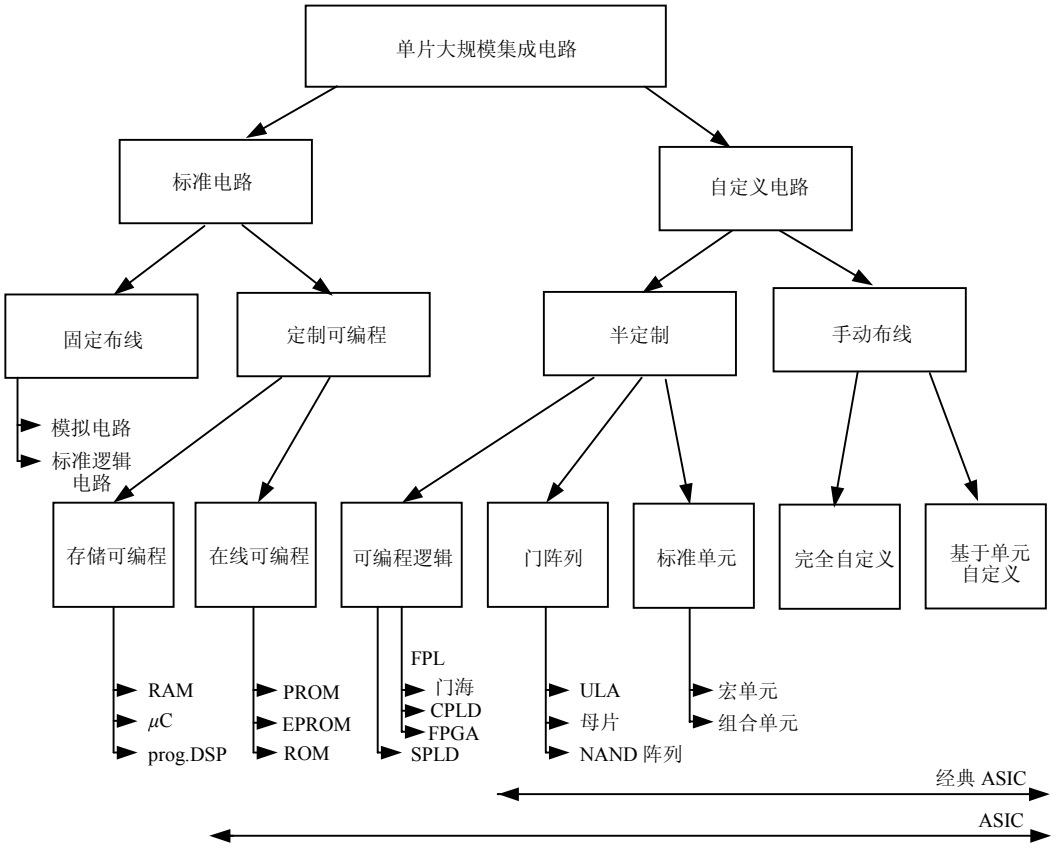


图 1-2　VLSI 电路的分类

1.2.1　按颗粒度分类

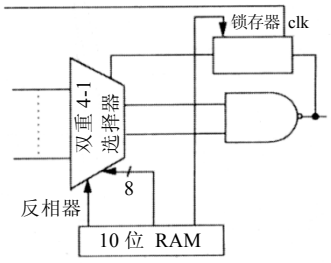
逻辑模块规模与器件的颗粒度相关，而器件的颗粒度又与模块之间需要完成的布线(路由通道)工作量相关。通常，3 种不同颗粒度分类如下：

- 小颗粒度(Pilkington 或“门海”体系结构)
- 中等颗粒度(FPGA)
- 大颗粒度(CPLD)

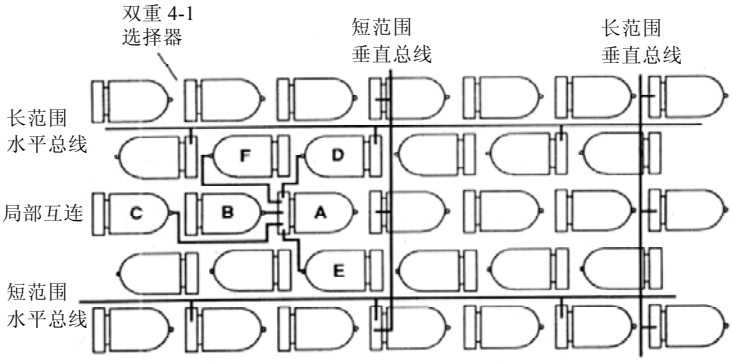
1. 小颗粒度器件

由 Pilkington 半导体公司提供的小颗粒度元器件最初获得了 Plessey 公司和 Motorola

公司的许可证。基本逻辑单元包括一个单一“与非”(NAND)门和一个锁存器(请参考图 1-3)。因为采用“与非”门可以实现任何二进制逻辑函数(请参阅练习 1-1)，所以“与非”门被称作通用函数。这一技术连同已被认可的逻辑合成工具(如 ESPRESSO)一起，还应用于门阵列的设计之中。门阵列的与非门之间的布线采用额外的金属层来实现。但对于可编程的体系结构，这就成为一个瓶颈，因为与已经实现的逻辑函数相比，它对布线资源的利用率非常高。此外，构建一个简单的 DSP 对象就需要大量的“与非”门。例如：一个高速的 4 位加法器就要用大约 130 个“与非”门。这使得小颗粒度技术在实现大多数 DSP 算法时并没有什么吸引力。



(a) 基本逻辑模块



(b) 布线体系结构

图 1-3 具有 10K 个“与非”逻辑模块的 Plessey ERA60100 体系结构(© Plessey[9])

2. 中等颗粒度器件

最常见的 FPGA 体系结构如图 1-4(a)所示。图 1-1 与图 1-12 给出了一个当前中等颗粒度 FPGA 器件的具体示例。具有代表性的基本逻辑模块是小规模的表(典型的具有 4 位或 5 位的输入表，1 位或 2 位的输出)或由专用的多路复用器(multiplexer, MPX)逻辑来实现，比如在 Actel 的 ACT-2 器件中所使用的 MPX[10]。布线通道的选择范围是从短到长。带有触发器的可编程 I/O 模块就附在器件的物理边缘。

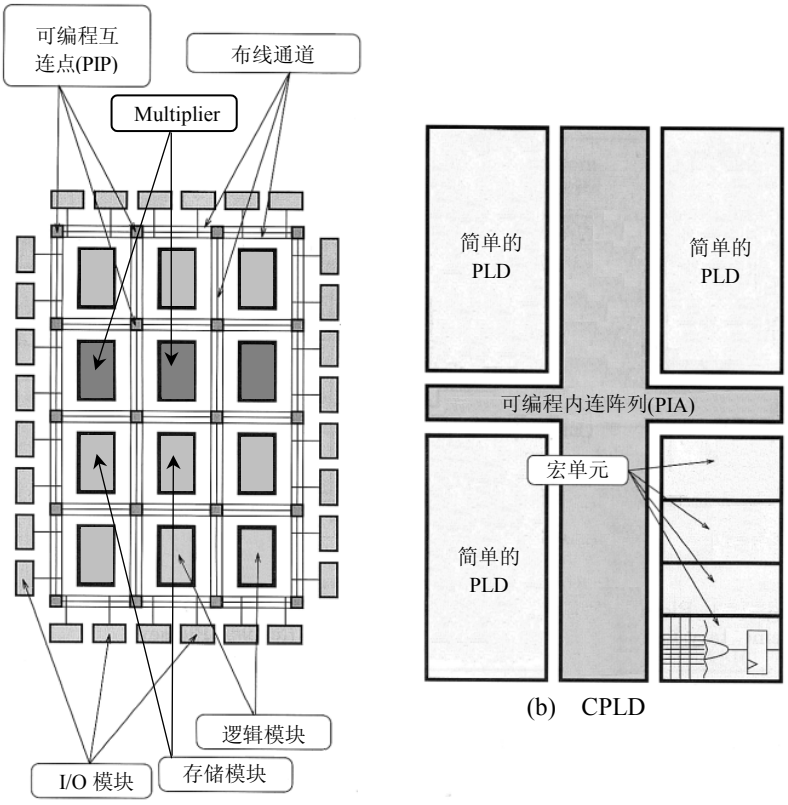
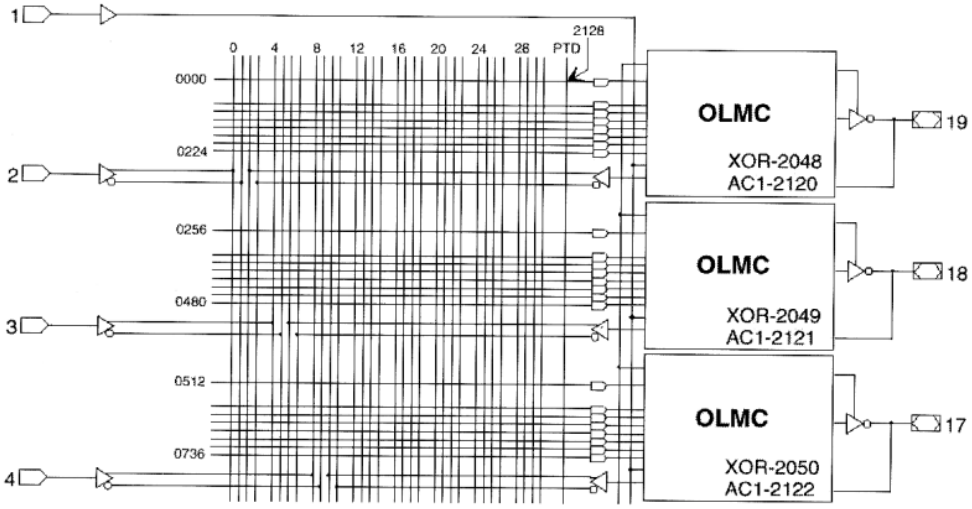


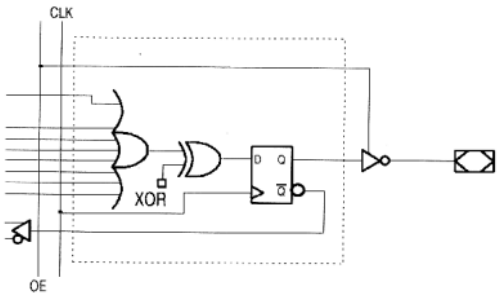
图 1-4　FPGA 和 CPLD 的体系结构

3. 大颗粒度器件

图 1-4(b)给出了大颗粒度器件(如复杂可编程逻辑器件(Complex Programmable Logic Device, CPLD)的特征。这些复杂的可编程逻辑器件(CPLD)可以定义成由简单的可编程逻辑器件(Simple Programmable Logic Device, SPLD)组成，如图 1-5 所示的典型 GAL16V8 芯片。该 SPLD 芯片由一个充当与/非阵列的可编程逻辑阵列和一个通用 I/O 逻辑模块组成。通常，CPLD 中的 SPLD 具有 8 到 10 个输入端，3 或 4 个输出端，并且支持大约 20 个乘积项。在这些 SPLD 模块之间的宽总线(Altera 称之为可编程互连阵列(Programmable Interconnect Array, PIA))上延时很短。通过将总线与固定的 SPLD 的时序结合起来，就能为 CPLD 提供引脚之间可预先计算的短暂延迟。



(a) 8 个宏单元中的前 3 个



(b) 输出逻辑宏单元(OLMC)

图 1-5　GAL16V8(© Lattice [11])

1.2.2　按技术分类

实际上，FPL 几乎可用于所有存储技术(SRAM、EPROM、E^2PROM 和抑熔断技术[12])。根据具体的技术可以将器件定义为可重复编程或一次性编程。大多数 SRAM 器件都可以通过单比特流来编程，从而降低了布线要求，但是也相应地增加了编程的时间(通常在毫秒(ms)级范围)。FPGA 的主导技术——SRAM 器件基于静态 CMOS 存储技术，并且在系统上可编程和可重复编程。然而，它们还需要一个外部“引导”器件用于配置。由于电可编程只读存储器(Electrically Programmable Read-Only Memory，EPROM)需要用紫外线照射来擦除，因此经常用在一次性 CMOS 可编程模式中。CMOS 电可擦可编程只读存储器(Electrically Erasable Programmable Read-Only Memory, E^2PROM)可以用作可重复编程和系统编程器件。EPROM 和 E^2PROM 的建立时间都很短。因为编程信息不必下载到器件中，所以能够得到更好的保护，禁止未授权使用。近来出现了一种基于 EPROM 的，称为闪存(flash memory)的革新技术。这类器件通常被视为“页方式”的，具有更小的单元，在系统上可重复编程系统中，等同于 E^2PROM 器件。最后，在表 1-2 中简要地给出了不同器件对

应技术的主要优缺点。

表 1-2　FPL 技术

技　　术	SRAM	EPROM	E^2PROM	抑 熔 断	Flash
可重复编程	√	√	√	—	√
在系统上可编程	√	—	√	—	√
易失性	√	—	—	—	—
复制保护	—	√	√	√	√
产品示例	Xilinx 的 Spartan、Altera 的 Cyclone	Altera 的 MAX5K、Xilinx 的 XC7K	AMD 的 MACH、Altera 的 MAX 7K	Actel 的 ACT	Xilinx 的 XC9500、Cypress 的 Ultra 37K

1.2.3　FPL 的基准

为 FPL 器件提供客观的标准是一项重要的任务。通常要根据设计人员的经验和技巧以及设计工具的功能来预测其性能。为了确定有效的基准，Xilinx[13]、Altera[14]和 Actel[15]共同建立了可编程电子产品性能协议(Programmable Electronic Performance Cooperative, PREP)，到目前为止已经有 10 多个成员。PREP 已经为 FPL 建立了 9 种不同的基准，表 1-3 总结了这些标准。遵循基准是为了让每个厂商利用自己的器件和软件工具在指定的器件中尽可能多次地实现简单的模块，同时尽可能地提高速度。器件中相同逻辑模块的实例化次数被称作重复率(repetition rate)，这是所有基准的基础。相对于 DSP，表 1-3 中的第 5 个和第 6 个基准是相关联的。在图 1-6 中按照相对频率的形式给出了 Altera(A_k)和 Xilinx(x_k)系列典型 FPGA 与 CPLD 器件的重复率，这些器件通常用于大学开发板。这些并不总是可用的最大器件，但所有器件都通过基于网络版本设计工具的支持。Xilinx 的速度似乎更高，而 Altera 的 FPGA 则具有更大数量的重复。由此可以得出结论：与 CPLD 相比，现代的 FPGA 系列提供了最佳的 DSP 复杂度和最大速度。这要归结于现代器件提供了允许快速进位逻辑延迟(每比特小于 0.1 纳秒)的功能(请参阅 1.4.1 节)，不需要昂贵的“超前进位”译码器就可以提供多位宽的快速加法电路。尽管 PREP 基准对于比较等效门数和提高速度有用，但对于具体的应用，其他属性也非常重要。这些属性包括：

- 阵列乘法器(如 18 位×18 位、18 位×25 位)
- 封装(如球栅阵列(Ball Grid Array, BGA)、扁平封装(Thin Quad Flat Package, TQFP)、管脚阵列(Pin Grid Array, PGA))
- 通过 DES 或 AES 构造数据流加密
- 嵌入式硬件微处理器(如 32 位的 ARM Cortex-A9)
- 片上大模块尺寸 RAM 或 ROM
- 片上快速 ADC
- 支持 ZBT、DDR、QDR、SDRAM 的外部存储器

- 引脚到引脚之间的延迟
- 内部三态总线
- 读回译码器或边界扫描译码器
- 可编程电压变化速度或 I/O 的电压
- 功耗
- ×1、×2 或×4 总线标准接口的 IP 硬核模块

表 1-3 FPL 的 PREP 基准

编 号	基 准 名 称	说 明
1	数据信道	8 个 4-1 乘法器驱动一个并行加载的 8 位移位寄存器(请参阅图 1-26)
2	定时器/计数器	通过 8 位数值寄存器对两个 8 位的数值进行定时和比较(请参阅图 1-27)
3	小型状态机	具有 8 个输入和 8 个输出的 8-状态机(请参阅图 2-64)
4	大型状态机	具有 40 个转换、8 个输入和 8 个输出的 16-状态机(请参阅图 2-65)
5	算法电路	4 位×4 位无符号乘法器和 8 位累加器(请参阅图 4-61)
6	16 位累加器	16 位累加器(请参阅图 4-62)
7	16 位计数器	可加载的二进制递增计数器(请参阅图 9-42)
8	16 位同步降值计数器	具有同步复位的可加载 16 位二进制计数器(请参阅图 9-42)
9	存储器映射器	16 位地址空间到 8 位地址空间的映射(请参阅图 9-43)

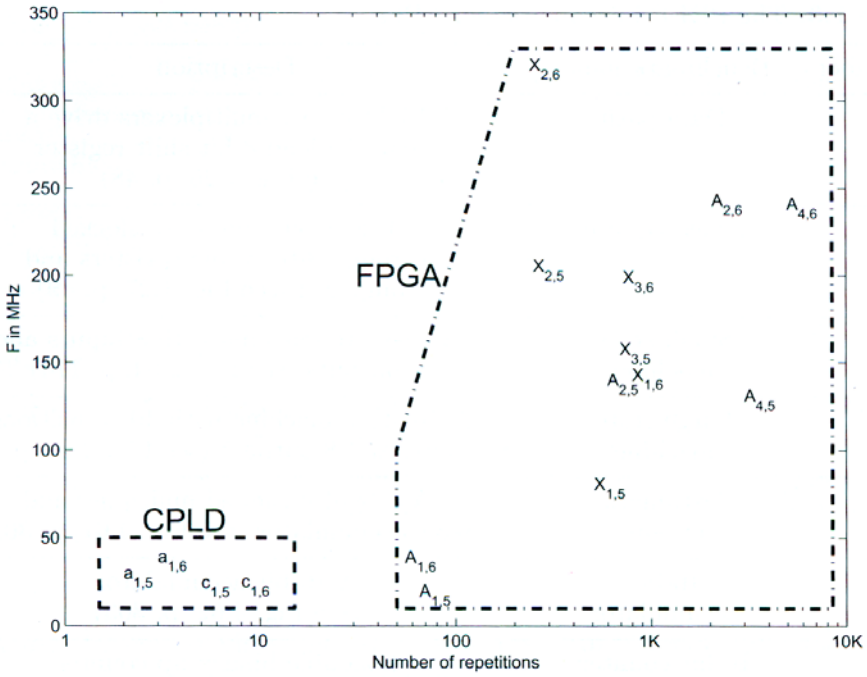


图 1-6

与其他功能相比，其中一些功能与 DSP 应用更加相关(这要取决于具体的应用)。表 1-4 和表 1-5 分别总结了 Xilinx 和 Altera 系列器件的某些关键功能的可用性。

第 1 列是器件系列名。第 2 至 9 列给出了(对于大多数 DSP 应用)相关功能：第 2 列为 LUT 地址输入(又称“扇入”)数量；第 3 列为嵌入式阵列乘法器大小；第 4 列为片上模块 RAM 大小(以 Kbit 为度量单位)；第 5 列为嵌入式微处理器：当前的 Xilinx ZYNQ 与 Altera 器件的 32 位 ARM Cortex-A9；第 6 列为是否支持 Xilinx 器件的片上快速模/数转换(第 6 代 Virtex：10 位 0.2MSPS；第 7 代：12 位 1MSPS)；第 7 列为该系列器件的价格和可用性，不再推出用于新产品的系列归为停产类，用 m 表示。低价器件标有$，而高价器件标有$$。第 8 列为该系列器件的生产日期；第 9 列为所用工艺技术(以 nm 为度量单位)。

当前 Xilinx 提供 4 类系列器件：Virtex 系列在产品性能与功能方面领先；Kintex 系列为 DPS 集中应用，且价格较低；Artix 系列价格最低，取代了 Spartan 系列器件。另外，还提供了一种名为 ZYNQ 的嵌入式微处理器。包括一个或多个 IBM PowerPC RISC 处理器的 Virtex-II、Virtex-4-FX 或 Virtex-5-FXT 系列器件不再引入新设计。Xilinx 器件有 18 位×18 位或 18 位×25 位阵列乘法器。目前大多数器件提供 18Kbit 或 36Kbit 存储。第 6 代 Virtex 增加了一个片上 10 位 0.2MSPS 快速 ADC。第 7 代包括一个 12 位 1MSPS 双通道 ADC，以附加传感器供电以及片上温度， 约 17 个传感器信号源用于 ADC，参见图 1-7(b)。需要注意的是，在许多开发软件的网络版中，只有 Spartan 系列是可用的，大多数其他器件需要 Xilinx ISE 软件的订购版。

表 1-4　Xilinx FPGA 系列 DSP 的功能

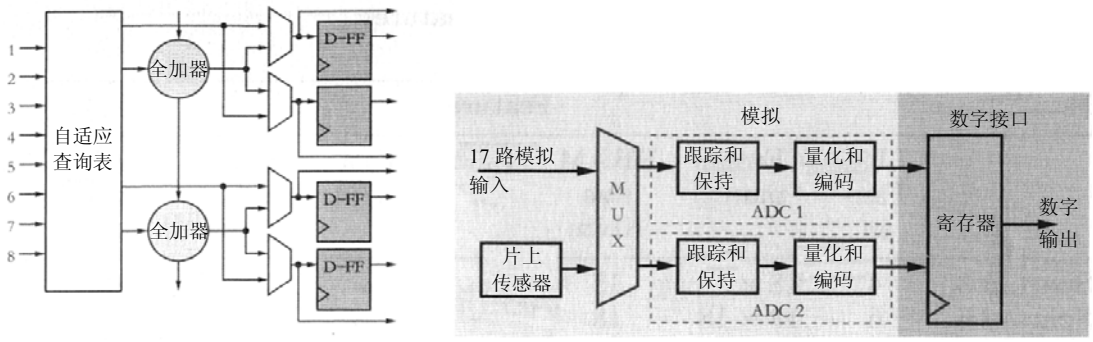
系　　列	功　　能							
	LUT 扇入	嵌入式乘法器大小	BRAM 大小/Kbit	嵌入式 μP	快速 A/D 转换	价格/停产	年份	工艺流程/nm
Spartan 3	4	18×18	18	—	—	m	2003	90
Spartan 6	6	18×18	18	—	—	$	2009	45
Virtex 4	4	18×18	36	PPC	—	m	2004	90
Virtex 5	6	25×18	36	PPC	—	m	2006	65
Virtex 6	6	25×18	36	—	√	$$	2009	40
Artix 7	6	25×18	36	—	√	$	2010	28
Kintex 7	6	25×18	36	—	√	$$	2010	28
Virtex 7	6	25×18	36	—	√	$$	2010	28
ZYNQ-7K	6	25×18	36	ARM	√	$$	2011	28

表 1-5 Altera FPGA 系列 DSP 的功能

系 列	功 能							
	LUT 扇入	嵌入式乘法器大小	BRAM 大小/Kbit	嵌入式 μP	快速 A/D 转换	价格/停产	年份	工艺流程/nm
FLEX10K	4	—	4	—	—	m	1995	420
Cyclone	4	—	4	—	—	$	2002	130
Cyclone Ⅱ	4	18×18	4	—	—	$	2004	90
Cyclone Ⅲ	4	18×18	9	—	—	$	2007	65
Cyclone Ⅳ	4	18×18	9	—	—	$	2009	60
Cyclone Ⅴ	8	27×27	10	—	—	$	2011	28
Arria	8	18×18	576	—	—	$	2007	90
Arria Ⅱ	8	18×18	9	—	—	$	2009	40
Arria Ⅴ	8	27×27	10	ARM	—	$$	2011	28
Stratix	4	18×18	0.5、4、512	—	—	$$	2002	130
Stratix Ⅱ	8	18×18	0.5、4、512	—	—	$$	2004	90
Stratix Ⅲ	8	18×18	9、144	—	—	$$	2006	65
Stratix Ⅳ	8	18×18	9、144	—	—	$$	2008	40
Stratix Ⅴ	8	27×27	20	—	—	$$	2010	28

Altera 主要提供了三类 FPGA 器件：Stratix 系列性能最高，Arria 系列性能属于中等水平，Cyclone 系列器件在三个系列中成本最低、功率最小、密度最小、性能也最低。近来，器件逻辑模块大小从 4 路 LUT 输入增加到最大 8 路不同输入，这使其可以与建立两路输入加法器几乎相同的速度建立 3 路输入加法器。ALM 具有两个双稳态多谐振荡器、两个全加器、两个 4 输入 LUT、4 个 3 输入 LUT 以及多个多路复用器以保证 6 路输入功能得以实现，见图 1-7(a)。Altera 器件的嵌入式乘法器大小分别为 9 位×9 位、18 位×18 位、27 位×27 位。较大的乘法器可以通过以降低速度为代价将模块组合在一起进行建立。从第 5 代开始，3 个 9 位模块被组合成一个快速 27 位×27 位乘法器。存储器存储范围较广，从 0.5K、M4K、M9K、M10K、M144K，一直到 M512K。需要注意的是，在 Quartus II 开发软件的网络版 12.1 版中，只有 Cyclone 系列可用，而 Arria 和 Stratix 系列器件需要软件的订购版。

FPL 功耗是另一个重要特征，在移动应用中尤其如此。CPLD 通常具有较高的“备用”功耗。在高频应用中，预计 FPGA 的功耗会更高一些。1.4.2 节给出了更加详细的功率分析示例。



(a) Altera 的 ALM 模块　　(b) Xilinx 系列 7 位高速片上的 ADC

图 1-7　最近的 FPGA 系列中所用的新的体系结构特征

1.3　DSP 的技术要求

图 1-8 给出了供应商提供的 PLD 市场占有份额。自 20 世纪 80 年代初期问世以来，在近 10 年 PLD 每年享有 20%的稳定增长率，超过 ASIC 增长率 10%以上。在 2001 年，全世界范围内微电子领域的衰退实质上延缓了 ASIC 和 FPLD 的发展。自 2003 年我们再一次看到两个市场领头羊迅猛增长(每年约 10%)。而 FPLD 胜过 ASIC 的原因部分在于 FPL 拥有许多和 ASIC 相同的优点，例如：

- 在尺寸、重量和功耗方面都有所降低
- 更高的吞吐量
- 更好的安全性能，可以禁止未授权的复制
- 降低了器件自身和开发的成本
- 降低了电路板的测试成本

而且还克服了 ASIC 的许多缺点：

- 缩短 3 到 4 倍的开发时间(快速原型设计)
- 在线可重复编程的能力
- 在少于 1000 个单元的解决方案中降低了 NRE 成本，从而可以得到更加经济的设计

CBIC ASIC 用在高端大批量(多于 1000 个副本)的应用中。与 FPL 相比较，典型的 CBIC ASIC 在相同的小片尺寸上有 10 倍以上的门个数。第二个问题的解决方案称为硬布线的 FPGA(Altera 命名为 HardCopy ASIC，Xilinx 命名为 EasyPath FPGA)，其中的门阵列用来实现已经验证的 FPGA 设计。

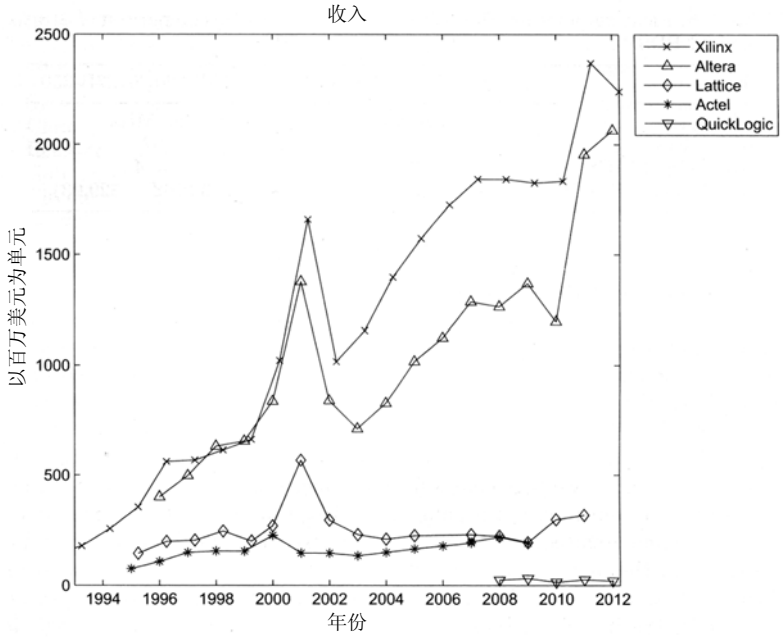


图 1-8　5 家主要厂商在 PLD/FPGA/CPLD 市场的收入

表 1-6　Stratix IV 和 TI PDSP 浮点乘法累加器性能对比

特　　征	Stratix IV	TI TMS320C6727B-350
F_{rmmax}	227MHz	350MHz
#器件	1	62
总 GFPMACS	43.5	43.4
总价(以 1000 美元为单位)	$871	$1 799(62×$29.03)

FPGA 和可编程信号处理器

通用的可编程数字信号处理器(Programmable Digital Signal Processor, PDSP)[6、16、17]在近 20 年里取得了巨大的成功。这些 PDSP 都是基于一种精简指令集计算机(Reduced Instruction Set Computer，RISC)范例的体系结构，至少由一个快速阵列乘法器(例如，16 位×16 位到 24 位×24 位定点数或 32 位浮点数)和一个已扩展字宽的累加器构成。PDSP 的优势源于大多数信号处理算法的乘-累加运算(Multiply And Accumulate, MAC)都非常频繁。通过多级流水线体系结构，PDSP 可以获得仅受阵列乘法器速度限制的 MAC 速度。第 9 章将详细介绍 PDSP。通常认为 FPGA 也能够用来实现 MAC 单元[18]，但是，如果 PDSP 能够满足 MAC 速度的需求，那么 PDSP 在成本方面将更具优势。另一方面，在许多高带宽的信号处理应用领域，如无线电、多媒体或卫星传输，FPGA 技术能够通过在一个芯片上集成多级 MAC 单元来提供更高的带宽。此外，在后续要讨论的 CORDIC、NTT 和差错

校正算法等算法中，还可以进一步证明 FPL 技术要比 PDSP 更有效率。据称[19]，未来 PDSP 将主宰需要复杂算法(例如，多重 if-then-else 结构)的应用领域，而 FPGA 将主宰更多前端(传感器)应用，如 FIR 滤波、CORDIC 算法或 FFT，介绍这些内容就是本书的主旨。

FPGA 在成本和功耗方面多年前已超越定点 PDSP。近年来，PFGA 的浮点性能得到改进，特别是在电路板尺寸、功耗和成本方面，目前提供了一个有吸引力的替代浮点 PDSP 的解决方案。对于一个典型设计，表 1.6 列出了一些关键因素。显然，在如此大规模的设计中，与比较经典的浮点 PDSP 设计相比，FPGA 提供了显著改善。在表 1-6 所示的例子中，使用 FPGA 和 DSP 进行相同的千兆/秒浮点乘法—— 累加操作(Giga Floating-Point Multiply-ACcumulate operations per Second, GFPMACS)。对于一个可以用单 FPGA 的 DE4 大学开发板即可完成的任务，我们需要 62 浮点 PDSP 来实现(DE4 板通过大学项目约$1000 可以买到，DE4 的商业价格为$3000)。对于 Stratix IV 系列来说，单次 FPMAC 需要大约 475 个 ALM 与 4 个嵌入式 9 位×9 位乘法器[20]。来自 TI 的最快的 FP PDSP 是 TI320C6727B-350，运行频率为 350MHz，并提供 700MMAC(兆次乘法加法运算)[21]。我们也注意到大量的电路板尺寸与功耗改进。整体器件成本改进超过 100%。

我们将在 2.7 节讨论通过 IP 内核和使用新的 VHDL 2008 标准(通过由 David Bishop 提供的综合库与 VHLD-1993 兼容)的浮点设计的更多细节。

1.4　设计实现

通常在 VLSI 设计中，其详细程度的层次可以涵盖从完全定制的 ASIC 几何布局到使用机顶盒的系统设计。表 1-7 给出了相应的概述。因为 FPGA 的物理结构虽然是可编程的，但也是固定的，所以在 FPGA 的设计过程中没有布局和布线任务。在门层次上采用寄存器转换设计语言就可以最佳地利用器件。投放市场时间与 FPGA 迅速增加的复杂程度共同迫使研究方法转移到知识产权(Intellectual Property, IP)宏单元(macrocell 或 mega-core cell)的使用上来。宏单元为设计人员提供了一个预先定义的功能集合，如微处理器或 UART。这样，设计人员就只需指定所选择的功能或特性(例如，精确度)，“合成器”就会生成最终解决方案的一份硬件描述代码或原理图。

表 1-7　VLSI 的设计层次

对　象	目　标	示　例
系统	性能说明	计算机、磁盘单元、雷达
芯片	算法	μP、RAM、ROM、UART、并行端口
寄存器	数据流	寄存器、ALU(运算器)、COUNTER(计数器)、MUX(多路复用器)
门电路	布尔方程	AND(与)、OR(或)、XOR(异或)、FF(触发器)
线路	微分方程	晶体管、R(电阻)、L(电感)、C(电容)
布局	无	几何形状

FPGA 技术的关键就是利用强大的设计工具来：

- 缩短设计周期
- 提高器件的利用率
- 提供合成器的选项，即在最佳速度和设计规模之间做出选择

图 1-9 给出了 CAE 工具分类方法以及在 FPGA 设计流程中的应用。设计入口既可以是图形界面，也可以是文本界面。在进行下一步设计之前，首先应进行格式检查，排除语法错误和图形设计规则错误(如导线悬空)。在功能选取中，从设计中选择基础设计信息并写入功能网表。功能网表可以进行电路的初步功能仿真(又名 RTL 级仿真)，构建一个以测试平台命名的样本数据集，为下一步测试带有时序信息的设计做准备。功能网表也允许电路的 RTL 视图。该视图给出了在 HDL 描述下电路的概述。如果功能测试未通过，就要从设计入口重新开始。如果功能测试满足要求，就进入设计实现阶段，该阶段有多个步骤，要花费更多的编译时间。然后是功能选取。设计实现结束后，FPGA 内的电路布线就完成了，它给出精确的资源数据并可以执行完整的时序延迟信息仿真和性能测试。如果所有实现的数据都符合预期要求，就可以进入实际的 FPGA 编程阶段；否则，还要回到设计入口对设计进行适当改动。利用现代 FPGA 的 JTAG 接口还可以直接监测 FPGA 内的数据处理情况：可以只读出 I/O 单元(称之为边界扫描)，也可以读回所有内部触发器的信息(也称之为完全扫描)。如果在系统上调试失败，就需要返回到设计入口重新开始。

通常，选择图形设计环境还是文本设计环境，取决于设计人员的个人偏好和经验。图形方式的 DSP 解决方案能够强调更加规则的数据流以及许多相关的 DSP 算法。而文本环境通常侧重于算法控制设计，并提供更加丰富的设计类型，后续设计示例将予以说明。具体地说，对于 Altera 的 Quartus II，利用文本方式能够设计更加具体的特性和更加精确的行为功能。

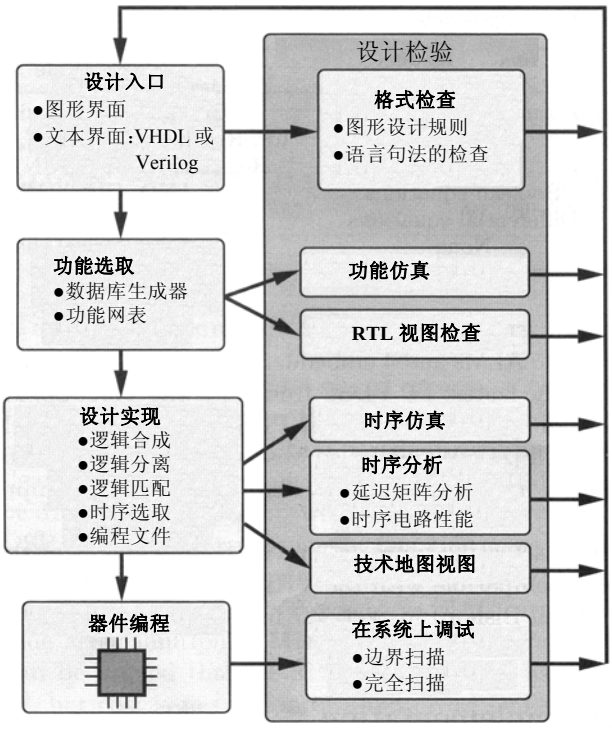


图 1-9　CAD 设计周期

例 1.1　VHDL 设计类型的比较

下面的设计示例阐述了 VHDL 环境下的 3 种设计策略。具体方法如下：

- 结构类型(组件实例化，例如图形网表设计)
- 数据流，例如并发语句
- 利用 PROCESS 模板进行顺序设计

VHDL 设计文件 example.vhd[2] 如下(“--”以后是注释)：

```
PACKAGE n_bit_int IS    -- User defined type
  SUBTYPE S8 IS INTEGER RANGE -128 TO 127;
END n_bit_int;
LIBRARY work; USE work.n_bit_int.ALL;

LIBRARY ieee;                    -- Using predefined packages
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_signed.ALL;
-- ------------------------------------------------------
ENTITY example IS                              ------> Interface
  GENERIC (WIDTH : INTEGER := 8);   -- Bit width
  PORT (clk   :  IN STD_LOGIC;      -- System clock
        reset : IN  STD_LOGIC;      -- Asynchronous reset
        a, b, op1  : IN STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
                                    -- SLV type inputs
        sum   :  OUT STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
                                    -- SLV type output
        c, d  :  OUT S8);           -- Integer output
END example;
-- ------------------------------------------------------
ARCHITECTURE fpga OF example IS
  COMPONENT lib_add_sub
    GENERIC  (LPM_WIDTH : INTEGER;
              LPM_DIRECTION :  string := "ADD");

  PORT(dataa : IN  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
       datab : IN  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
       result: OUT STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0));
  END COMPONENT;

  COMPONENT lib_ff
  GENERIC (LPM_WIDTH : INTEGER);
  PORT (clock : IN  STD_LOGIC;
        data  : IN  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
        q     : OUT STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0));
  END COMPONENT;

  SIGNAL  a_i, b_i   :  S8 := 0;    -- Auxiliary signals
  SIGNAL  op2, op3   :  STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
BEGIN

  -- Conversion int -> logic vector
  op2 <= b;
```

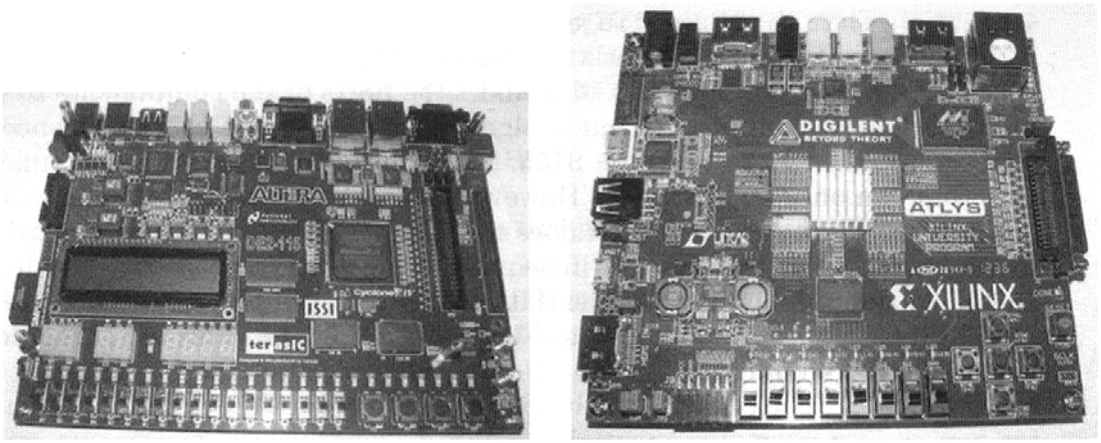
2. 等价的 Verilog 代码 example.V 可在附录 A 中找到，合成结果位于附录 B 中。

```
  add1: lib_add_sub          ------> Component instantiation
    GENERIC MAP (LPM_WIDTH => WIDTH,
                 LPM_DIRECTION => "ADD")
    PORT MAP (dataa => op1,
              datab => op2,
              result => op3);
  reg1: lib_ff
    GENERIC MAP (LPM_WIDTH => WIDTH )
    PORT MAP (data => op3,
              q => sum,
              clock => clk);

  c <= a_i + b_i;       ------> Data flow style (concurrent)
  a_i <= CONV_INTEGER(a); -- Order of statement does not
  b_i <= CONV_INTEGER(b); -- matter in concurrent code

  P1: PROCESS(clk, reset) ----> Behavioral/sequential style
  VARIABLE  s :  S8 := 0;     -- Auxiliary variable
  BEGIN
    IF reset = '1' THEN                -- Asynchronous clear
      s := 0; d <= 0;
    ELSIF rising_edge(clk) THEN -- pos. edge triggered FFs
      s := s + a_i;          ----> Sequential statement
      -- d <= s;             -- "d" at this line: b_i would
      s := s + b_i;          -- not be added to output.
      d <= s;                -- Ordering of statements matters
    END IF;
  END PROCESS P1;

END fpga;
```

所有 HDL 代码起始于 I/O 端口定义，其次是内部网络定义。在 VHDL 中使用的库需要在端口声明之前进行指定，然后开始进行实际的电路描述。该示例代码显示的是所有三种类型中的短代码：组件、并行和顺序。lib_ff 和 lib_add_sub 模块是被实例化的类似图形设计的组件库组件。我们需要指定主要参数和组件的端口连接到设计中的内部网或 I/O 端口。接下来显示的是并行代码的短序列。这里必须使用 SIGNAL 类来描述只能使用一次的单一网络。然而，这些语句的顺序并不重要。类型转换不需要像在顺序编码(例如 C 代码)一样必须放在算术运算之前。最后，一个 PROCESS 例子显示的是顺序编码类型。这里我们也可以使用局部类型 VARIABLE(仅能够在 PROCESS 中使用)。VARIABLE 不需要是电路中的单一网络，并且可以多次使用。PROCESS 中的语句顺序确实就像正常的顺序程序代码，只有指定后的声明才会被评估。例如，如果不在 IF 语句结束时指定给 d，那么 b_i 不会被加到 d。还有一点要注意的是 VHDL 语言的严格类型。需要通过功能调用来完成 INTEGER 和 STD_LOGIC 之间的转换。

关于如何合成和模拟电路(根据流程图 1-9)的详细研究将随后在 1.4.3 节展开。在 CAD 工具流程结束时，将会有一个编程文件准备就绪，可以下载到硬件开发板(如图 1-10 所示的原型开发板)。可对器件进行编程，并采用“读回”方法完成其他硬件测试。

Altera 支持多种 DSP 开发板，上面都带有整套有用的原配组件，包括快速 A/D、D/A、音频编码解码器(CODEC)、DIP 开关、独立的 7 段 LED 显示装置和按钮。这些开发板可以直接从 Altera 获得。Altera 提供 Stratix 以及 Cyclone 开发板，价格范围在$199~$24 995，除 FPGA 的规模不同外，A/D 通道的数量、精度和速度，以及存储模块等其他特征也有差别。对于大学教学，成本最低的 Cyclone IV 系列的 DE2-115 开发板是不错的选择，虽然价格相比许多数字逻辑实验室使用的 UP2 或 UP3 开发板贵很多，但它有双通道音频编码解码器、FPGA 外部的大型存储库以及其他许多有用的端口(USB、VGA、PS/2、以太网、7 段 LED 显示装置、LCD、开关、按钮等)，见图 1-10(a)。而 Xilinx 几乎不直接提供开发板，大学项目中可用的演示开发板都来自第三方。这些开发板的价格非常低廉，就如同它们是非营利性设计。例如 Digilent 公司提供的一款用于 DSP 演示的(带有片上乘法器与音频编码解码器)Atlys 开发板很不错，仅售$199 或$349(分别为学术和常规定价)，如图 1-10(b)所示。开发板上有一块 Spartan-6 XC6SLX45 FPGA 芯片、16MB 闪存、128MB 内存、8 个 LED 显示、8 个开关和 5 个按钮。对于 DSP 与视频试验，可以利用 AC-97 音频编码解码器上的 A/D 和 D/A 转换以及 4 个 HDMI 端口。



(a) Cyclone IV 系列的 DE2-115 Altera 开发板　　(b) 带有 ADC 和 DAC 音频编码解码器的 Xilinx Atlys 开发板

图 1-10　低成本原型开发板

1.4.1　FPGA 的结构

在 21 世纪初，FPGA 器件系列拥有诸多对实现 DSP 算法非常有利的功能，这些 FPGA 器件具有快速进位逻辑功能，能够以超过 300MHz 的速度实现 32 位(非流水线)的加法器[1、22、23]，还具有嵌入式 18 位×18 位乘法器和大容量存储器。

Xilinx FPGA 以早期的 XC4000 系列的基本逻辑模块为基础，最新派生的系列分别命名为 Spartan(成本低)和 Virtex(性能高)。Altera 器件基于 FLEX 10K 逻辑模块，最新派生的系列分别命名为 Stratix(性能高)和 Cyclone(成本低)。Xilinx 器件具有 FPGA 典型的宽泛布

线层次，而 Altera 器件则是基于 Altera 的 CPLD 中使用的宽总线体系结构，但 Cyclone 和 Stratix 器件的基本模块已经不再是 CPLD 中大规模的 PLA，取而代之的是 FPGA 中常见的中等颗粒度器件，即小型查询表(Look-Up Table, LUT)。Altera 将这些 LUT 称为逻辑元件(Logic Element，LE)，多个 LE 组合起来就成了逻辑阵列模块(Logic Array Block，LAB)。LAB 中 LE 的数量随系列型号而异，通常新系列中每个 LAB 内包含更多的 LE：Flex10K 的每个 LAB 利用 8 个 LE，APEX20K 的每个 LAB 利用 10 个 LE，而 Cyclone II 的每个 LAB 利用 16 个 LE。

Spartan-6 器件 XC6SLX45 是图 1-10(b)所示的 Digilent 公司提供的 Atlys DSP 开发板的一部分，接下来将仔细研究该 FPGA 系列。Xilinx Spartan-6 的基本逻辑元件也称为切片，有三种不同的版本：M、L 和 X。

在 Spartan-6 系列中，2 个切片组合起来构成一个可配置逻辑模块(Configurable Logic Block, CLB)，共计 8 个 6 输入 1 输出 LUT(或 16 个 5 输入 LUT)和 16 个触发器，256 位分布式 RAM，128 位移位寄存器。在所有切片中，25%为 M 型，并且具有所有这些特征；25%为 L 型，没有内存移位寄存器功能；50%为 X 型，没有移位寄存器选项、算术进位和宽多路复用。图 1-11 显示了每种切片的 1/4 以及各自特征。M 型切片中的每个 LUT 都可以用作 64×1 的 RAM 或 ROM。Xilinx 器件具有多层布线，涵盖从 CLB 间的短连接线到跨过整块芯片的长连接线。Spartan-6 还具有可用作单端或双端 RAM 或 ROM 的大规模存储器(共 18 432 位，若不使用奇偶校验位，则是 16 384 位)。存储器可以配置成 2^9×32、2^{10}×16、…、2^{14}×1 等，每增加一个地址位，数据位长度就缩短二分之一。Spartan-6 系列适用于 DSP 的另一个突出功能是其拥有嵌入式乘法器，这些乘法器是快速 18 位×18 位有符号阵列乘法器。也可进行 17 位×17 位的无符号乘法。该系列具有 4 个完整的时钟网络(DCM)，使得在同一个 FPGA 内能够同时运行多种不同时钟频率的设计，而且时钟脉冲相位差很小。Spartan-6 编程需要多达 33Mbit 的配置文件。表 1-8 给出了 Xilinx Spartan-6 系列器件最重要的 DSP 功能。

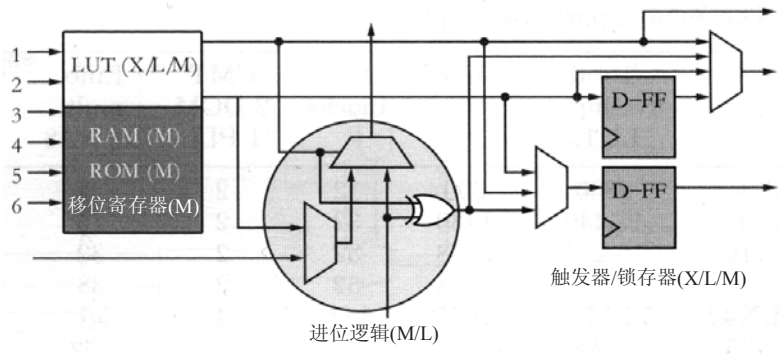


图 1-11　1/4 部分的 Spartan-6 切片。X 切片只有 LUT 与两个触发器。L 切片增加了快速进位逻辑。M 切片具备所有特征(© Xilinx)

表 1-8　Xilinx Spartan-6 系列

器　　件	5 输入 LUT	切　　片	RAM 模块	CMT 2DCM 1PLL	嵌入式 18 位×18 位乘法器	配 置 文 件 (Mbit)
XC6SLX4	4800	600	12	2	8	2.7
XC6SLX9	11440	1430	32	2	16	2.7
XC6SLX16	18224	2278	32	2	32	3.7
XC6SLX25	30064	3758	52	2	38	6.4
XC6SLX45	**54576**	**6822**	**116**	**4**	**58**	**11.9**
XC6SLX75	93296	11662	172	6	132	19.7
XC6SLX100	126576	15822	180	6	180	16.7
XC6SLX150	184304	23038	180	6	180	33.9

接下来看一下 Altera 系列中用于低成本原型开发板 DE2-115 的 Cyclone IV E 器件 EP4CE115，如图 1-10(a)所示。Altera Cyclone IV 器件的基本模块采用小规模 LUT，达到中等颗粒度水平。Cyclone 器件类似于 UP2 和 UP3 开发板上常用的 Altera 10K 器件，随着 RAM 模块的存储规模增加到 9Kbit，既不再像 Flex 10K 中称为 EAB，也不像 APEX 系列中称为 ESB，而称为 M9K 存储器，这一命名更好地反映了其容量。Altera FPGA 中的基本逻辑元件称为逻辑元件(Logic Element, LE)[3]，包括一个触发器、一个 4 输入 1 输出的 LUT，或一个 3 输入 1 输出的 LUT 和一个快速进位，或“与/非”乘积项扩展电路，如图 1-12 所示。每个 LE 在正常模式下作为一个 4 输入 LUT，或是在算法模式下用作带有一个额外快速进位的 3 输入 LUT。Cyclone IV 器件中的 16 个 LE 组成一个逻辑阵列模块。每行至少包括一个 18 位×18 位乘法器和一个 M9K 存储器。18 位×18 位乘法器可用作两个有符号 9 位×9 位乘法器。M9K 存储器可配置成 2^8×32、2^9×16、…、8192×1 的 RAM 或 ROM。此外每个字节还有一位奇偶校验位(如 256×36 配置方式)可用于数据完整性检查。M9K 和 LAB 通过高速宽总线连接起来，如图 1-13 所示。在同一器件内采用多个 PLL 产生时钟脉冲相位差很低的多个时钟域。对 EP4CE115 器件进行编程至少需要 29Mbit 大小的配置文件。表 1-9 给出了 Altera Cyclone IV 系列的部分器件。

如果将这两种分别来自 Altera 和 Xilinx 的路由策略加以比较，就会发现这两种方法都很有价值：Xilinx 的方法倾向于使用较多的局部路由资源而较少使用全局路由资源，这对 DSP 的使用有促进作用，因为绝大部分数字信号处理算法都在局部处理数据。具有宽总线的 Altera 方法也有其价值，因为典型的操作不是在“位切片”(bit slice)操作中逐位处理，更常见的是必须把 16 至 32 位宽的数据矢量转移到下一个 DSP 模块中。

3. 在设计报告文件中有时也称为逻辑单元(LC)。

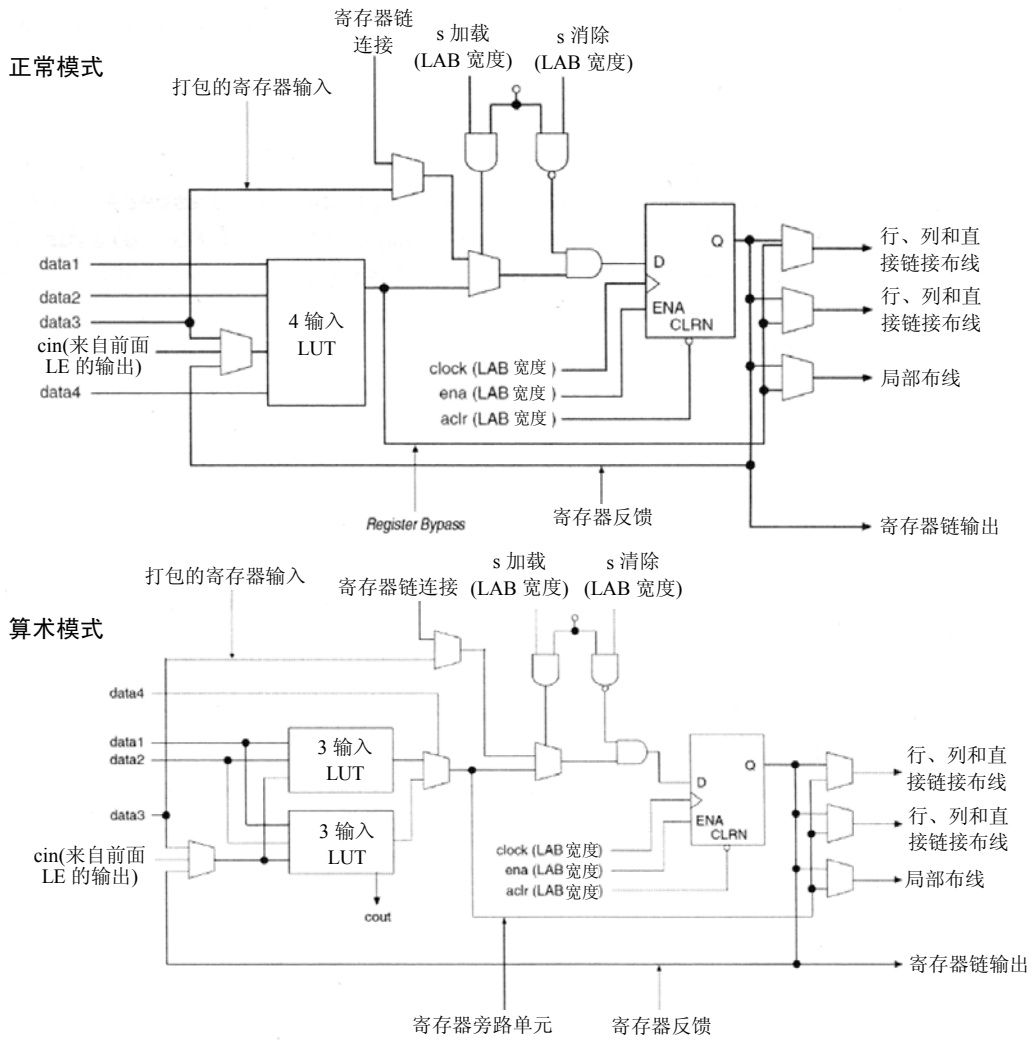


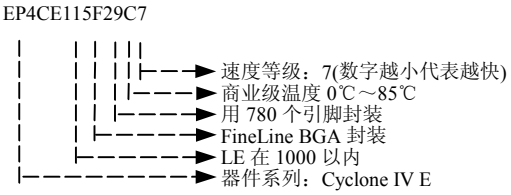
图 1-12　Cyclone IV 的逻辑单元(© Altera[24])

表 1-9　Altera Cyclone IV E 系列器件

器　　件	4 输入 LUT	存储器 M9K	PLL/时钟网络	嵌入式乘法器 18 位×18 位	最大 I/O	配置文件 (Mbit)
EP4CE6	6272	30	2/10	15	179	2.94
EP4CE10	10 320	46	2/10	23	179	2.94
EP4CE15	15 408	56	4/20	56	343	4.09
EP4CE22	22 320	66	4/20	66	153	5.75
EP4CE30	28 848	66	4/20	66	532	9.53
EP4CE40	39 600	126	4/20	116	532	9.53
EP4CE55	55 856	260	4/20	154	374	14.89
EP4CE 75	75 408	305	4/20	200	426	19.97
EP4CE115	**114 480**	**432**	**4/20**	**266**	**528**	**28.57**

1.4.2　Altera EP4CE115F29C7

本书从头到尾使用的器件是由 Altera 的大学项目提供的、作为 DSP 原型开发板 DE2-115 一部分的 Altera EP4CE115F29C7 器件，该器件属于 Cyclone IV E 系列。对该器件命名法的解释如下：



只要在可能的情况下，具体设计示例用到 Cyclone IV 器件 EP4CE115F29C7 时，都使用 Altera 提供的软件。本书学习资料提供的 Quartus II 软件是一个完全集成的系统，包括 VHDL 编辑器、Verilog 编辑器、定时估算和位流发生器。该软件可以从 www.altera.com 免费下载。Web 版的唯一局限性是不能适用于每个器件的所有引脚引出线。鉴于所有示例在 VHDL 和 Verilog 下都是可行的，因此可能还会需要任何 12.1 以外的版本或其他的仿真器。例如，Xilinx ISE 编译器以及 ISIM 仿真器就可以成功地应用于编译本书的示例。对于其他的 Quartus II 软件版本，所包括的 qvhdl.tcl 与 qv.tcl 可只使用一个脚本编译新软件版本的所有实例。

1. 逻辑资源

EP4CE115 是 Altera Cyclone IV 系列的一员，其逻辑密度大约相当于 115 000 个逻辑单元。此外还有 266 个 18 位×18 位乘法器可用(如果用作 9 位×9 位，数量还可以翻一倍)，如图 1-14 所示。从表 1-9 可以看到，EP4CE115 器件具有 114 480 个基本逻辑单元(Logic Element，LE)。这也是可以实现的全加法器的最大数目。每个 LE 都可以用作一个 4 输入 LUT，或在“算法”模式中用作带有一个额外快速进位的 3 输入 LUT，如图 1-12 所示。16 个 LE 组成一个逻辑阵列模块(Logic Array Block, LAB)，如图 1-13 所示。LAB 的数目就是 114 480/16 = 7155。在器件左侧中间区域布置了 JTAG 接口，使用了 5×9=45 个 LAB。这就是为什么 LAB 的总数不等于行列乘积的原因。该器件还包括 6 列 9-Kbit 的存储器(称作 M9K 存储器)，列宽为一个 LAB，这样 M9K 的总数就是 6×72 = 432。M9K 可以配置为 256×36、256×32、512×18、512×16、…、8192×1 的 RAM 或 ROM，其中每个字节有一位奇偶校验位可用。EP4CE115 中还有一列 18 位×18 位的快速阵列乘法器，也可以配置成两个 9 位×9 位乘法器。图 1-19 给出了 EP4CE115 的左下角以及显示器件内部的整体布局的鸟瞰图。

2. 额外的资源和路由

在数据表中，EP4CE115 的水平和垂直总线的确切数目不再像过去的(例如 FLEX10K)系列进行指定[25]。从 Compiler Report 中，我们可以判断设备的整体路由资源，见 Fitter→Resource Section→Logic and Routing Section。局部连接通过互联模块和直接链路提供。

Cyclone IV 器件的每个 LE 能够通过快速局部与直接链接互联驱动多达 48 个 LE。下一层布线是 R4 和 C4 的快速行列局部连接，可以到达±4 个 LAB 距离内的 LAB，或 3 个 LAB 和一个嵌入式乘法器或 M4K 存储模块。可用的最长连接是 R24 和 C16，它们分别允许连接 24 行和 16 列 LAB，构成了基本横跨整个芯片的连接。如果源逻辑阵列模块和目的逻辑阵列模块既不在同一行也不在同一列，还可以采用任意行列连接的组合。全局时钟网络跨越整个 FPGA。表 1-10 给出了 EP4CE115 中可用路由资源的概述。

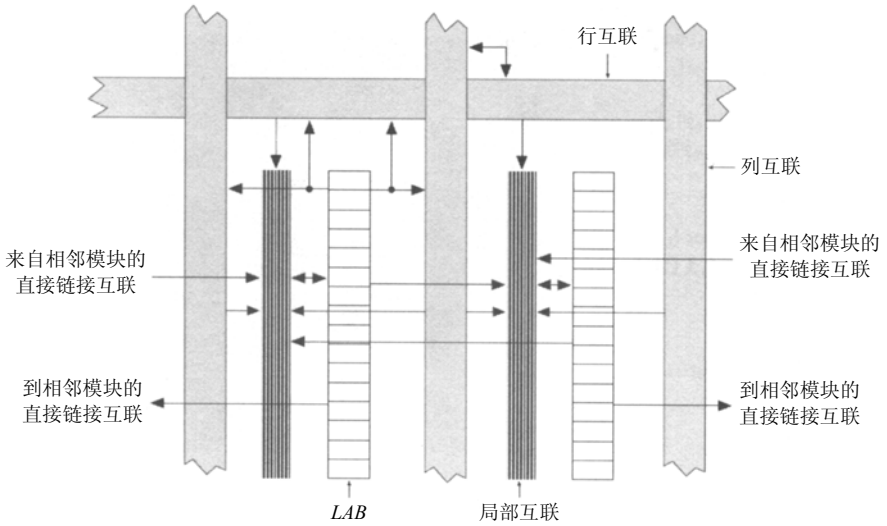


图 1-13　Cyclone IV 逻辑阵列模块资源(© Altera[24])

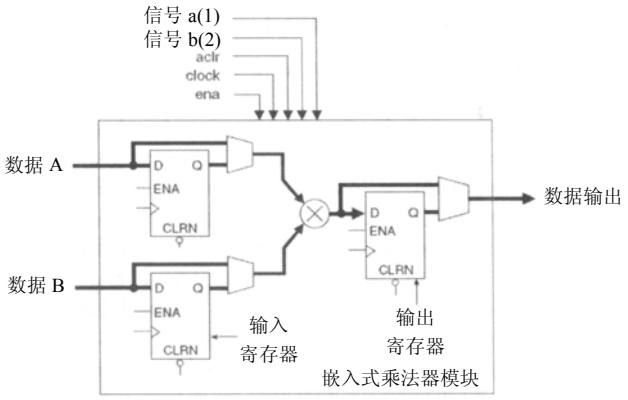


图 1-14　嵌入式乘法器的体系结构(© Altera[24])

表 1-10　Cyclone IV 系列器件 EP4CE115 的路由资源

模块	直接	C4	R4	C16	R24	时钟
342 891	342 891	209 544	289 782	10 120	9963	20

数据表提供有关可用的内部逻辑阵列模块控制信号和全局时钟的更多细节。

EP4CE115 有 4 个 PLL，共有 20 个时钟网络覆盖了整个芯片，以保证短时钟偏移。每个 PLL 有一个可用来产生不同频率或相位偏移的 5 个输出计数器。比如在 DE2 板上 DRAM 需要 0 和−3 纳秒的时钟偏移时，可用一个 PLL 来产生。

所有 16 个 LE 在 LAB 中共享相同的同步清零和负载信号。两个异步清零、两个时钟以及两个使能信号由 LE 在 LAB 中共享。但由于 DSP 信号通常在宽总线信号中处理，因而将限制控制信号的自由度在一个 LAB 内。

LAB 本地互联由行或列总线信号或 LE 在 LAB 中进行驱动。邻近的 LAB、PLL、BRAM、左/右嵌入式乘法器还可以驱动本地 LAB。在下一节你将会了解到在这些不同连接的组合上的延迟变化范围很大，合成工具总是尽量将逻辑单元彼此之间布置得更近，将内部连接延迟最小化。例如，一个 32 位加法器最好布置在相邻两行的两个 LAB 上，一个在另一个上面，如图 1-19 所示。

3. 定时估算

过去，Altera 的 Quartus II 与 Xilinx ISE 软件只有两种定时估算目标：面积或速度。然而，随着增加的器件密度与带多时钟域的片上系统设计、不同的 I/O 时钟模式、多路复用器时钟以及时钟分频器，这种完整器件的面积或速度优化策略可能不是一个好方法了。Altera 公司现在提供了更复杂的时序规范，它基于概要设计约束(Synopsis Design Constrain，SDC)文件，看起来更类似于基于单元的 ASIC 设计风格。对于相同的电路而言，一个合成工具可以有不同的库元素，如脉动进位、进位保存或加法器快速前瞻类型。首先，该工具优化设计，以满足规定的时序约束；然后，优化面积。如果想要通过以往的工具(如最小面积或最高速度法)达到类似的合成结果，可以通过过度或欠约束实现这一目标。举例来说，如果不指定 SDC 文件，然后假定有 1GHz 的目标性能，这最有可能过度约束您的设计和设备。如果只想优化面积，那么应该用较低的目标时钟速率，将所有的工作放在面积优化上。基于 SDC 规范的流程近期已经推出，查阅教程有助于更好地理解。Altera 大学项目在“使用 TimeQuest 时序分析器”教程中进行了介绍。更多细节详见“Quartus II TimeQuest Analyzer Cookbook”，里面有许多不同的多时钟域的例子。

在设计例子中，仅对优化速度感兴趣时，可使用默认设置(例如没有 SDC 文件)，在期望的 1GHz 时钟速率下运行过约束设计。你应该会在编译报告中看到一条警告信息，即定时未得到满足，这是可预见的，因而不必考虑。

为达到最佳性能，有必要从物理上了解软件如何实现设计。因此对解决方案进行一下大致评估还是有益的，然后再决定如何改进设计。

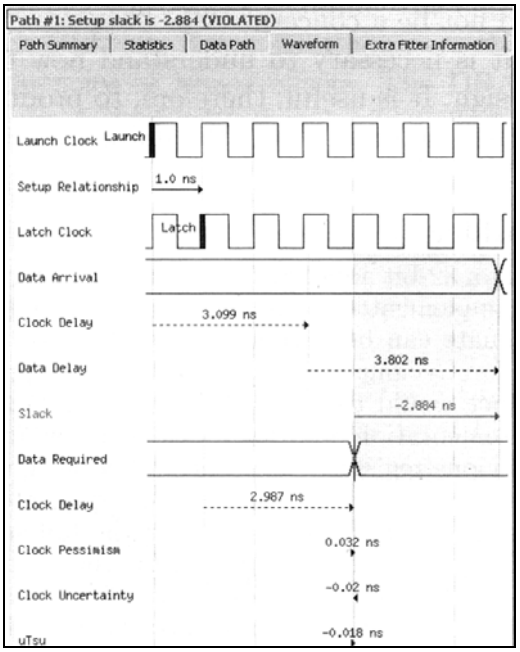
例 1.2　32 位加法器的速度

假定设计人员需要实现一个 32 位加法器并估算其设计的最大速度。加法器可以在两个 LAB 中实现，每个都需要使用快速进位链。大致的初步估算可以使用进位输入到进位输出的延迟，对于 Cyclone IV 第 7 个速度等级的芯片，这一时间为 66ps(微微秒)。最佳性能的上限值就是 32×66ps = 2.112ns(纳秒)，也就是 473MHz。但在实际的实现中还会出现额外的时延：首先，从前级寄存器到第一个全加器的互联时延是 0.501ns。接下来必然出

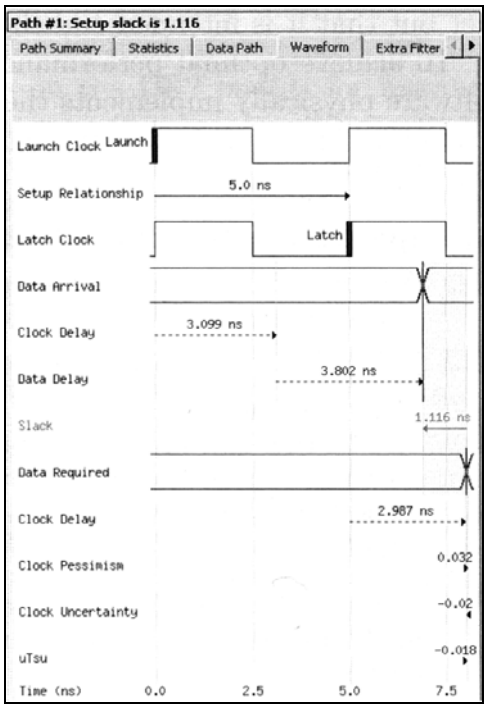
现第一次进位时间 t_{cgen}，大约为 0.414ns。最后，在进位链的终端需要计算完整的和位(大约 536ps)，而且输出寄存器的设置时间(87ps)也需要考虑进来。计算结果存储在 LE 寄存器中。下面总结了这些时间数据：

LE 寄存器时钟到输出的时延	t_{co}	=	232ps
互联时延	t_{ic}	=	501ps
数据输入到进位输出的时延	t_{cgen}	=	414ps
进位输入到进位输出的时延	$30\times t_{cico}$	=	$30\times 66ps=1980ps$
LE 查阅表时延	t_{LUT}	=	536ps
LE 寄存器设置时间	t_{su}	=	87ps
总计		=	3802ps

对于可能出现的时钟偏移，应加上额外的 82ps，因而总估算的时延是 3884ps，或说速率是 257.47MHz。这一设计对加法器而言预计大约需要使用 32 个 LE，还需要额外的 2×32 个 LE，用于在寄存器中存储输入的数据(请参阅练习 1.7)。在图 1-15 中，TimeQuest Analyzer 显示包括时钟延迟的完整时序路径。由于未指定所需的时钟周期，TimeQuest 将使用默认的 1ns 或 1GHz，这在大多数情况下为过度约束，Slack 为负值(见图 1-15(a)第 7 行)，以红色显示，视为违规。为了满足时序，通常会采用默认的 1ns，并添加(绝对)负松弛。如果用一个 Synopsys Design Constrain 文件指定一个时序裕量的定时为 200MHz(即时钟周期为 5ns)，分析将表明，正时序裕量满足时序要求(以绿色显示)，见图 1-15(b)。



(a) 负时序裕量显示时序违规



(b) 正时序裕量表明满足时序要求

图 1-15 TimeQuest 分析

如果所使用的两个 LAB 不放置在同一列并且彼此相邻，那么还会出现额外的时延。如果信号直接来自 I/O 引脚，那么延迟时间还要更长。对于数据来自 I/O 引脚的情况，Quartus II 的 TimeQuest Analyzer 给出的 DE2_115 开发板上的从 SW[1]输入到引脚 HEX[1]输出的直接 FPGA 连接传输时延为 11.3ns，远大于测试时数据直接从寄存器到本设计所需的时间。数据表[24]通常报告的最佳性能是在放置测试设计的 I/O 数据的寄存器靠近设计单元的前提下达到的。乘法器和 M9K 模块(但不是加法器)还有额外的 I/O 寄存器以实现最大速度，如图 1-14 所示。但这些额外的 I/O 寄存器在 LE 资源估算时通常不予考虑，因为通常假定前面的处理单元对于所有数据使用一个输出寄存器。当然，因为情况也不总是如此，所以需要将加法器设计方案使用的额外寄存器都放在圆括号内。表 1-11 给出了这些假定前提下的部分典型测量结果。如果将这些测量数据与数据手册[24]中给出的时延加以比较，就会注意到，对于某些模块，TimeQuest Analyzer 将上限频率限制到比数据手册中给出的具体延迟范围小一些。这是一种保守并且更安全的估算——实际上设计运行在较高一点儿速度也不会出现错误。

表 1-11　Cyclone IV C7 的部分典型时序电路性能 Fmax 和资源数据

设　　计	LE	M9K 存储器	嵌入式乘法器 9 位×9 位	时序电路性能(MHz)
16 位加法器	16(+32)	—	—	363
32 位加法器	32(+64)	—	—	257
64 位加法器	64(+128)	—	—	169
ROM 2^8×36	47	1	—	274
RAM 2^8×36	—	1	—	274
9 位×9 位乘法器	—	—	1	300
18 位×18 位乘法器	—	—	2	300

4. 功耗

FPGA 的功耗可以说是一个关键的设计约束条件，特别是对于移动应用。因此在本例中推荐使用 3.3V 或更低电压级别电子工艺的器件。本例中的 Cyclone IV 系列是中国台湾 ASIC 生产厂家 TSMC 采用 2.5V、60nm 的低 k 电介质工艺制造的，但也支持 3.3V、2.5V、1.8V、1.5V 和 1.2V 的 I/O 接口电压。为了估算 Altera 器件 EP4CE115 的功耗，必须考虑两个功耗来源，分别是：

1) 静态功耗，对于 EP4CE115F29C7，$P_{static} \approx 135mW$

2) 动态(逻辑、乘法器、RAM、PLL、时钟、I/O)功耗 I_{active}

第一个参数与设计无关，而且 CMOS 技术中产生的待机功耗非常少。有效电流主要与时钟频率和使用的 LE 数目或其他资源有关。Altera 提供了一张 Excel 工作表，名为 PowerPlay Early Power Estimator，在项目的早期阶段就获取给出功耗(如电池寿命)和可能的冷却要求的建议。

对于 LE，还可以根据下面的经验公式估算动态功耗：

$$P \approx I_{\text{dynamic}} V_{CC} = K \times f_{\max} \times N \times \tau_{\text{LE}} V_{CC} \tag{1-1}$$

其中 K 是常数，$f_{\max}$ 是按 MHz 计算的工作频率，N 是器件中使用的所有逻辑单元的总数，τ_{LE} 是逻辑单元在每个时钟周期内触发的平均百分比(典型值是 12.5%)。表 1-12 给出了当设计人员用到 EP4CE115F29C7 的所有资源且系统时钟为 100MHz 时的功耗估算结果。对于资源使用较少或系统时钟较低的情况，就要调整公式(1-1)中的数据。例如，如果系统时钟从 100MHz 降低到 10MHz，功耗就降低到 159+2079/10=366.9mW，静态功耗就占到 43%。

表 1-12 Cyclone IV EP4CE115F29C7 的功耗估算

参　　数	单　　元	触发百分比(%)	功耗(mW)
P_{static}			159
LE 114 480 在 100MHz 下	114 480	12.5%	1170
M9K 模块存储器	432	50%	74
9 位×9 位乘法器	532	12.5%	153
I/O 单元(2.5V,4mA)	528	12.5%	164
PLL	4		32
时钟网络	115 706		486
总计			2238

尽管 PowerPlay Estimation 在项目计划阶段是很有用的工具，但在精度方面有其局限性，因为设计人员必须规定触发的速率。例如在图 1-16 频率合成设计的示例中，情况就会变得更复杂。RAM 模块估算为 50%的触发率也许是准确的，但很难确定累加器部分 LE 的触发率，因为 LSB 的触发频率比 MSB 高得多，累加器生成一个三角形输出函数。要想获得更精确的功耗估算，可以使用 Altera 的 Processing 菜单中的 PowerPlay Power Analyzer Tool 选项。Analyzer 工具允许设计人员从仿真的输出读取计算的触发数据。仿真器生成一个“Signal Activity File”或“Value Change Dump”文件，该文件可以作为 Analyzer 工具的输入文件。PowerPlay Power Analyzer 工具允许选择不同的选项。在没有模拟器的帮助下，可以通过 TimeQuest Timing Analyzer SDC 文件指定一个时钟频率，并设置一个默认的切换速度。如果使用功能性或 RTL 仿真作为功率估计，只需要设计一个完整的编译输入，并且将会更加准确。如果在包括精确 LE 切换、故障、总线驱动数据等的 MODELSIM 仿真输入中使用编译设计，将会得到最佳估计。表 1-13 显示了功耗估算和三个功率分析器之间的对比。

表 1-13　图 1-16 中采用 Cyclone IV EP4CE115F29C7 的设计功耗 50MHz，使用 SDC TimeQuest 文件与基于 Excel 的 Power-Play 早期功耗估算器(PowerPlay Early Power Estimator，PPEPE)或基于 Quartus 的 Power-Play 功耗分析器(PowerPlay Power Analyzer，PPPA)，估算 12.5%触发率下的功耗

	PPEPE 估算器	PPPA 估算器	PPPA RTL 仿真器	PPPA 定时
所需 VCD	-	-	√	√
参数	功率/mW	功率/mW	功率/mW	功率/mW
静态	135	98.4	98.4	98.5
动态	2	2.3	2.6	3.7
I/O	4	38.9	38.9	50.4
总计	141	139.6	139.9	152.6

可以看到在估算结果与分析结果之间有 10%的偏差。虽然该分析需要包括测试平台在内的一个完整设计，但在项目的早期阶段就可以进行估算。

后续章节的示例将用到下面的案例研究，该案例研究也为以后的自学问题提供了一个详细的图解。

1.4.3　案例研究：频率合成器

本案例研究的设计目标是基于 Philips PM5190(大约 1979 年，请参阅图 1-16)模式实现一个经典的频率合成器。该频率合成器包括一个 32 位的累加器，有 8 个最高有效位(Most Significant Bit, MSB)连接到一个 SIN-ROM 的查阅表(LUT)上，从而产生所需的输出波形。等效的 HDL 文本文件 fun_text.vhd 和 fun_text.v 采用行为级 HDL 代码的方式实现该设计，从而避免 LPM 组件的实例化。该设计的挑战是必须由 Altera Quartus II 与 Xilinx ISE 软件合成行为级 HDL 代码，且与推荐的模拟器(Altera 的 ModelSim 和 Xilinx 的 ISIM)在 RTL 和时序仿真方面相匹配。接下来演示一下使用 Quartus II 实现某个设计通常需要执行的所有步骤：

(1) 设计的编译
(2) 设计结果和平面布置图
(3) 设计的仿真
(4) 性能评估

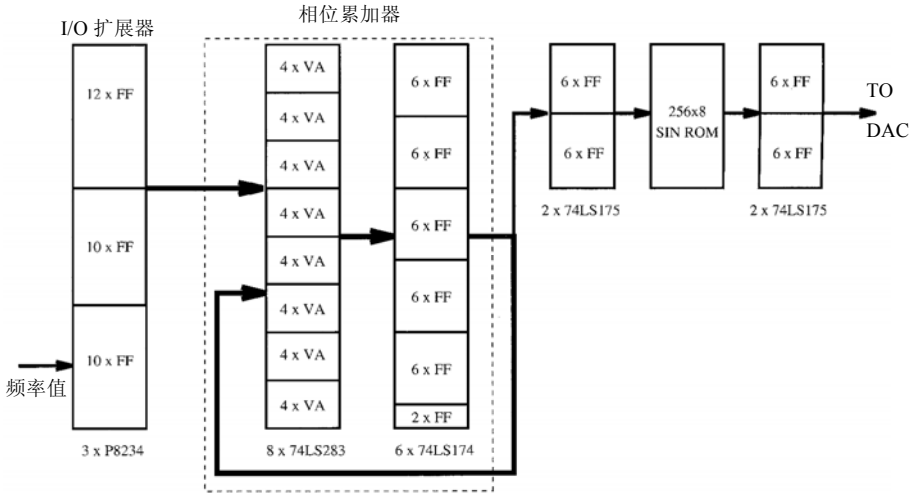


图 1-16　PM5190 频率合成器

1. 设计的编译

如果还没有项目文件，要检查和编译文件，首先要启动 Quartus II 软件，选择 File | Open Project 命令或启动 File | New Project Wizard 命令。在项目向导中指定想要使用的项目目录并将项目名称和顶层设计命名为 fun_text。然后单击 Next 按钮指定想要添加的 HDL 文件，在这个示例中是 fun_text.vhd。再次单击 Next 按钮，从 Cyclone IV 系列中选择 EP4CE115F29C7 (Quartus12.1 器件列表中的倒数第 5 个)，单击 Next 按钮，选择 MODELSIM-ALTERA 作为仿真工具，并按 Finish 按钮。如果使用本书学习资料里的项目文件 fun_text.qsf，就已经有正确的文件和器件说明。选择 File | Open 命令加载 HDL 文件。VHDL[4]设计如下所示：

```
--  A 32 bit function generator using accumulator and ROM
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.ALL;
USE ieee.STD_LOGIC_arith.ALL;
USE ieee.STD_LOGIC_signed.ALL;
-- ------------------------------------------------------
ENTITY fun_text IS
  GENERIC ( WIDTH   : INTEGER := 32);    -- Bit width
  PORT (clk   : IN  STD_LOGIC; -- System clock
        reset : IN  STD_LOGIC; -- Asynchronous reset
        M     : IN  STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
                                  -- Accumulator increment
        acc   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                                   -- Accumulator MSBs
        sin   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END fun_text;                          -- System sine output
-- ------------------------------------------------------
ARCHITECTURE fpga OF fun_text IS
```

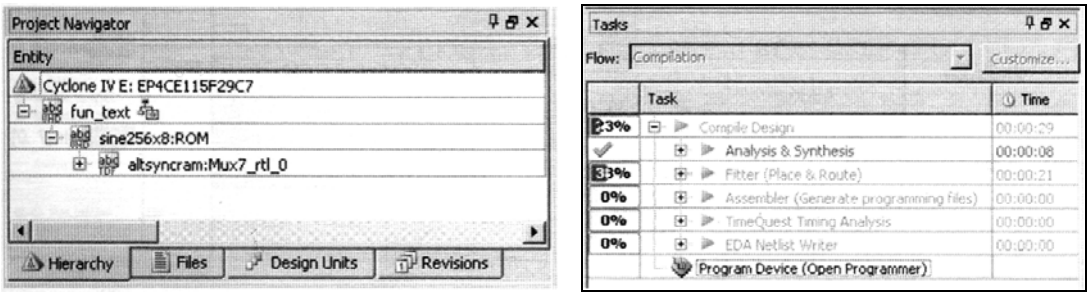
4. 这一示例相应的 Verilog 代码文件 fun_text.v 可以在附录 A 中找到，附录 B 给出了合成结果。

```
  COMPONENT sine256x8
    PORT (clk : IN STD_LOGIC;
          addr : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          data : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
  END COMPONENT;

  SIGNAL acc32 : STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
  SIGNAL msbs  : STD_LOGIC_VECTOR(7 DOWNTO 0);
                                        -- Auxiliary vectors
BEGIN

  PROCESS (reset, clk, acc32)
  BEGIN
    IF reset = '1' THEN
      acc32 <= (OTHERS => '0');
    ELSIF rising_edge(clk) THEN
      acc32 <= acc32 + M; -- Add M to acc32 and

   END IF;               -- store in register

  END PROCESS;

  msbs <= acc32(31 DOWNTO 24); -- Select MSBs
  acc <= msbs;

  -- Instantiate the ROM
  ROM: sine256x8 PORT MAP
                 (clk => clk, addr => msbs, data => sin);

END fpga;
```

在代码开头部分的 LIBRARY 对象中包括预定义模块和定义。ENTITY 模块指定了器件的 I/O 接口和泛型变量。在编码中接下来的是应用组件和附加的信号定义。HDL 编码起始于关键字 BEGIN 之后。所述第一个 PROCESS 包括一个 32 位累加器，即加法器后接寄存器。累加器有一个异步高电平有效复位。接下来的两条语句连接本地信号到 I/O 端口。最后，在设计中 ROM 表被实例化为组件，组件的端口信号被连接到局部信号。想要查询 ROM 表设计文件 sine256x8.vhd，可以通过加载文件，或者在 Project Navigator 窗口(左上)中双击它，如图 1-17(a)所示。通常，一个带有在内存中加载初始数据的综合 ROM 或 RAM 设计不是一个简单的任务。可以在 VHDL 子集——VHDL 1076.6-2004 中查看综合代码，或者简单一点——查阅由工具供应商提供的语法模板(Alter：Edit→Insert Template→VHDL→Full Designs→RAMs and ROMs→Dual-Port ROM 或者 Xilinx：Edit→Insert Template→VHDL→Synthesis Constructs→Coding Examples→ROM→Example Code)。Altera 公司推荐使用一个函数调用用于初始化；而 Xilinx 采用 CONSTANT 阵列初始定义。事实证明，后者使用工具以及模拟器进行功能和时序仿真效果很好，将成为本书后续介绍的内容的首选方法。VHDL 1076.6-2004 中定义的综合属性目前没有任何厂商支持。在 VERILOG RAM 和 ROM 中，初始化已在语法参考手册(Language Reference Manual，LRM)中进行规范，最可靠的方法是结合一条初始声明语句来使用$readmemh()函数。

要为速度优化设计，选择 Assignment→Settings→Analysis & Synthesis Settings。在

Optimization Technique 中单击 Speed。时序要求可以通过使用 SDC 文件设置。默认速度设置为 1ns，该设置在 Speed 优化中通常运行良好。启动 Processing 菜单下的编译器工具(右箭头符号)。HDL 窗口左边的窗格显示了编译进度，如图 1-17(b)所示。可以看到所有编译的步骤，分别是：Analysis & Synthesis、Fitter、Assembler、Timing Analysis、Netlist Writer 以及 Program Device。或者，也可以通过单击 Processing→Start→Start Analysis & Synthesis 或使用 Ctrl+K 快捷键启动 Analysis & Synthesis。编译器检查基本语法错误，并生成一个报告文件，其中列出设计的资源估算。在语法检查通过后就可以单击编译器工具窗口左下角的 Start 按钮开始编译，也可以使用 Ctrl+L 快捷键。如果所有编译步骤均顺利通过，该设计就完整实现了。在编译器窗口中单击 Compilation Report 按钮就会得到一份流程摘要报告，报告显示应该使用 32 个 LE 和 2048 个存储器位。检查存储器初始化文件——VHDL 的 sine256x8.vhd 和 Verilog 的 sine256x8.txt。这些文件由本书学习资料里的 util 目录下的 sine.exe 程序生成。图 1-17(b)总结了汇编中的所有处理步骤，如 Quartus II 编译器窗口中所示。



(a) 项目导航器　　(b) Quartus II 中的编译步骤

图 1-17　处理步骤

对于 HDL 设计描述所需电路的图形验证，我们可以使用 Altera Quartus II 软件的 RTL 查看器。fun_text.vhd 电路的结果如图 1-18 所示。要启动 RTL 查看器，请单击 Tools→Netlist Viewer→RTL Viewer。另一个名为 Technology Map Viewer 的网表查看器提供了电路如何映射到 FPGA 资源的精确图像。然而，即使像函数发生器这样的小型设计，为了验证其 HDL 代码，技术图也提供了太多我们不会在设计研究中用到的有用细节。

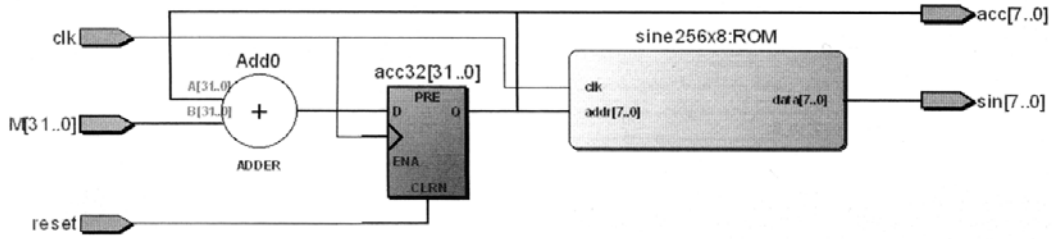


图 1-18　频率合成器的 RTL 视图

2. 平面布置图

通过单击第 6 个按钮(也就是 Chip Planner 或打开 Tool | Chip Planner 命令)获得关于芯片布局更详细的信息。Chip Editor 视图如图 1-19 所示。单击 Zoom in 按钮(也就是±放大

镜)得到图 1-19 所示的屏幕。放大以不同颜色突出显示的 LAB 和 M9K 所在的区域，就会看到用于累加的两个 LAB 以蓝色突出显示，M9K 模块以绿色突出显示。此外，几个 I/O 单元也以褐色突出显示。单击位于左侧菜单按钮区的 Bird's Eye View 按钮[5]，用滑块选择图像显示器，就会看到 Chip Editor 中的蓝色线条标明了从累加器的第一位运行到最后一位的最长路径。接下来单击左侧菜单按钮中的 Bird's Eye View 按钮，会弹出另一个窗口。还可以试验一下连接显示。现在，首先选中 M4K 模块，单击几次 Generate Fan-In Connections 按钮或 Generate Fan-Out Connections 按钮，就会显示越来越多的连接。

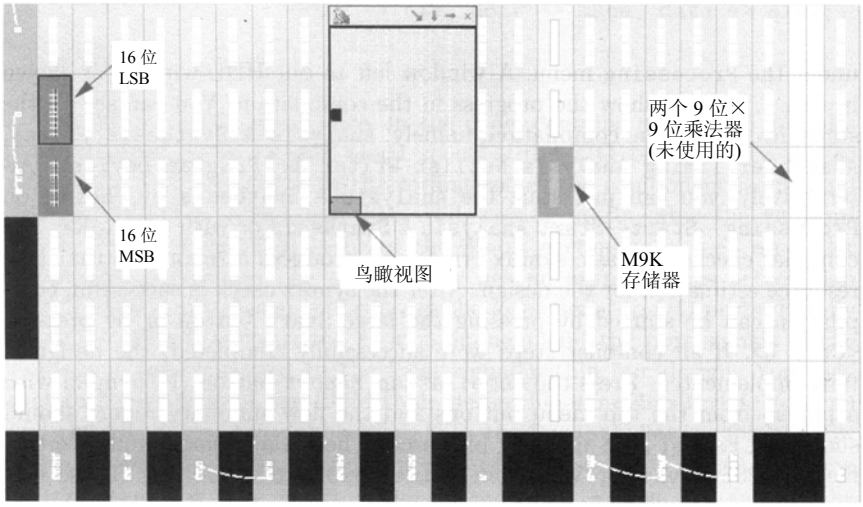


图 1-19　频率合成器设计的平面布置图

3. 仿真

在仿真方面，近几年两个 FPGA 市场的领导者采取相反的方向。过去，Altera 青睐内部 VWF 模拟器(一直到 Quartus II 9.1 版)，现在推荐外部 ModelSim-Altera 或 Qsim。另一方面，Xilinx 从 12.3 版(2010 年底)起，不再提供免费 ModelSim 模拟器，而是提供了一个集成在 ISE 工具内的免费嵌入式 ISIM 模拟器。

当模拟一个设计时，例如 ISIM 模拟器，可以从 TCL 脚本模拟器到 MODELSIM DO 文件中选择使用一个激励文件，或者我们也可以用 HDL 写一个测试平台。测试平台是一个短 HDL 文件，在这里我们实例化测试的电路，然后以如下声明：

```
clk <= NOT clk AFTER 5 ns;
```

生成和应用测试信号，产生 2×5ns=10ns 的时钟周期。然而，Xilinx 测试平台的困难来自于带定时信息(例如*_times.vhdl)的电路从功能网表直接合成，并且 STD_LOGIC 被应用于整个实体描述中。原来的 ENTITY 数据类型和 GENERIC 变量被忽略。如果想用同样的 VHDL 测试平台进行 RTL 和时序仿真，实体将被限制为单一的数据类型。更准确地说，不可使用整数、有符号或浮点数据类型、缓冲区及泛型参数。这将与编码设计重用形

5. 请注意，正如所有的 MS Windows 程序，只要将鼠标移动到按钮上(不用单击按钮)，其名称/功能即可显示出来。

成极大干扰，需要使用一个单独的测试平台进行 RTL 和时序仿真。但是，如果不使用测试平台，直接使用 TCL 激励脚本模拟电路，就可以使用相同的脚本进行 RTL 和时序仿真。此外，对于 VHDL 和 Verilog 而言，可以使用相同的激励文件，仅编译序列不同。该 ISIM TCL 脚本和 ModelSim DO 文件在简化这两个模拟器之间的过渡编码风格上非常相似。

Altera 的 Quartus II 软件有两个免费的仿真器选项。MODELSIM-Altera 允许使用来自明导公司的专业工具。第二个选择是 Altera 的 Qsim 工具，它比 MODELSIM 少一些功能(例如，没有模拟波形)，但也更容易处理，因为在分配 I/O 信号时并不需要写 HDL 测试平台或 DO 文件脚本。然而，在 12.1 版本中 Qsim 不支持 Cyclone IV 器件。因此，选择 MODELSIM-Altera 作为默认模拟器。当使用 ModelSim-Altera 的 DO 文件而不是 HDL 测试平台时，Altera 和 Xilinx 的激励文件在 VHDL 和 Verilog 之间的移动也被简化了。

在讨论一个仿真例子之前，首先看一下使用的文件名。其中*代表项目名称。对于 RTL 仿真，我们使用*.vhd 和*.v 文件。对于时序仿真，有 4 类不同的文件名：使用 Altera 工具的 VHDL 和 Verilog 的*.vho 和*.vo 以及 Xilinx 的*_timesim.vhd 和*_timesim.v。对于 RTL 和时序仿真，我们使用相同的激励文件*.do(Altera)和*.tcl(Xilinx)。

打开 ModelSim-Altera 工具进行仿真，可以看到已经加载许多预定义的库。使用 File→Change Directory 移动到包含 HDL 文件和模拟脚本的目录。使用 dir*.do 和 dir*.vhd 来验证这些文件确实是在当前路径中。为 RTL 和时序仿真分别运行脚本类型 do fun_text.do 0 和 do fun_text.do 1。对于功能仿真，应该得到类似于图 1-20 所示的结果。对于时序仿真，需要首先在 Quartus II 或 ISE 中编译这个设计，然后用定时信息模拟 VHDL 代码。该脚本编译这个文件，打开波形以及模拟电路。这个例子的“do”脚本如下所示：

```
set project_name "fun_text"
do tb_ini.do $1 sine256x8

########## Add I/O signals to wave window
add wave -divider  "Inputs:"
add wave reset clk
radix -unsigned
add wave M
add wave -divider  "Outputs:"
add wave acc sin
add wave -divider -height 80 {Analog sine:}
add wave -color Red -format Analog-Step \
         -radix unsigned -scale 0.25 sin

######### Add stimuli data
force clk 0 0 ns, 1 5 ns -r 10 ns
force reset 1 0 ns, 0 10 ns
force M 214748365 0 ns

########## Run the simulation
run 250 ns
wave zoomfull
configure wave -gridperiod 5ns
configure wave -timelineunits ns
```

仿真脚本包括 4 个典型部分：

1) 首先，定义了项目名称，并且项目中所有的组成分别通过 vcom 或 vlog 编译 VHDL 和 Verilog。在第二段脚本中，调用 tb_ini.do 的一个参数(0 为 RTL，1 为时序仿真)对顶层项目进行编译。它还包括一个添加局部信号的功能，这些局部信号在时序仿真中不可用。

2) 其次，信号按先输入后输出的指定顺序添加到波形窗口中。我们以输入信号开始，后面是输出。分频器用于帮助分化或添加额外的信息。请特别注意最后一个条目，表示的是信号的“模拟量”显示如何被定义。

3) 接下来的是刚刚定义的波形信号的任意顺序的激励数据。以 clk 定义周期信号，以 reset 和 M 定义非周期信号。

4) 最后，运行仿真，以特定时间和缩放显示完整的时间框架。网格和时间单元也可以根据波形窗口进行定义。

从脚本与波形窗口中可以看到下列数据被用到。因为周期选择了 1/100MHz = 10ns，设置 $M = 214\ 748\ 364 (M = 2^{32}/20)$，合成器的周期就是 20 个时钟周期长。注意 ROM 是以二进制偏移量编码的(也就是 0 = 128)，这种典型的编码方式被用在 D/A 转换器中。一个短 MATLAB 脚本或 C 程序可以生成这个数据。本书学习资料包括一个短 C 程序 sine.exe，它可产生该组件的 VHDL 代码以及 Verilog 代码的表格数据。为简便起见，使用十六进制显示，但 sine.exe 程序还允许二进制或八进制。

4. 性能分析

为了显示时序数据，设计的完整编译必须首先完成。对于 Altera 工具，我们默认使用 1ns 的相当于 1GHz 的时钟频率将体现在 Quartus II QSF 文件中 FMAX_REQUIREMENT“为 1ns”的 perioc 设置。由于我们的设计最有可能的运行速度较慢，这将确保 Quartus 编译器合成的设计速度有最快的可能。在下行路上，我们总是会得到一个编译警告—— Synopsys 设计约束文件“fun_test.sdc”没有被发现，以及一个严重警告——定时不符合要求，但我们可以忽略这些消息。

采用 TimeQuest 分析器的结果在图 1-21 所示的编译报告中提供。它为缓慢 85C、0C 和快速 0C 模式包括三组时序数据。我们采用最悲观的，即缓慢 85C 模式。所述的 FmaxSummary 频率是我们的电路可以运行的最大频率。有时性能得到进一步限制，缘于最大 250MHz 的 I/O 引脚速度。如果电路在一个纯粹的组合设计中，没有寄存器至寄存器通路的 Fmax 将释放出空消息：没有路径汇报。

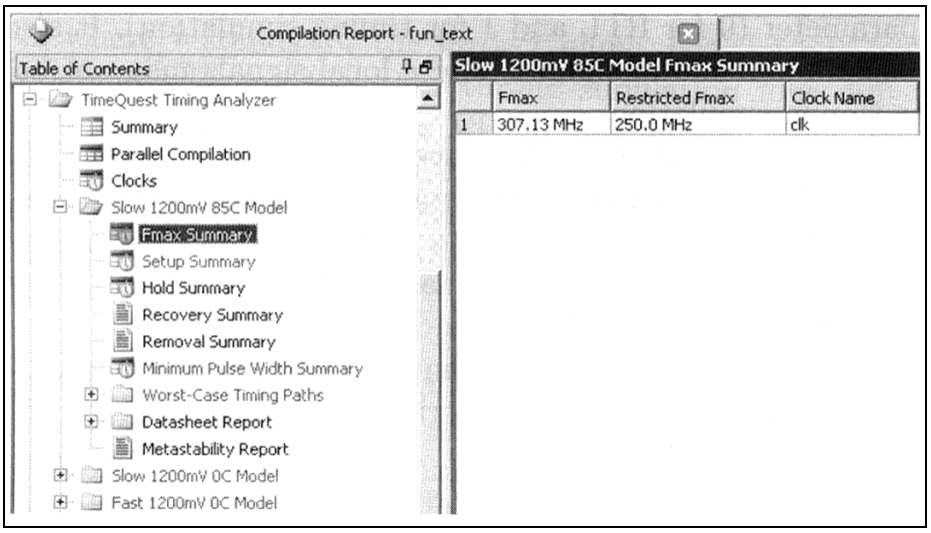


图 1-21 来自 TimeQuest 时序分析器设计的频率合成器的寄存器性能

Xilinx 软件的性能分析相对简单，因为只需要对速度与面积进行设置。在实现视图中右键单击 Synthesizer-XST 选项进入，在综合选项中选择优化目标的录入速度。在完整编译后检查 Post-PAR 静态时序报告，当单击设置目的时钟 CLK 时，会在报告中发现最大时钟作为最后的时序输入。

这样，频率合成器的案例研究就完成了。

1.4.4 用知识产权内核进行设计

尽管 FPGA 以其支持快速原型设计的功能而著称，但这只适用于 HDL 设计已经可用并且已经过充分测试的情况。复杂的模块(如 PCI 总线接口、流水线 FFT 和 FIR 滤波器或 μP 等)都需要数周甚至数月的开发时间。有一个选择可以从根本上缩短开发时间，这就是使用所谓的知识产权(Intellectual Property, IP)内核。这些内核是预先开发的(大规模)模块，其中典型的标准模块(如数控振荡器(Numeric Controlled Oscillator, NCO)、FIR 滤波器、FFT 或微处理器等)都可以从 FPGA 供应商直接获得，而更专业化的模块(例如 AES、DES 或 JPEG 编解码器、浮点库或 I2C 总线或以太网接口模块)则可以从第三方供应商那里获得。Quartus II 软件包中的一些模块是免费的，而更大更复杂的模块则价格昂贵。但只要模块能满足设计要求，使用这些预定义的IP 模块通常来说更经济有效。

下面快速浏览一下不同类型的 IP 模块，然后讨论每种类型的优缺点[26~28]。典型的 IP 模块分为三种主要形式，如下所述。

1. 软内核

软内核是对组件的行为说明，需要用 FPGA 供应商的工具来合成。模块通常以硬件描述语言(Hardware Description Language, HDL)，如 VHDL 或 Verilog 的形式提供，便于用户修改，甚至可以在合成之前为某一特定供应商或器件增删一些新功能。在底层，IP 模块还需要更多的工作来满足期望的规模、速度和功耗要求。由 FPGA 供应商提供的这种形式的

模块非常少，如 Altera 的 Nios 微处理器和 Xilinx 的 PICO blaze 微处理器。这种内核很难实现对 FPGA 供应商的知识产权保护，因为这些模块以可合成的 HDL 语言形式提供，很容易被有竞争力的 FPGA 工具/器件集或基于单元的 ASIC 使用。以 HDL 形式提供的第三方 FPGA 模块的价格通常都要比后面将要讨论的参数化内核的中等定价高很多。

2. 参数化内核

参数化和稳固的内核是对组件的结构化说明。虽然该设计的参数可以在合成前更改，但通常 HDL 不可用。Altera 和 Xilinx 提供的多数内核都是这种形式的内核。虽然这种内核非常灵活易用，但因为禁止其他 FPGA 供应商和 ASIC 厂商使用这些内核，所以与软内核相比，它们为 FPGA 供应商提供了更好的知识产权保护。Altera 和 Xilinx 可用的参数化内核例子有 NCO、FIR 滤波器编译器、FFT(并行和串行)和嵌入式处理器，如 Altera 的 Nios II。参数化内核的另一个优点是经常连接的可用资源(LE、乘法器和模块 RAM)都在百分之几以内，这就为在合成之前就快速地设计规模、速度和功耗要求提供了宽松的空间。HDL 形式的测试平台(针对 MODELSIM 仿真器)提供周期精确的建模，而且 C 语言和 MATLAB 也有针对参数化内核的行为精确建模的标准。生成代码通常只需要几秒钟。本节后面以及随后章节(第 3 章关于 FIR 滤波器和第 6 章关于 FFT)将进一步研究 NCO 参数化内核。

3. 硬内核

硬内核(固定的网表内核)是一种物理说明，以各种物理层布局格式(如 EDIF 形式)。当需要严格的实时约束时，如 PCI 总线接口，这些内核通常都针对具体的器件(系列)优化。虽然该设计的参数固定，如 16 位 256 点 FFT，但行为 HDL 说明允许在更大的项目中进行仿真和集成。FPGA 供应商的大多数第三方知识产权内核和 Xilinx 的一些免费的 FFT 内核都采用了这种内核类型。由于布局固定，因此时序和资源数据是精确的，不依赖于合成结果。但是不能修改底层的参数，因此如果 FFT 有 12 位或 24 位输入数据，就不能使用 16 位 256 点的 FFT 模块。

4. 知识产权内核的比较和难题

如果比较不同的知识产权模块类型，就不得不在设计灵活性(软内核)与快速出结果和数据可靠性(硬内核)之间做出抉择。虽然软内核更加灵活，如改变系统参数或器件/进程的技术很容易，但可能花费较长的时间进行调试。硬内核已经在硅片上验证过。虽然硬内核缩短了开发、测试和调试时间，但无法查看 VHDL 代码。参数化内核通常是已生成内核的灵活性和可靠性之间的最佳折中方案。

然而，当前的知识产权模块技术还有两大难题亟待解决，分别是模块的定价和与之相关的知识产权的保护。由于内核可重复使用，因此供应商定价就要依赖于客户使用 IP 模块单元的次数。这一问题在专利权方面已经存在了很多年，通常需要长期的许可协议，如果用户滥用就会被处以高额罚金。FPGA 供应商提供的参数化模块(以及设计工具)的定价非常合理，因为客户如果要在许多器件上使用 IP 模块，就得经常从唯一的供应商购买器件，而供应商就会从中获益。但对于第三方 IP 模块提供商，情况就不一样了，他们没有这种第二次收益。

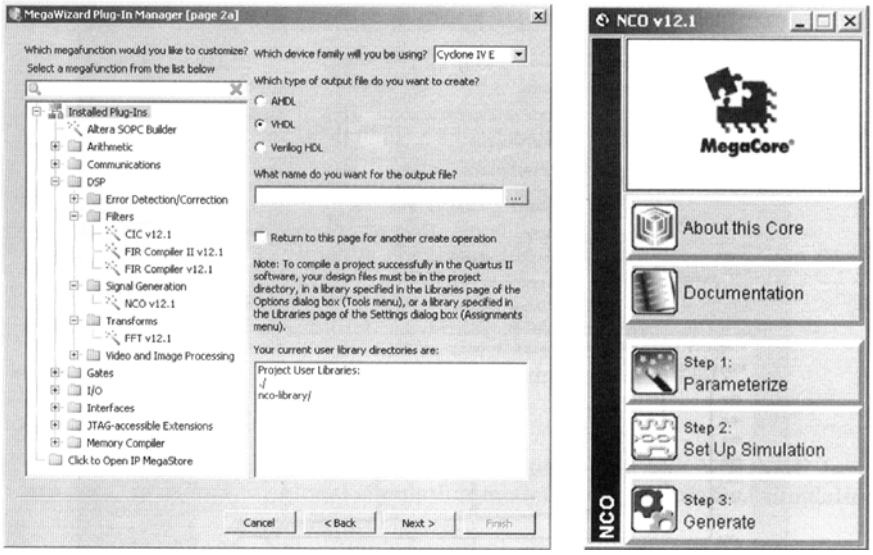
因此，授权协议(特别是对于软内核的授权协议)就必须非常仔细地斟酌。

对于参数化内核的保护，FPGA 供应商采用基于 FlexLM 密钥的方式允许/禁止单个 IP 内核的生成。可以直到硬件验证时鉴定参数化内核，具体方法是提供使用有时间限制的编程文件或将主机与开发板之间用 JTAG 电缆保持永久连接，从而允许用户在购买许可证之前对器件进行编程并且验证设计。例如，Altera 的 OpenCore 评估功能允许在目标系统内仿真 IP 内核的功能，验证设计的功能性并且方便快捷地估计其规模和速度。对 IP 内核功能完全满意并且想要将设计投入生产时就可以购买许可证，这样才允许生成不受时间限制的编程文件。Quartus 软件会自动从 Altera 的网站上下载最新版的 IP 内核。虽然许多第三方 IP 供应商也支持 OpenCore 评估流程，但需要自己直接联系 IP 供应商来启用 OpenCore 功能。

对软内核的保护更加困难。修改 HDL 可以使软内核非常难以阅读，也有建议在高级设计中通过将外部硬件最小化方式嵌入水印[28]。水印应该是健壮的，水印中某一位的改变不应破坏对拥有者的授权。

5. 基于知识产权内核的 NCO 设计

最后利用上一节的案例研究评价 IP 模块的设计过程，不过这次的设计将采用 Altera 的 NCO 内核生成器。NCO 编译器生成针对 Altera 器件优化的数控振荡器(Numerically Controlled Oscillator, NCO)。我们还可以使用 IP 工具平台接口实现包括基于 ROM、基于 CORDIC 或基于乘法器的各种 NCO 体系结构。MegaWizard 还包括基于参数设置动态显示 NCO 的功能的时域和频域图形。对于简单的评价，以项目(即 NCO)命名 IP 内核。打开一个新项目，命名为 NCO，单击 Tools 菜单下 MegaWizard Plug-In Manager 按钮。第一步，在 MegaWizard Plug-In Manager 窗口选中 NCO 模块，如图 1-22(a)所示。可以在 DSP 内核下的 Signal Generation 组中找到 NCO 模块，然后选择想要的输出格式(AHDL、VHDL 还是 Verilog)并指定工作目录。接下来弹出 IP 工具平台窗口(如图 1-22(b)所示)，就可以访问该文档并开始第 1 个步骤，也就是给模块赋参数值。要想再现前一节的函数发生器，需要在参数窗口中选择 32 位累加器。然而，最小的输出位宽为 10 位，并且在之前的设计中不可使用 8 位输出。我们在参数窗口使用 Large Rom 生成算法，如图 1-23 所示。在 fun_text 研究中，时钟速率为 100MHz，输出过程有 20 个时钟周期或等效 5MHz 的频率。虽然相位脉动会使噪声更加均匀地分布，但需要两倍的 LE。随着相位抖动，我们会得到 60dB 左右的旁瓣抑制;若无抖动,我们将得到 50dB 左右的旁瓣抑制,在 NCO 窗口下方的 Frequency Domain Response 图中可以看到。在 Implementation 窗口中选择 Single Output，因为 I/Q 接收器只需要正弦信号输出而不需要余弦信号，详情请参阅第 7 章相关内容。Resource Estimation 提供的数据如下：72 个 LE、2560 个存储器位和一个 M9K 模块。如果想要生成行为 HDL 代码，当参数选择满足要求后就进入到第 2 个步骤，这样可以加快仿真速度。由于模块较小，因此取消这一选项，直接用完整的 HDL 生成代码。接下来继续第 3 个步骤，也就是在工具平台上选择 Generate。表 1-14 列出了所生成的文件并给出每个文件的概述。



(a) 库元件的选取　　(b) IP 工具平台

图 1-22　NCO 的 IP 设计

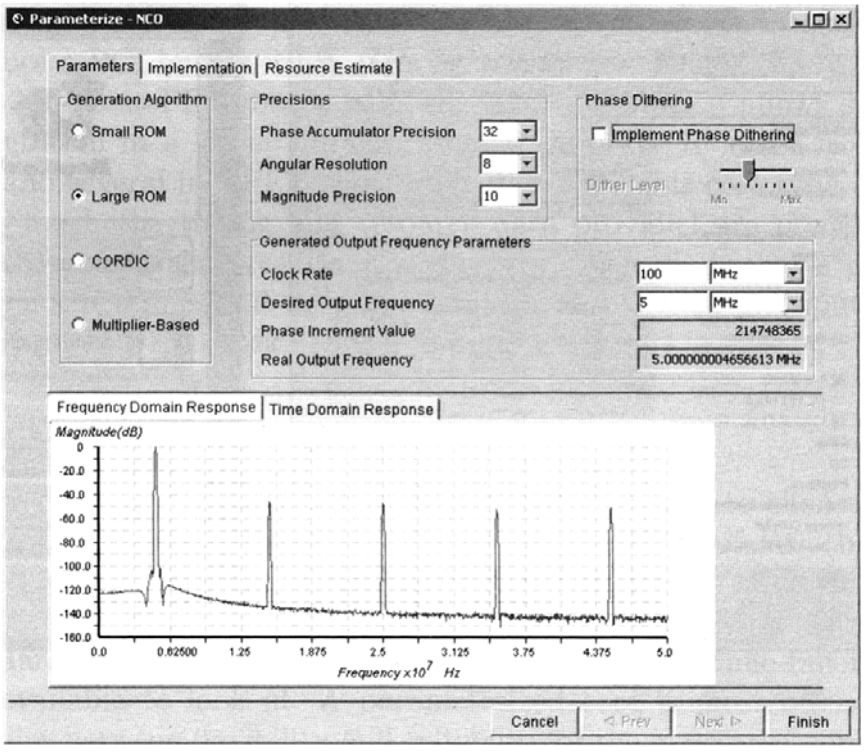


图 1-23　根据前一节案例研究所得数据设置 NCO 内核的 IP 参数

表 1-14 为 NCO 内核生成的 IP 文件

文 件	说 明
nco.vhd	定制 IP 内核功能的 VHDL 顶层描述符号文件
nco.cmp	IP 内核函数变量的 VHDL 组件声明
nco.bsf	IP 内核函数变量的 Quartus II 符号文件
nco_st.v	生成的 NCO 可合成网表
nco.vho	VHDL IP 函数仿真模型
nco_tb.vhd	VHDL 测试平台
nco_vho_msim.tcl	MODELSIM 仿真软件中运行 VHDL IP 功能仿真模型的 MODELSIM TCL 脚本
nco_wave.do	MODELSIM 波形文件
nco_model.m	描述 MATLAB 位精确模型的 MATLAB M-文件
nco_tb.m	MATLAB 测试平台
nco_sin.hex	Intel 的十六进制格式 ROM 初始化文件
nco.vec	Quartus 向量文件
nco_nativelink.tcl	NativeLink 仿真测试平台
nco.qip	Quartus 项目信息
nco.html	列出所有已生成文件的 IP 内核函数报告文件

可以看到与组件文件同时生成的不仅有 VHDL 文件和 Verilog 文件，还提供了 MATLAB (精确到位)和 ModelTech(精确到周期)测试平台以启用简单的验证路径。我们决定直接使用内核作为顶层设计输入，因此避免需要在另一个设计中实例化模块以及连接输入和输出。通过检查顶层 VHDL 文件 nco.vhd，注意到模块还有预期的模块输入 clk、phi_inc_i 以及输出 fsin_o 信号。但是也有其他几个有用的控制信号，例如 reset_n、clken 和 out_valid，其功能显而易见。在完全编译后进行时序仿真，生成 nco.vho 文件。有了完全编译的数据，就可以将实际资源要求与估计值进行比较。需要的存储器和预计的模块 RAM 要正确，但 LE 的数量分别是 88 个(实际值)和 72 个(估计值)，误差范围是 18%。

为了仿真设计，我们使用生成的 TCL 脚本。启动 MODELSIM 仿真器并切换到 NCO 目录，然后在命令窗口中键入 do nco_vho_msim.tcl。几个库和设计文件被编译，执行 22 000ns 时长的仿真。我们放大起始的几个时钟周期，看输出有效延迟(≈6 个时钟周期)和正弦周期，如图 1-24 所示。将相同的值 M = 214 748 365 用在函数发生器中(请参阅图 1-20)，在输出信号中得到 20 个时钟周期。我们还会注意到 IP 模块的一个小问题，由于输出是一个有符号的值，而 D/A 转换器要求是无符号数(或更准确地说是二进制偏移量)。在软内核中可以修改该设计的 HDL 代码，但在参数化内核中没有这一选项。不过可以通过追加一个加法器来解决这个问题，在输出结果中加一个常数 512，使得输出结果成为偏移二进制表示方式。偏移二进制不是可以在模块中选择的参数，而是我们遇到的额外的设计工作。这是使用参数化内核的典型经验——内核可以缩减 90%甚至更多的设计时间，但通常还需要

一点儿额外的设计工作来满足项目要求。

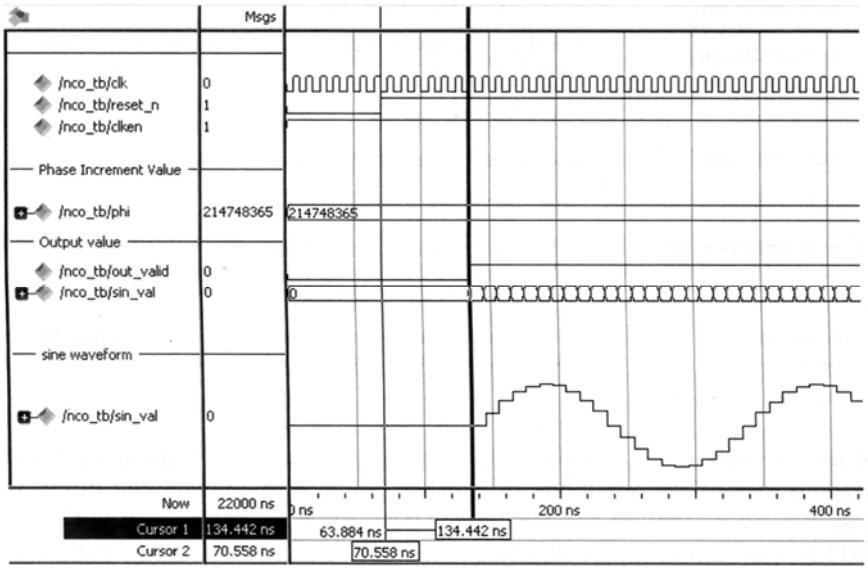


图 1-24　NCO IP 设计的测试平台，通过时序仿真验证

1.5　练习

注意：

如果没有使用 Quartus II 软件的经验，请参阅 1.4.3 节的案例研究。如果没有特别说明，Quartus II 合成分析均使用 Cyclone IV E 系列中的 EP4CE115F29C7 器件。

1.1 只使用两个输入“与非”门来实现一个全加器：

(a) $s = a \oplus b \oplus c_{in}$(注：$\oplus$ = XOR，“或非”门)。

(b) $c_{out} = a \times b + c_{in} \times (a + b)$ (注：+ = OR，“或”门；× = AND，“与”门)。

(c) 利用“与非”门实现“非”门、“与”门和“或”门，从而说明两输入“与非”门是通用的。

(d) 重复(a)~(c)对应的练习，生成两输入“或非”门。

(e) 重复(a)~(c)对应的练习，生成两输入乘法器 $f = xs' + ys$。

1.2 (a) 利用 ModelSim-Altera 工具编译 HDL 文件 example。启动 ModelSim-Altera 工具，改变 HDL 文件的目录。利用 vlib work 生成工作目录。然后利用 vcom exaple.vhd 开始编译。

(b) 利用文件 example.do 对该设计进行仿真。利用 do example.do 0 启动脚本进行功能仿真。

(c) 利用 Quartus II 编译器的时序编译 HDL 文件 example。用 Processing 菜单中的

Compiler Tool 执行完整的编译。

(d) 利用 example.do 脚本对该设计进行时序仿真。利用 do example.do 1 启动脚本进行时序仿真。

1.3 (a) 编写 ModelSim-Altera 仿真脚本 example.do，为 clk、a、b、op1 生成波形文件，近似如图 1-25 所示。

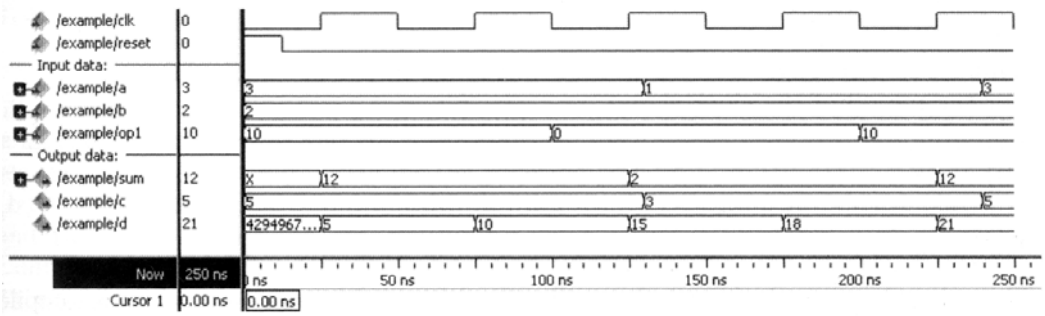


图 1-25 例 1.1 的波形文件

(b) 利用 example.do 脚本进行仿真。

(c) 解释 a、b、op1 和 sum、d 之间的代数关系。

1.4 (a) 在 Assignments 菜单的 EDA Tool Settings 选项下的 Analysis & Synthesis Settings 部分将合成 Optimization Technique 设置为 Speed、Balance 或 Area，编译 HDL 文件 fun_text。

(b) 评估(a)中设计的有关缓慢 85C 时序模型的时序电路性能 Fmax 和 LE 的使用情况，并解释其结果。

1.5 (a) 参照 Altera 教程中的 Using TimeQuest Timing Analyzer 设计生成一个 SDC 文件。在 Assignments 菜单的 EDA Tool Settings 选项下的 Analysis & Synthesis Settings 部分将合成 Optimization Technique 设置为 Speed，编译 HDL 文件 fun_text。

分别将时钟信号的周期设置为：

(a) 20ns

(b) 10ns

(c) 5ns

(d) 3ns

使用时钟报告决定时间裕量。

请注意，对于每个时钟周期规范，需要运行电路的完整编译，还需要报告电路资源的任何改变。

1.6　(a) 修改 fun_text.vhd 文件，使用 lpm_rom 组件和 MIF 文件 sine.mif。在 ModelSim-Altera 中进行电路仿真。

(b) 选择 File | Open 命令打开 sine.mif 文件，看到文件显示在 Memory Editor 中。接下来选择 File | Save As 命令并选择 Save as type 选项为(*.hex)，将文件存储为 Intel 十六进制格式的 sine.hex。

(c) 更改 fun_text HDL 文件，使其使用 Intel 十六进制形式的 ROM 表文件 sine.hex，进而通过仿真验证结果的正确性。

1.7　(a) 应用 Quartus II 软件设计一个 32 位加法器。

(b) 添加 I/O 寄存器，测定其时序电路性能 Fmax，将结果与例 1.2 中的数据进行比较。

1.8　(a) 用 Quartus II 软件设计如图 1-26 (a)所示的 PREP 基准 1。PREP 基准 1 是一个数据通路电路，包含一个 4 到 1 的 8 位乘法器、一个 8 位寄存器，其后跟随一个移位寄存器，受移位/加载输入信号 sl 控制。对于 sl = 1 寄存器的内容循环移动一位，也就是 $q(k) = q(k-1)$，$1 \leqslant k \leqslant 7$，且 $q(0) \leqslant q(7)$。所有触发器的复位信号 rst 是异步复位信号，8 位寄存器通过 clk 的上升沿触发。查看图 1-26(c)中功能测试的仿真结果。

(b) 使用 TimeQuest 缓慢 85C 模型确定单级设计的时序电路性能 Fmax 和所用资源(LE、乘法器和 M4K/M9K)。在 Assignments 菜单的 EDA Tool Settings 下的 Analysis & Synthesis Settings 部分将合成 Optimization Technique 设置为 Speed、Balance 或 Area，编译 HDL 文件。对于规模和时序电路性能 Fmax，哪个合成选项是最优的？

分别选择下列器件进行仿真：

(b1) Cyclone IV E 系列的 EP4CE115F29C7

(b2) Cyclone II 系列的 EP2C35F672C6

(b3) MAX7000S 系列的 EPM7128SLC84-7

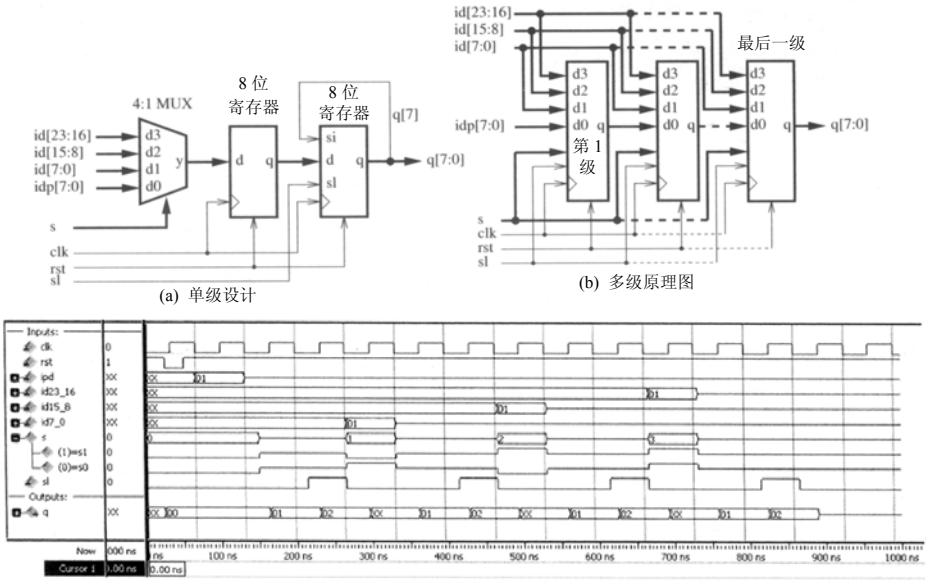
(c) 为基准 1 设计如图 1-26(b)所示的多级原理图。

(d) 使用 TimeQuest 缓慢 85C 模型确定 PREP 基准 1 级数最多的多级原理图设计的时序电路性能 Fmax 和所用资源(LE、乘法器和 M4K/M9K)。使用(b)中的优化合成选项分别选中下列器件进行仿真：

(d1) Cyclone IV E 系列的 EP4CE115F29C7

(d2) Cyclone II 系列的 EP2C35F672C6

(d3) MAX7000S 系列的 EPM7128SLC84-7



(a) 单级设计　　(b) 多级原理图

(c) 测试功能的测试平台

图 1-26　PREP 基准 1

1.9 (a) 用 Quartus II 软件设计如图 1-27(a)所示的 PREP 基准 2。PREP 基准 2 是一个计数器电路，其中两个寄存器分别加载计数器启动和停止的值。该设计有两个 8 位寄存器和一个计数器，异步复位信号 rst 和同步加载使能信号(ld、ldpre 和 ldcomp)和通过 clk 上升沿触发的触发器。计数器可以通过 2:1 乘法器(受 sel 输入信号控制)直接加载来自 data1 的输入信号或是来自保存 data2 对应值的寄存器。在计数器的 data 值与存储在 ldcomp 寄存器中的值相等的条件下，启用计数器的加载信号。尝试匹配图 1-27(c)中功能测试的仿真。注意，在原始 PREP 定义与实际实现之间存在失配现象：不能想当然认为计数器在复位后就会开始计数，因为所有的寄存器都置 0，ld 将一直为真，从而强制将计数器置 0。另外在仿真时，减少测试平台的信号值，使仿真满足在 1μs 时间范围内。

(b) 使用 TimeQuest 缓慢 85C 模型确定单级设计的时序电路性能 Fmax 和所用资源(LE、乘法器和 M4K/M9K)。在 Assignments 菜单的 EDA Tool Settings 下的 Analysis & Synthesis Settings 部分将合成 Optimization Technique 设置为 Speed、Balance 或 Area，编译 HDL 文件。对于规模和时序电路性能 Fmax，哪个合成选项是最优的？

分别选择下列器件进行仿真：

(b1) Cyclone IV E 系列的 EP4CE115F29C7

(b2) Cyclone II 系列的 EP2C35F672C6

(b3) MAX7000S 系列的 EPM7128SLC84-7

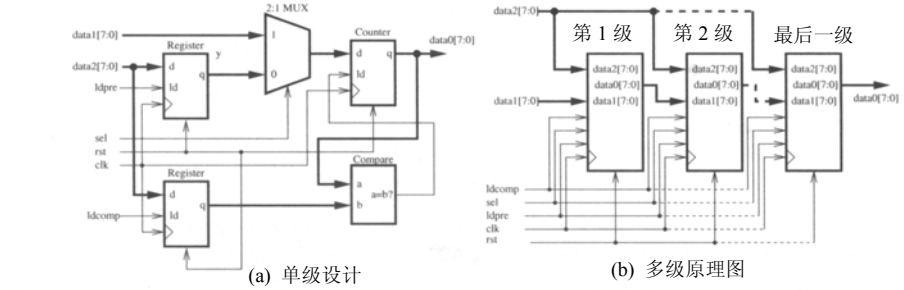
(c) 为基准 2 设计如图 1-26(b)所示的多级原理图。

(d) 使用 TimeQuest 缓慢 85C 模型确定 PREP 基准 2 级数最多的多级原理图设计的时序电路性能 Fmax 和所用资源(LE、乘法器和 M4K/M9K)。使用(b)中的优化合成选项分别选中下列器件进行仿真：

(d1) Cyclone IV E 系列的 EP4CE115F29C7

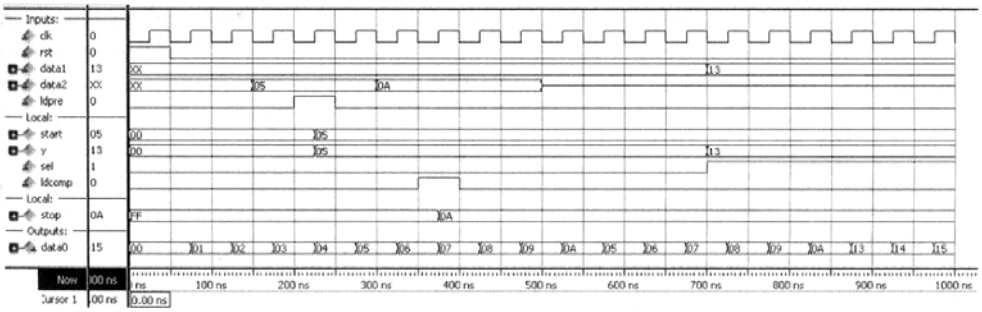
(d2) Cyclone II 系列的 EP2C35F672C6

(d3) MAX7000S 系列的 EPM7128SLC84-7



(a) 单级设计

(b) 多级原理图



(c) 测试功能的测试平台

图 1-27　PREP 基准 2

1.10 应用 Quartus II 软件并用结构 HDL 风格(只使用一或两个基本输入门电路，也就是“非”门、“与”门、“或”门)和行为 HDL 风格分别编写两种不同的代码，用于

(a) 2:1 多路复用器

(b) XNOR(异或非)门

(c) 半加器

(d) 2:4 译码器(多路分配器)

VHDL 设计注意事项：在结构设计文件中使用 a_74xx Altera SSI 元件。由于元件标识符不能以数字开头，因此 Altera 在每个 74 系列元件的前面都添加了 a_。为了找到输入端口和输出端口的名称及数据类型，需要在 Altera 安装路径下查询库文件 libraries\vhdl\altera\ maxplus2.vhd。这样会发现库文件使用了 STD_LOGIC 数据类型，端口名称是 a_1、a_2 和 a_3(根据需要)。

(e) 通过以下方式验证该设计的功能：

(e1) Functional 仿真。

(e2) 找到位于 Tools 菜单的 Netlist Viewers 选项下的 RTL Viewer。

1.11 应用 Quartus II 软件语言模板分别编译下列 HDL 设计：

(a) 三态缓冲器，请参阅 Logic | Tri-State 命令。

(b) 带有全部控制信号的触发器，请参阅 Logic | Registers | Full-Featured Positive Edge Register with All Secondary Signals 命令。

(c) 二进制计数器，请参阅 Full Designs | Arithmetic | Counters 命令。

(d) 带有异步复位信号的状态机，请参阅 Full Designs | State Machines 命令。

打开一个新的 HDL 文本文件，从 Edit 菜单中选择 Insert Template 选项。

(e) 通过以下方式验证该设计的功能：

(e1) Functional 仿真。

(e2) RTL Viewer 可以在 Tools 菜单的 Netlist Viewers 选项下找到。

1.12 应用 Quartus II 软件帮助中的 search 选项研究下列 HDL 设计：

(a) 14 个计数器，参见 search | implementing sequential logic 命令。

(b) 手动指定状态赋值，参见 Search | enumsmch 命令。

(c) 锁存器，参见 Search | latchinf 命令。

(d) 计数器，参见 Search | proc | Using Process Statements 命令。

(e) 实现 CAM、RAM 和 ROM，参见 Search | ram256x8 命令。

(f) 实现用户定义的元件，参见 Search | reg24 命令。

(g) 实现带有 clr、load 和 preset 信号的寄存器，参见 Search | reginf 命令。

(h) 状态机，参见 Search | state_machine | Implementing…命令。

打开一个新项目和 HDL 文本文件，复制/粘贴 HDL 代码，保持并编译代码。注意：在 VHDL 代码中需要添加 STD_LOGIC_1164 IEEE 库文件，这样运行代码就不会出错了。

(i) 通过以下方式验证该设计的功能：

(i1) Functional 仿真。

(i2) RTL Viewer 可以在 Tools 菜单的 Netlist Viewers 选项下找到。

1.13 确定下列 VHDL 标识符是否有效：

(a) VHSIC (b) h333 (c) A_B_C

(d) XyZ (e) N#3 (f) My-name

(g) BEGIN (h) A_ _B (i) ENTITI

1.14 确定下列 VHDL 字符串字面量是否有效：

(a) B"11_00" (b) 0"5678" (c) 0"0_1_2"

(d) X"5678"　　(e) 16#FfF#　　(f) 10#007#

(g) 5#12345#　　(h) 2#0001_1111_#　　(i) 2#00_00#

1.15 确定表示以下整型数所需的位数：

(a) INTEGER RANGE 10 TO 20；

(b) INTEGER RANGE −2**6 TO 2**4−1；

(c) INTEGER RANGE −10 TO −5；

(d) INTEGER RANGE −2 TO 15；

注意：

**是表示幂的符号。

1.16 确定表 1-15 中 VHDL 代码的错误行(回答 Y/N)，并解释错误原因或给出正确代码。

表 1-15　示例 VHDL 代码(一)

VHDL 代码	是 否 有 误	解 释 原 因
LIBRARY ieee; /*Using predefined packages*/		
ENTITY error is		
PORTS (x: in BIT; c: in BIT;		
Z1: out INTEGER; z2: out BIT);		
END error		
ARCHITECTURE error OF has IS		
SIGNAL s; w: BIT;		
BEGIN		
w := c;		
Z1<= x;		
P1: PROCESS (x)		
BEGIN		
IF c= '1' THEN		
X <= z2;		
END PROCESS P0;		
END OF has;		

1.17 确定表 1-16 中 VHDL 代码的错误行(回答 Y/N)，并解释错误原因或给出正确代码。

表 1-16 示例 VHDL 代码(二)

VHDL 代码	是 否 有 误	解 释 原 因
LIBRARY ieee; /*Using predefined packages*/		
USE altera.std_logic_1164.ALL;		
ENTITY srhiftreg IS		
GENERIC (WIDTH :POSITIVE =4);		
PORT (clk,din : IN STD_LOGIC;		
dout : OUT STD_LOGIC);		
END		
ARCHITECTURE a OF shifting IS		
COMPONENT d_ff		
PORT (clock, d: IN std_logic;		
q : OUT std_logic);		
END d_ff;		
SIGNAL b: logic_vector(0 TO width－1);		
BEGIN		
d1: d_ff PORT MAP (clk, b(0), din);		
g1: FOR j IN 1 TO width－1 GENERATE		
d2: d_ff		
PORT MAP (clk=> clock,		
din=> b(j-1),		
q => b(j));		
END GENERATE d2;		
dout <= b(width);		
END a;		

1.18 为下列过程语句声明确定：

(a) 合成的电路和为 I/O 端口设定标签。

(b) 设计的成本，假定每个加法器/减法器的成本为 1。

(c) 每一过程的电路的关键路径(也就是最坏的情况)，假定每个加法器/减法器的延迟为 1。

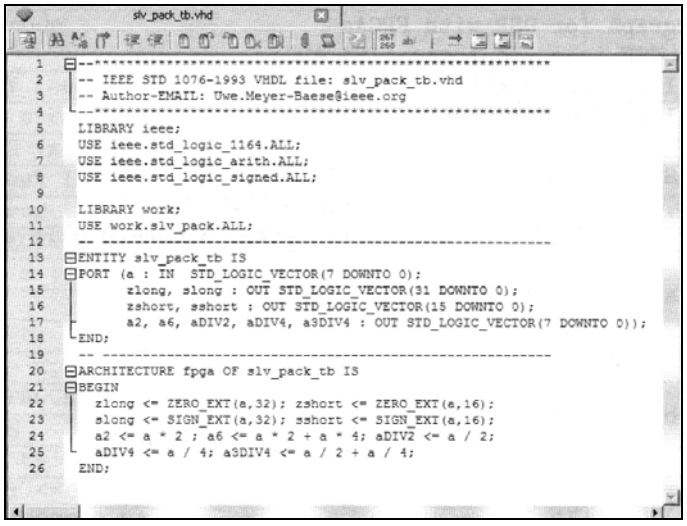
```
-- QUIZ VHDL2gragh for DSP with FPGAs
LIBRARY ieeel; USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY qv2g IS
  PORT(a, b, c, d : IN std_logic_vector (3 DOWNTO 0);
       u, v, w, x, y, z : OUT std_logic_vector(3 DOWNTO 0));
```

```
END;
ARCHITECTURE a OF qv2g IS BEGIN

  P0 : PROCESS (a, b, c, d)
  BEGIN
    u <= a + b – c + d;
  END PROCESS;

  P1 : PROCESS (a, b, c, d)
  BEGIN
    v <= (a + b) – (c – d);
  END PROCESS;

  P2 : PROCESS (a, b, c)
  BEGIN

    w <= a + b + c;
    x <= a – b – c;
  END PROCESS;

  P3 : PROCESS (a, b, c)
  VARIABLE t1 : std_logic_vector (3 DOWNTO 0);
BEGIN
    t1 <= b + c;
    y <= a + t1;
    z <= a – t1;
  END PROCESS;
END;
```

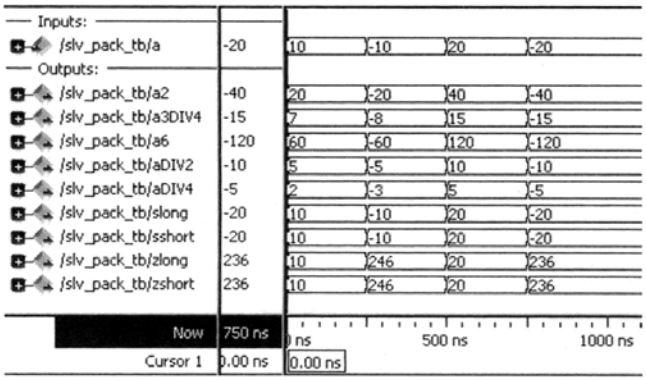
1.19 (a) 为 STD_LOGIC_VECTOR 数据类型的名为 SIGN_EXT(ARG,SIZE)和 ZERO_EXT (ARG, SIZE)的 0 和符号扩展设计一个函数。

(b) 设计“*”和“/”函数重载以实现 STD_LOGIC_VECTOR 数据类型的乘法和除法运算。

(c) 用图 1-28 所示的测试平台验证设计功能的正确性。



(a) HDL 代码



(b) 功能仿真效果

图 1-28 STD_LOGIC_VECTOR 包测试平台

1.20 为 STD_LOGIC_VECTOR 数据类型设计一个函数库，实现如下操作(仅在 VHDL-1993 中为 BIT_VECTOR 数据类型定义)：

(a) SRL (b) SRA (c) SLL (d) SLA

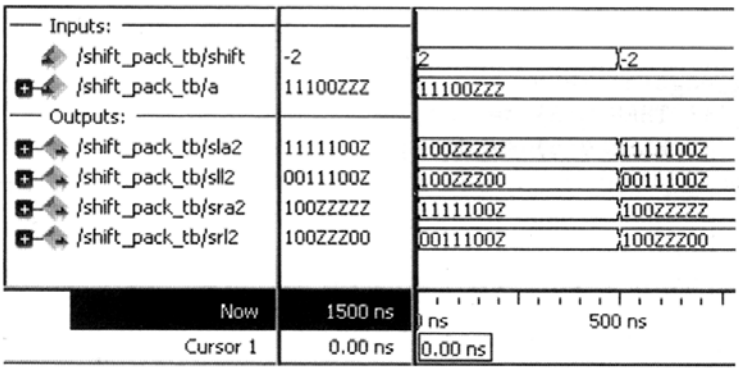
(e) 用图 1-29 所示的测试平台验证设计功能的正确性。注意：高阻抗值 Z 是 STD_LOGIC_VECTOR 数据类型的一部分，但不包括在 BIT_VECTOR 数据类型中。在函数内，左移/右移的一个负值应当被右移/左移的一个相应正值代替。

shift_pack_tb.vhd

```
--*******************************************************
-- IEEE STD 1076-1993 VHDL file: shift_pack_tb.vhd
-- Author-EMAIL: Uwe.Meyer-Baese@ieee.org
--*******************************************************
LIBRARY ieee;
USE ieee.std_logic_1164.ALL; USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_signed.ALL;

LIBRARY work; USE work.shift_pack.ALL;
-- -----------------------------------------------------
ENTITY shift_pack_tb IS
PORT (a     : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      shift : IN INTEGER;
      sll2, sla2, srl2, sra2 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END;
-- -----------------------------------------------------
ARCHITECTURE fpga OF shift_pack_tb IS
BEGIN
  PROCESS (shift, a)
  BEGIN
    IF shift = -2 THEN
      sll2 <= a SLL -2; sla2 <= a SLA -2;
      srl2 <= a SRL -2; sra2 <= a SRA -2;
    ELSE
      sll2 <= a SLL 2; sla2 <= a SLA 2;
      srl2 <= a SRL 2; sra2 <= a SRA 2;
    END IF;
  END PROCESS;
END;
```

(a) HDL 代码

图 1-29 STD_LOGIC_VECTOR 移位库测试平台



(b) 功能仿真结果

图 1-29　(续)

1.21 为下列 PROCESS 声明确定合成的电路类型(组合逻辑、闭锁器、D 触发器还是 T 触发器)和 a、b、c 的函数，也就是时钟、非对称设置(AS)或复位(AR)以及对称设置(SS)或复位(SR)。利用表 1-17 设定分类。

表 1-17　用来设定分类的表

PROCESS	电 路 类 型	CLK	AS	AR	SS	SR
P0						
P1						
P2						
P3						
P4						
P5						

```
LIBRARY ieee; USE ieee.std_logic_1164.ALL;

ENTITY quiz IS
  PORT(a, b, c   : IN std_logic;
         d          : IN std_logic_vector(0 TO 5);
       q         : BUFFER std_logic_vector(0 TO 5));
END quiz;
ARCHITECTURE a OF quiz IS BEGIN
  P0: PROCESS (a)
  BEGIN
     IF rising_edge(a) THEN
        q(0) <= d(0);
     END IF;
  END PROCESS P0;

  P1: PROCESS (a, d)
```

```
BEGIN
   IF a= '1'  THEN q(1) <= d(1);
              ELSE q(1) <= '1';
   END IF;
END PROCESS P1;

P2: PROCESS (a, b, c, d)
BEGIN
  IF a= '1' THEN q(2) <= '0';
  ELSE IF rising_edge(b) THEN
         IF c = '1' THEN q(2) <= '1';
                    ELSE q(2) <= d(1);
         END IF;
       END IF;
    END IF;
END PROCESS P2;

P3: PROCESS (a, b, d)
BEGIN
  IF a= '1' THEN q(3) <= '1';
  ELSE IF rising_edge(b) THEN
          IF c = '1' THEN q(3) <= '0';
                     ELSE q(3) <= not q(3);
          END IF;
        END IF;
    END IF;
END PROCESS P3;

P4: PROCESS (a, d)
BEGIN
  IF a= '1' THEN q(4) <= d(4);
    END IF;
END PROCESS P4;

P5: PROCESS (a, b, d)
BEGIN
  IF rising_edge(a) THEN
     IF b = '1' THEN q(5) <= '0';
                ELSE q(5) <= d(5);
     END IF;
   END IF;
END PROCESS P5;
```

1.22 给定如下 MATLAB 指令：

```
a = -1:2:5
b = [ones(1,2), zeros(1,2)]
```

```
c =a*a'
d = a.*a
e = a'*a
f = conv(a,b)
g = fft(b)
h = ifft(fft(a).*fft(b))
```

确定 a~h 对应的输出效果。