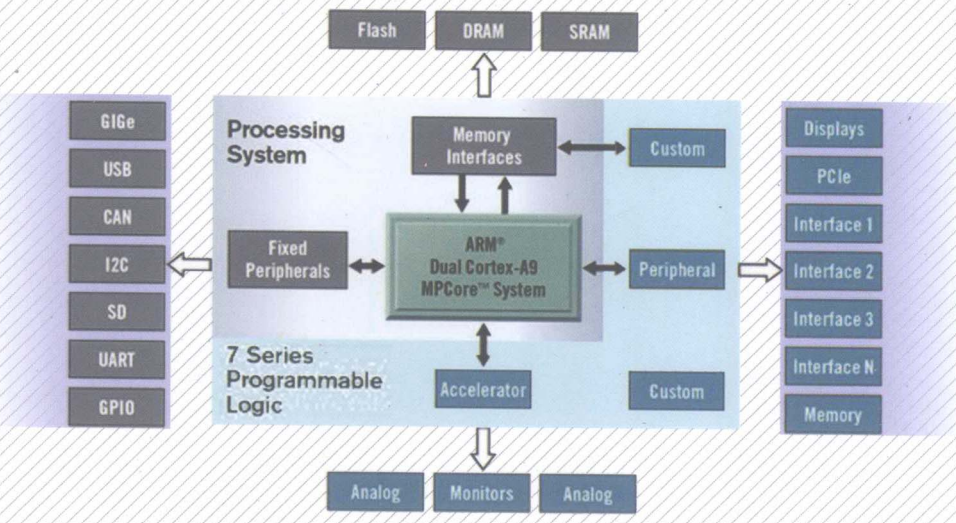

FPGA 技术开发 高级篇

李洪涛 顾 陈 朱晓华 编著

国防工业出版社
National Defense Industry Press

本书附光盘一张

FPGA 技术开发
——高级篇

李洪涛　顾　陈　朱晓华　编著

国防工业出版社

·北京·

内 容 简 介

本书系统介绍了利用 EDA 软件开发 FPGA 的基本流程，FPGA 高级开发技术以及 FPGA 发展的趋势等。全书内容主要包括第三方 EDA 软件介绍；Xilinx 公司开发软件 ISE 简介；利用嵌入式逻辑分析仪调试 FPGA 的方法与技巧；FPGA 底层开发技术——源语、约束与伪指令的设计；对 Xilinx 公司 IP 软核与嵌入式硬件资源的介绍；结合开发实例详细介绍 FPGA 在通信系统中的应用等。

本书第 1 章到第 3 章介绍利用 EDA 软件开发、调试 FPGA 的基本流程，第 4 章介绍源语、约束与伪指令等 FPGA 高级开发技术；第 5 章介绍利用 IP 软核简化 FPGA 设计的方法；第 6 章介绍 FPGA 发展的趋势——嵌入式硬件资源；第 7 章介绍 FPGA 在通信系统中的应用以及开发实例。

全书内容丰富、结构合理、图文并茂，便于实施系统教学。本书可以作为高等工科院校电类专业的教学用书，也可用于自学或供工程技术人员参考。

图书在版编目（CIP）数据

FPGA 技术开发. 高级篇/李洪涛，顾陈，朱晓华编著. —北京：国防工业出版社, 2013.9

ISBN 978-7-118-08903-5

Ⅰ. ①F… Ⅱ. ①李… ②顾… ③朱… Ⅲ. ①可编程序逻辑器件 Ⅳ. ①TP332.1

中国版本图书馆 CIP 数据核字（2013）第 195914 号

※

国防工业出版社出版发行

（北京市海淀区紫竹院南路 23 号 邮政编码 100048）

北京奥鑫印刷厂印刷

新华书店经售

*

开本 787×1092 1/16 **印张** 16¼ **字数** 369 千字

2013 年 9 月第 1 版第 1 次印刷 **印数** 1—2500 册 **定价** 49.00 元（含光盘）

（本书如有印装错误，我社负责调换）

国防书店：（010）88540777 发行邮购：（010）88540776

发行传真：（010）88540755 发行业务：（010）88540717

前　言

自 1985 年 Xilinx 公司推出全球首款 FPGA 产品——XC2064 之日起，FPGA 经历了近 30 年的发展。通过提高芯片密度、增加外围硬件电路、降低功耗以及成本等手段，FPGA 已经不仅在传统 ASIC 领域，还向众多新兴应用领域加速渗透。如今 FPGA 将应用触角不断伸向更为广泛的领域。

随着技术的不断发展，FPGA 的功能已由最初的实现基本组合逻辑发展到可实现算法逻辑再到数字信号处理、高速串行收发以及嵌入式处理等，FPGA 真正从系统应用中的配角变成了主角。在以高速发展的半导体产业里，30 年足够改变一切。可以想象在未来的若干年内每一个电子设备都将有一个 FPGA 的理想正成为现实。

如今许多系统都是以 FPGA 为中心来设计的。FPGA 走过了从初期开发应用到限量生产应用，再到大批量生产应用的发展历程。从技术上来说，FPGA 最初以实现逻辑功能为主要目标，现在强调平台概念，加入数字信号处理、嵌入式处理、高速串行和其他高端技术，从而被应用到更多更广的领域。

笔者长期从事 FPGA 以及与之相关的研究工作，对 FPGA 的特点有着比较深刻的理解。目前，专门讲授 FPGA、Verilog HDL 以及与之相关完整开发过程的书籍还比较少，大量的资料需要查找和翻阅英文文献，而 FPGA 开发是一项实践性非常强的技术，需要积累丰富的经验，这对初学者造成了很大的困难。笔者在《Verilog HDL 与 FPGA 开发设计及应用》一书中，详细介绍了 Verilog HDL 的开发过程以及 FPGA 的基本知识，可以作为初学者的入门教材，本书在此书的基础上，通过查阅大量参考文献，以及结合十多年的 FPGA 开发经验，详细介绍了与 FPGA 开发相关的各种高级技术，以及 FPGA 的发展趋势等内容，希望本书对从事 FPGA 以及相关技术的设计人员能够提供一定的帮助。

本书系统地介绍了利用 EDA 软件开发 FPGA 的基本流程，FPGA 高级开发技术以及 FPGA 发展的趋势等。全书共分为 7 章。第 1 章介绍与 FPGA 开发相关的 3 类主要 EDA 软件，包括用于仿真的 Modelsim 软件，用于综合的 Synplify/Synplify Pro 软件以及用于高级调试的 Debussy 软件。第 2 章介绍 Xilinx 公司的开发软件 ISE 以及其具体开发过程。第 3 章介绍 FPGA 的新型调试工具——嵌入式逻辑分析仪 ChipScope 及其具体应用。第 4 章介绍 FPGA 高级开发技术，包括 FPGA 源语的意义及其设计方法，伪指令的意义以及如何添加伪指令，最后介绍约束文件的意义及如何编写一个完成的约

束文件。第 5 章介绍简化 FPGA 开发过程的手段——FPGA IP 软核的设计。第 6 章介绍 FPGA 发展的趋势——嵌入式硬核及其设计方法。第 7 章介绍 FPGA 在通信系统中的应用，通过 MAC 开发等几个实例，详细介绍 FPGA 完整的开发过程，具有较强的工程应用价值。

在本书撰写过程中，得到了南京理工大学电子工程与光电技术学院的各位领导、老师和同事们的支持，特别是苏卫民教授、王建新教授和顾红教授为本书提出了宝贵意见和指导。陈诚、曾文浩、陈恒明、马义耕、赵恒博士研究生以及何俊杰、陈曦、杨宇宸、王骏扬、顾思婧、郑驹、蒋艳、姚力等硕士研究生在搜集资料、文章编排和校对方面做了大量工作。在此表示衷心的感谢。

本书的第 1 至第 4 章及第 7 章由李洪涛撰写，并对全书进行了统稿，第 5 章由朱晓华撰写，第 6 章由顾陈撰写。

本书可作为从事 FPGA、数字信号处理以及相关技术设计研究人员的参考书籍，同时也可以作为高校自动化、电子信息、信号处理、通信系统等相关专业本科生、研究生的教材或者参考资料。

技术的发展是无止境的，本书只是着重讲述了原理、概念以及基本的设计方法。希望本书能够成为广大技术爱好者和从事 FPGA 以及相关技术开发的专业人士前进的铺路石。由于时间仓促，加上作者水平有限，书中难免有不妥之处，希望各位读者与同行批评指正。

作　者

2013 年 6 月于南京

目　录

第 1 章　第三方 EDA 工具

第三方 EDA 工具是指 Xilinx 等 FPGA/CPLD 生产厂商以外的所有其他厂商提供的辅助 FPGA 开发的 EDA 软件工具。业界常用的 EDA 软件很多，根据 FPGA 开发的基本流程，主要分为仿真软件、综合软件以及调试软件等。

本章主要介绍仿真的概念、意义以及仿真软件 Modelsim 的使用方法；综合的概念、意义以及综合工具 Synplify/Synplify Pro 的使用方法；最后介绍调试工具 Debussy 的原理及其使用方法。

1.1　Modelsim 及其应用——前仿真、后仿真的作用与区别

1.1.1　仿真的意义与 Modelsim 软件

1．仿真的意义

仿真是指在软件环境下，验证电路行为与设计意图是否一致的方法。仿真与验证是一门科学，在逻辑设计领域，仿真与验证是整个设计流程中最重要、最复杂和最耗时的步骤。特别是 ASIC 设计中，仿真与验证投入资源与初期逻辑设计的比重约为 10:1。虽然 FPGA/CPLD 设计灵活、可以反复编程，这种灵活性在一定程度上可以弥补仿真与验证的不足，但是对于大型、高速或复杂的逻辑设计，仿真与验证仍是整个流程中最重要的环节。

仿真分为前仿真和后仿真两种。

（1）前仿真：对源代码进行的仿真，主要是验证电路的功能是否符合设计的要求，其特点是不考虑电路的门延时与布线延时，考察重点为电路在理想环境下的行为与设计构想是否一致。前仿真又称为功能仿真。

（2）后仿真：对布局布线后生成的仿真文件进行仿真，主要是验证电路的时序能否满足设计的要求，其特点是考虑电路门延时与布线延时的影响，考察重点为电路在实际环境下的行为与设计构想是否一致。后仿真又称为时序仿真。

2．仿真工具 Modelsim

Modelsim 是业界最优秀的 HDL 语言仿真器，它提供最友好的调试环境，可单内核支持 VHDL 和 Verilog HDL 混合仿真，是 FPGA/ASIC 设计电路仿真的较优选择。它采用直接优化的编译技术、Tcl/Tk 技术、单一内核仿真技术，编译仿真速度快，编译的代码与平台无关，便于保护 IP 核，个性化的图形界面和用户接口，为用户加快调试提供强有力的保障。全面支持 VHDL 和 Verilog HDL 语言的 IEEE 标准，支持 C/C++功能调用和调试。

Modelsim 专业版，具有快速的仿真性能和最先进的调试能力，全面支持 UNIX（包括 64 位）、Linux 和 Windows 平台。

主要特点：

（1）本地编译结构、编译仿真速度快；

（2）单内核 VHDL 和 Verilog HDL 混合仿真；

（3）源代码模版和助手，项目管理；

（4）集成了性能分析、波形比较、代码覆盖等功能；

（5）数据流 ChaseX、Signal Spy 功能；

（6）C 和 Tcl/Tk 接口、C 语言调试功能。

3．Modelsim 的界面

Modelsim 的主界面如图 1.1 所示。

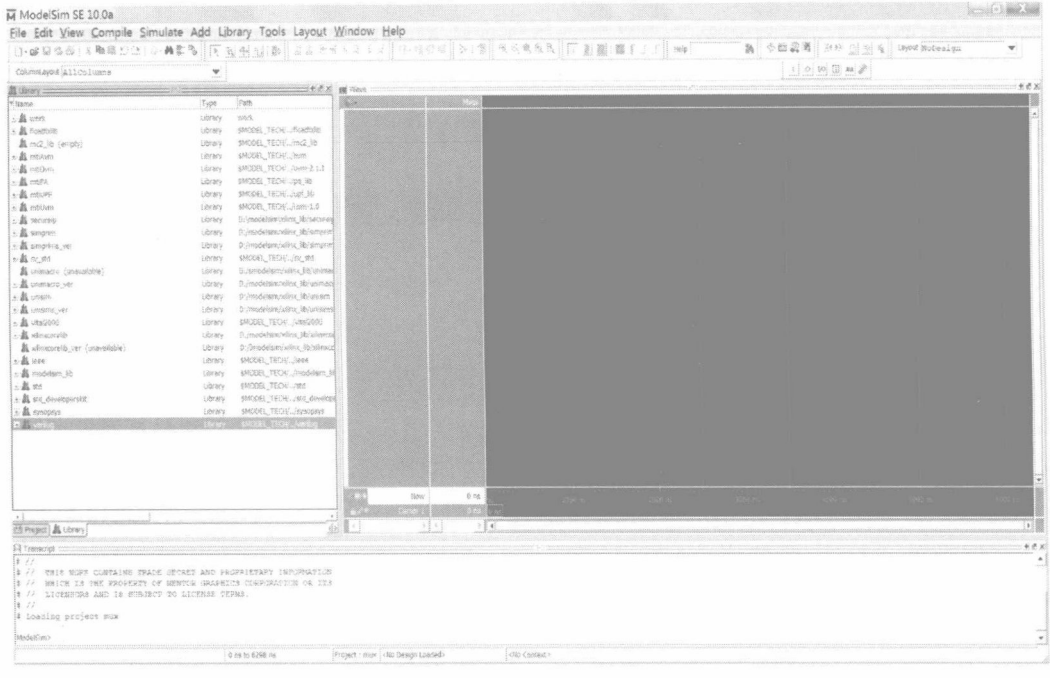

图 1.1　Modelsim 的主界面

图 1.1 是 Modelsim 的主界面，它是在 Modelsim 启动时直接打开的，是其他所有窗口运行的基础。

在图 1.1 的主界面下有 3 个窗口：波形窗口（右侧窗口）、工作区窗口（左侧窗口）以及命令控制台（最下面的窗口）。

（1）波形窗口：主要作用是显示仿真波形。

（2）工作区窗口：主要作用是对当前工程的工作库以及所打开的数据集合进行控制。

（3）命令控制台：输入 Modelsim 的命令，并且可以将执行的结果反馈回来。

Modelsim 中还有信号窗口、结构窗口、源文件窗口、进程窗口、存储器窗口等，这些窗口将在设计实例中具体介绍。

1.1.2　测试向量

测试向量（testbench）是用来测试源代码的激励信号。源代码编好后，以一系列激

励信号作为源代码的输入，查看输出信号正确与否，进而验证源代码的正确性。本节对测试向量进行一个总结，并以一个具体的例子来讲解如何用测试向量对一个具体的项目用 Modelsim 进行仿真；如何对该项目使用 Synplify Pro 进行综合；约束如何设定；以及如何利用 Debussy 的特性来加快调试的进度。

测试向量就是源代码输入信号的激励，以及对源代码输出信号的响应。测试向量的主要作用是验证源代码的正确性。

测试向量的特点：

（1）测试向量没有输入、输出信号，只有激励和响应信号；

（2）测试向量可以使用不可综合语句进行设计；

（3）测试向量可以更多的从“C 语言”的角度来考虑激励信号的产生以及输出信号的响应，而不用过多思考硬件的结构；

（4）测试向量同样遵循 FPGA 开发中的模块化开发的原则；

（5）测试向量可以嵌入到源代码中，作为源代码仿真时的补充，如赋初值等。

下面将介绍如何对一个实例设计测试向量，并通过 Modelsim 进行仿真。

【例 1.1】双 CPU 接口及其测试向量。

两个 CPU 进行通信，由于接口不相同不能无缝对接，所以使用 FPGA 进行接口转换，两个 CPU 的接口时序图如图 1.2 和图 1.3 所示。

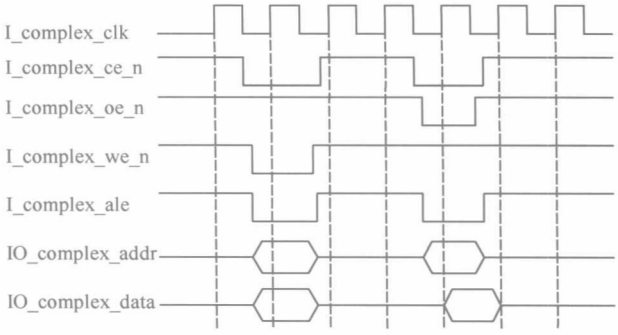

图 1.2　CPU1 接口时序图

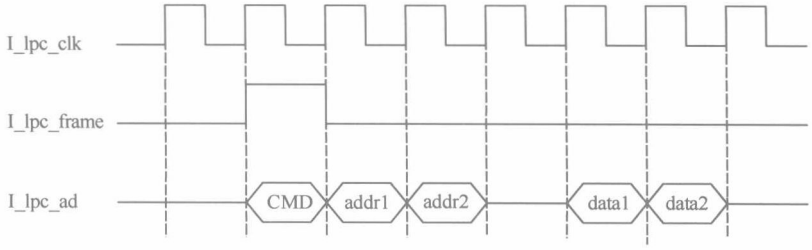

图 1.3　CPU2 接口时序图

（1）设计方案。将两个 CPU 的接口作为两个独立的模块，以双口 RAM 存储交换数据。

（2）模块说明。

top.v：调用两个子模块 cpu_data_addr_complex 和 cpu_lpc 以及 Xilinx 公司的 RAM 的 IP 核。

cpu_data_addr_complex.v：实现 CPU1 的接口时序，并将从 CPU1 接收的数据，转化为标准的双口 RAM 总线，传输给双口 RAM，并从双口 RAM 中读取 CPU2 传输过来的数据。

cpu_lpc.v：实现 CPU2 的接口时序，并将从 CPU2 接收的数据转化为标准的双口 RAM 总线，传输给双口 RAM，并从双口 RAM 中读取 CPU1 传输过来的数据。

dual_ram1.v：Xilinx 公司的双口 RAM 存储器的 IP 核。

dual_ram2.v：Xilinx 公司的双口 RAM 存储器的 IP 核。

（3）详细设计。

顶层模块

```
module     top(
    //====system reset====
    I_reset_n                    ,          //system    reset signal active low

    //====LOW PIN   COUNT    CPU interface====
    //====input====
    I_lpc_clk                    ,          //local bus clock signal 100MHz
    I_lpc_frame                  ,          //local bus write enable active low
    //====inout====
    IO_lpc_ad                    ,          //local bus data[15:0]

    //====CPU interface====
    //====input====
    I_complex_clk                ,          //cpu local bus clock signal
    I_complex_oe_n               ,          //cpu local bus output enable signal
    I_complex_we_n               ,          //cpu local bus write enable signal
    I_complex_ce_n               ,          //cpu local bus chip select 0
    I_complex_ale                ,          //cpu local bus address latch enable signal
    I_complex_addr               ,          //cpu local bus address
    IO_lbus_data                            //cup local bus data
);

//====input====
input       I_reset_n                ;
input       I_lpc_clk                ;
input       I_lpc_frame              ;
input       I_complex_clk            ;
input       I_complex_oe_n           ;
input       I_complex_we_n           ;
input       I_complex_ce_n           ;
```

```
input        I_complex_ale                ;
input[31:0] I_complex_addr                ;

//=====inout=====
inout[31:0] IO_lbus_data                  ;
inout[3:0]  IO_lpc_ad                     ;

//=====internal   register    define=====
wire[15:0]  W_complex_data_in             ;
wire[15:0]  W_complex_data_out            ;
wire        W_complex_we                  ;
wire[6:0]   W_complex_addr                ;
wire[7:0]   W_fifO_lpc_fifo_data          ;
wire[7:0]   W_lpc_fifo_data               ;
wire        W_lpc_we                      ;
wire[7:0]   W_lpc_addr                    ;
//----------------------------Main Body of Code-----------------------------
cpu_data_addr_complex       cpu_data_addr_complex_u(
      //=====system reset=====
      .I_reset_n              (I_reset_n              ),    //system reset signal active low

      //=====CPU       interface=====
      //=====input=====
      .I_complex_clk          (I_complex_clk          ),    //local bus clock signal
      .I_complex_oe_n         (I_complex_oe_n         ),    //local bus output enable signal
      .I_complex_we_n         (I_complex_we_n         ),    //local bus write enable signal
      .I_complex_ce_n         (I_complex_ce_n         ),    //local bus chip select 0
      .I_complex_ale          (I_complex_ale          ),    //local bus address latch enable signal
     .I_complex_addr          (I_complex_addr         ),    //local bus address

      //=====inout=====
      .IO_lbus_data           (IO_lbus_data           ),    //local bus data

      //=====to fifo interface=====
      //=====input=====
      .I_complex_data         (W_complex_data_in      ),    //from fifo data[15:0]

      //=====output=====
      .O_complex_data         (W_complex_data_out     ),    //to fifo data[15:0]
```

```
    .O_complex_we       (W_complex_we       ),    //to fifo write signal
    .O_complex_addr     (W_complex_addr     )     //read fifo data address[6:0]
    );

dual_ram1  dual_ram_u1(
    .dina       (W_lpc_fifo_data        ),
    .addrb      (W_complex_addr         ),
    .clkb       (I_complex_clk          ),
    .addra      (W_lpc_addr             ),
    .clka       (I_lpc_clk              ),
    .wea        (W_lpc_we               ),
    .doutb      (W_complex_data_in      )
    );

dual_ram2  dual_ram_u2(
    .dina       (W_complex_data_out     ),
    .addrb      (W_lpc_addr             ),
    .clkb       (I_lpc_clk              ),
    .addra      (W_complex_addr         ),
    .clka       (I_complex_clk          ),
    .wea        (W_complex_we           ),
    .doutb      (W_fifO_lpc_fifo_data   )
    );

 cpu_lpc     cpu_lpc_u(
    //====system reset====
    .I_reset_n              (I_reset_n          ),    //system reset signal active low

    //====CPU interface====
    //====input====
    .I_lpc_clk              (I_lpc_clk          ),    //local bus clock signal 100MHz
    .I_lpc_frame            (I_lpc_frame        ),    //local bus write enable active low

    //====inout====
    .IO_lpc_ad              (IO_lpc_ad          ),    //local bus data[15:0]

    //====fifo interface===
    //====input====
    .I_fifO_lpc_fifo_data   (W_fifO_lpc_fifo_data),   //from fifo data[15:0]
```

```
    //=====output=====
    .O_lpc_fifo_data    (W_lpc_fifo_data    ),    //to fifo data[15:0]
    .O_lpc_we           (W_lpc_we           ),    //to fifo write signal
    .O_lpc_addr         (W_lpc_addr         )     //read fifo data address[7:0]
    );
endmodule
```

子模块 cpu_data_addr_complex：

```
module    cpu_data_addr_complex(
    //=====system reset=====
    I_reset_n                ,    //system reset signal active low

    //=====CPU interface=====
    //=====input=====
    I_complex_clk            ,    //cpu local bus clock signal
    I_complex_oe_n           ,    //cpu local bus output enable signal
    I_complex_we_n,               //cpu local bus write enable signal
    I_complex_ce_n           ,    //cpu local bus chip select 0
    I_complex_ale            ,    //cpu local bus address latch enable signal
    I_complex_addr ,              //cpu local bus address
    //=====inout=====
    IO_lbus_data             ,    //cpu local data

    //=====to fifo interface=====
    //=====input=====
    I_complex_data ,              //from fifo data[15:0]

    //=====output=====
    O_complex_data ,              //to fifo data[15:0]
    O_complex_we   ,              //to fifo write signal
    O_complex_addr                //read fifo data address[6:0]
    );

//=====input=====
input       I_reset_n             ;
input       I_complex_clk         ;
input       I_complex_oe_n        ;
input       I_complex_we_n        ;
```