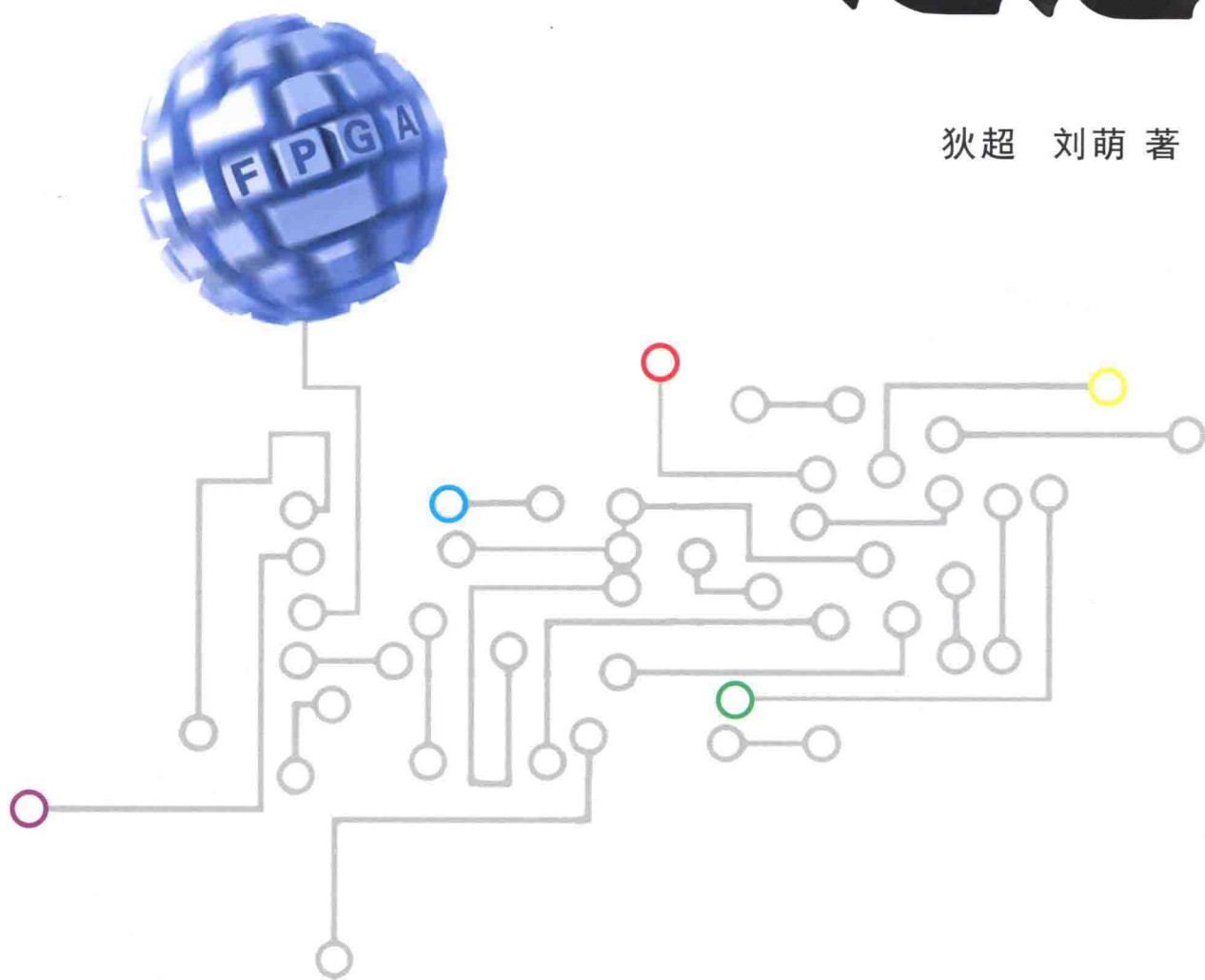


FPGA

之道

狄超 刘萌 著



西安交通大学出版社
XI'AN JIAOTONG UNIVERSITY PRESS

提供各种书籍的pd电子版代找服务，如果你找不到自己想要的书的pdf电子版，我们可以帮您找到，如有需要，请联系QQ1779903665.

PDF代找说明：

本人可以帮助你找到你要的PDF电子书，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签索引和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我QQ1779903665。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ，大部分我都可以找到，而且每本100%带书签索引目录。因PDF电子书都有版权，请不要随意传播，如果您有经济购买能力，请尽量购买正版。

声明：本人只提供代找服务，每本100%索引书签和目录，因寻找pdf电子书有一定难度，仅收取代找费用。如因PDF产生的版权纠纷，与本人无关，我们仅仅只是帮助你寻找到你要的pdf而已。

责任编辑：杨 璠 封面设计：任加盟

个人简介

狄超，籍贯陕西西安，生于1983年。

2001年免试保送至西安交通大学信息工程系就读本科，因成绩优异，于2005年保送至西安交通大学通信与信息系统专业就读研究生。毕业后，先后投身于航天及工业测控领域，至今已积累了十余年的FPGA相关项目经验，遂有感而发，写成了这本《FPGA之道》。值得一提的是，作者不光专注于如何使用FPGA，还经常尝试编写一些能够辅助FPGA项目开发的工具软件，其中的代表性作品就是一款集HDL代码的编辑器、语法检查器和编码规则检查器于一体的Quick HDL软件。除了FPGA之外，作者还对硬件电路设计、应用软件开发、网页设计与编程、游戏设计与开发、动漫设计与制作等领域有着极大的兴趣和广泛的涉猎，希望将一生都投入到技术与创造当中。



ISBN 978-7-5605-6189-9



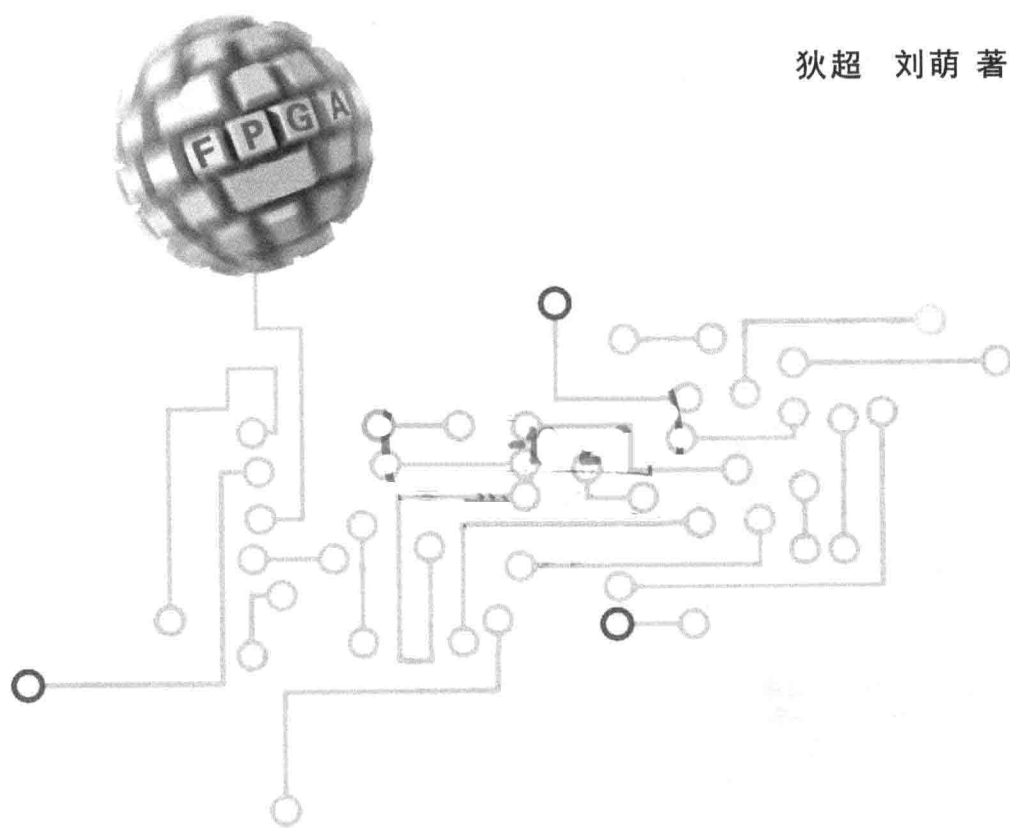
9 787560 561899 >

定价：338.00元

FPGA

之道

狄超 刘萌 著



 西安交通大学出版社
XI'AN JIAOTONG UNIVERSITY PRESS

内容简介

本书是一本针对 FPGA 技术进行全面、深入讲解的书籍,内容涵盖了数字电路相关基础理论的介绍、FPGA 芯片的构成及工作原理、FPGA 项目开发的全流程追踪、主流 HDL 与 HVL 语法的详细讲解与阐述等,尤其是针对 FPGA 项目开发流程中最为重要的三个环节——程序设计、功能仿真、时序分析——进行了深度剖析,通过丰富的思路阐述与实例介绍相结合的形式,力求让读者对 FPGA 技术能够“知其然”,且“知其所以然”。本书不仅是教材,更是工具书,适用于所有跟 FPGA 有缘的人阅读,无论你是本科生、研究生、老师或者工程师,无论你在校园、国企、外企或私企,只要在你的工作或生活中需要接触 FPGA,相信本书就是一本对你有用的书。本书是笔者从事 FPGA 事业十年经验的总结、两年心血的结晶,因此本书带给大家的绝不是空洞的口号,而是实实在在的 FPGA 技术。

图书在版编目(CIP)数据

FPGA 之道/狄超著. —西安:西安交通大学出版社,
2014. 5
ISBN 978-7-5605-6189-9

I. ①F… II. ①狄… 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(2014)第 094160 号

书 名 FPGA 之道
著 者 狄 超 刘 萌
责任编辑 杨 璠

出版发行 西安交通大学出版社
(西安市兴庆南路 10 号 邮政编码 710049)
网 址 <http://www.xjtupress.com>
电 话 (029)82668357 82667874(发行中心)
(029)82668315 82669096(总编办)
传 真 (029)82668280
印 刷 北京京华虎彩印刷有限公司

开 本 787mm×1092mm 1/16 印张 75.75 字数 1891 千字
版次印次 2014 年 8 月第 1 版 2014 年 8 月第 1 次印刷
书 号 ISBN 978-7-5605-6189-9/TP·617
定 价 338.00 元

读者购书、书店添货、如发现印装质量问题,请与本社发行中心联系、调换。
订购热线:(029)82665248 (029)82665249
投稿热线:(029)82669097 QQ:8377981
读者信箱:lg_book@163.com

版权所有 侵权必究

序

拿到本书样稿之时,惊讶之余,更多是惊喜。这本厚度堪比字典的关于 FPGA 设计的书,把 FPGA 入门、设计、验证的方法及过程详实严谨地展现在我们面前,细节部分的讲解,不乏作者的独到用心,全书提纲挈领而具有可操作性,相信对于广大读者有正确的引导作用。

该书作者在我们研究中心 3 年读研期间,其 FPGA 设计的才能已崭露头角,后经 2 年 FPGA 设计验证及 3 年 FPGA 专业开发工作的历练,已成为 FPGA 方面不可多得的专业人才,其学术水准在我们的研究生中间也颇具威信。

我问作者,作为一个不受论文、专著指标约束的“体制外”的人,为何耗费精力编写并出版这样的书?作者回答说:“回顾自己多年的 FPGA 设计经历,感觉大部分时间都花在一些细节处的思考和实践上,若是书中能够多些指点,开发效率自然会提高许多。写书的目的就是想把自己多年的经验、教训和感悟告诉后来者,让他们能少走弯路。”

这是一本有感而发的兴趣书,也是一本开卷有益的良心书!有幸得到的样稿,亦经被争相传阅,篇幅虽长,但处处都可圈可点。我认为,它应该成为从事 FPGA 设计研究生及工程师的手边书,有困惑难解之处,可以随时请教这位“良师益友”。

西安交大电信学院数据广播研究中心 沈雪峰

2013 年 9 月 25 日

目 录

1 与我同行	(1)
1.1 个人简介	(1)
1.2 “道”的含义	(1)
1.3 写作目的	(2)
1.3.1 技术人生,有总结才有提高	(2)
1.3.2 从入门到精通,带你走上 FPGA 之路	(2)
1.3.3 即是教材,也是参考	(3)
1.3.4 一起出生吧!	(3)
1.4 内容综述	(4)
1.5 默认共识	(5)
1.6 3,2,1,我们开始	(6)
2 共同语言	(7)
2.1 什么是 FPGA?	(7)
2.1.1 名词解释	(7)
2.1.2 发展简史	(7)
2.1.3 应用方向	(8)
2.2 模拟与数字	(10)
2.2.1 模拟的与数字的	(11)
2.2.2 模拟信号与数字信号	(11)
2.2.2.1 模拟信号的概念	(12)
2.2.2.2 数字信号的概念	(12)
2.2.2.3 模拟信号转换为数字信号	(13)
2.2.3 模拟电路与数字电路	(14)
2.2.3.1 电信号的本质	(14)
2.2.3.2 模拟电路简介	(15)
2.2.3.3 数字电路简介	(16)
2.2.4 模拟信号处理电路系统与数字信号处理电路系统	(17)
2.2.4.1 模拟信号处理电路系统	(17)
2.2.4.2 数字信号处理电路系统	(17)
2.2.4.3 模拟系统与数字系统的优缺点比较	(18)
2.2.4.4 模拟系统与数字系统之间的接口	(19)
2.2.4.5 数字系统之间的接口	(20)
2.3 数字逻辑电路基础知识	(24)

2.3.1	数与数制简介	(24)
2.3.1.1	常见数制简介	(24)
2.3.1.2	数制间的转换	(25)
2.3.1.3	二进制编码简介	(30)
2.3.2	数字逻辑的基本运算	(32)
2.3.2.1	集合与数字逻辑	(32)
2.3.2.2	数字逻辑的基本运算	(33)
2.3.2.3	真值表	(34)
2.3.2.4	数字逻辑运算的基本公式与规律	(35)
2.3.2.5	数字逻辑运算中的三个规则	(36)
2.3.3	数字逻辑的化简	(37)
2.3.3.1	公式化简法	(37)
2.3.3.2	卡诺图化简法	(38)
2.3.3.3	系统化简法	(46)
2.3.4	数字逻辑功能单元	(49)
2.3.4.1	基本逻辑功能单元简介	(50)
2.3.4.2	小规模集成组合逻辑单元简介	(57)
2.3.4.3	时序逻辑基本单元简介	(71)
2.3.5	数字逻辑的载体	(93)
2.3.5.1	半导体器件简介	(93)
2.3.5.2	CMOS 门电路举例	(98)
2.4	硬件描述语言	(102)
2.4.1	图形化设计方法简介	(102)
2.4.2	从图形化设计到 HDL	(103)
2.4.3	HDL 简史	(103)
2.4.3.1	VHDL 简史	(103)
2.4.3.2	Verilog HDL 简史	(104)
2.4.4	软件编程思路与 FPGA 编程思路的变革	(105)
2.4.4.1	名称对比分析	(105)
2.4.4.2	抽象层级对比	(107)
2.4.4.3	编译原理对比	(107)
2.4.4.4	执行方式对比	(108)
2.5	硬件扩展	(110)
2.5.1	电源转换芯片系列	(110)
2.5.2	数模转换芯片系列	(110)
2.5.3	数据存储芯片系列	(111)
2.5.4	微处理器芯片系列	(112)
2.6	最自由的开发者	(113)

3 知己知彼	(114)
3.1 可编程逻辑器件的原理架构	(114)
3.1.1 集成电路简介	(114)
3.1.2 PLA 原理架构	(115)
3.1.3 PAL 原理架构	(115)
3.1.4 GAL 原理架构	(116)
3.1.5 FPLA 原理架构	(118)
3.1.6 CPLD 原理架构	(119)
3.1.7 FPGA 原理架构	(119)
3.2 FPGA 产品介绍	(121)
3.2.1 Xilinx 公司 FPGA 产品简介	(121)
3.2.2 Altera 公司 FPGA 产品简介	(121)
3.2.3 其他公司 FPGA 产品简介	(121)
3.2.4 FPGA 产品的工业等级简介	(122)
3.2.5 FPGA 产品的速度等级简介	(122)
3.3 FPGA 内部资源介绍	(123)
3.3.1 逻辑资源块	(123)
3.3.2 时钟网络资源	(126)
3.3.3 时钟处理单元	(127)
3.3.3.1 PLL	(127)
3.3.3.2 DCM	(129)
3.3.4 BLOCK RAM	(130)
3.3.5 DSP	(130)
3.3.6 布线资源	(132)
3.3.7 接口资源	(133)
3.3.8 专用高速接口资源	(135)
3.3.9 To be continued	(135)
3.4 FPGA 芯片的供电	(135)
3.4.1 外部端口供电机制	(135)
3.4.2 内部逻辑供电机制	(136)
3.4.3 专有电路供电机制	(136)
3.4.4 电源稳定性讨论	(137)
3.5 FPGA 芯片的配置方法	(137)
3.5.1 主动配置模式	(137)
3.5.2 被动配置模式	(138)
3.5.3 JTAG 配置模式	(139)
3.5.4 应用模式简介	(139)
3.6 IP 核介绍	(140)
3.6.1 IP 核概述	(140)

3.6.2 嵌入式微处理器概述	(141)
3.7 组合逻辑电路与时序逻辑电路	(141)
3.8 同步逻辑电路与异步逻辑电路	(142)
4 开发流程	(145)
4.1 背景知识的分析与研究	(145)
4.2 项目方案的设计与制定	(146)
4.2.1 写清楚项目背景	(147)
4.2.2 写清楚项目需求	(147)
4.2.3 写清楚方案框架	(147)
4.2.4 写清楚算法细节	(148)
4.2.5 确保逻辑完备性	(148)
4.2.6 确保实现无关性	(148)
4.2.7 确保书面易懂性	(149)
4.3 算法可行性仿真与验证	(149)
4.3.1 Why	(149)
4.3.2 When	(151)
4.3.3 How	(151)
4.4 FPGA 设计方案的制定	(151)
4.4.1 编写 FPGA 设计方案的好处	(152)
4.4.2 如何编写 FPGA 设计方案	(152)
4.5 FPGA 功能代码的编写	(153)
4.6 FPGA 设计的功能仿真	(154)
4.7 FPGA 顶层模块的门级仿真	(156)
4.7.1 门级仿真简介	(156)
4.7.2 形式化验证简介	(157)
4.7.3 形式化验证在 FPGA 设计中的应用	(158)
4.8 FPGA 设计的用户约束	(159)
4.9 FPGA 设计的时序分析	(160)
4.10 FPGA 顶层模块的时序仿真	(163)
4.11 FPGA 设计的实现过程	(164)
4.11.1 编译概述	(165)
4.11.2 编译流程之综合	(165)
4.11.2.1 综合的输入	(165)
4.11.2.2 综合的输出	(168)
4.11.2.3 综合的工具	(169)
4.11.3 编译流程之翻译融合	(169)
4.11.3.1 翻译融合的输入	(169)
4.11.3.2 翻译融合的输出	(170)

4.11.3.3	翻译融合工具	(171)
4.11.4	编译流程之映射	(171)
4.11.4.1	映射的输入	(171)
4.11.4.2	映射的输出	(173)
4.11.4.3	映射工具	(174)
4.11.5	编译流程之布局布线	(174)
4.11.5.1	布局布线的输入	(174)
4.11.5.2	布局布线的输出	(176)
4.11.5.3	布局布线工具	(176)
4.11.6	编译流程之配置生成	(176)
4.11.6.1	配置生成的输入	(177)
4.11.6.2	配置生成的输出	(177)
4.11.6.3	配置生成工具	(178)
4.11.7	高级编译方法简介	(178)
4.11.7.1	Partition	(178)
4.11.7.2	手动布局布线	(181)
4.11.7.3	智能化实现指导	(182)
4.11.7.4	时序收敛方案	(182)
4.12	开发板制作	(182)
4.13	FPGA 设计的上板调试	(183)
4.13.1	“实践是检验真理的唯一标准”	(183)
4.13.2	如何解决问题	(183)
4.13.3	如何找到问题	(185)
4.13.4	“时间是检验真理的第二标准”	(186)
4.14	项目总结 备份与后期维护	(187)
4.15	开发流程中的团队分工	(187)
5	程序设计	(189)
5.1	编程语法	(189)
5.1.1	VHDL 基本语法	(189)
5.1.1.1	VHDL 基本程序框架	(189)
5.1.1.2	VHDL 注释语法	(198)
5.1.1.3	VHDL 的 signal、variable 与 constant	(199)
5.1.1.4	VHDL 数据类型	(200)
5.1.1.5	VHDL 初始化	(206)
5.1.1.6	VHDL 的操作符号	(207)
5.1.1.7	VHDL 的并行语句	(215)
5.1.1.8	VHDL 的串行语句	(225)
5.1.1.9	编写自己的 VHDL 库文件	(234)

5.1.1.10	VHDL 编写注意事项	(240)
5.1.2	Verilog 基本语法	(246)
5.1.2.1	Verilog 基本程序框架	(246)
5.1.2.2	Verilog 注释语法	(253)
5.1.2.3	Verilog 数据类型	(254)
5.1.2.4	Verilog 初始化	(260)
5.1.2.5	Verilog 的操作符号	(261)
5.1.2.6	Verilog 的并行语句	(271)
5.1.2.7	Verilog 的串行语句	(285)
5.1.2.8	Verilog 中的编译指令	(294)
5.1.2.9	Verilog 中的编写注意事项	(296)
5.1.3	VHDL 与 Verilog 的比较	(301)
5.1.3.1	语法比较	(301)
5.1.3.2	语言比较	(304)
5.2	编程思路	(305)
5.2.1	代码风格	(306)
5.2.1.1	代码风格四步走	(306)
5.2.1.2	HDL 选择	(306)
5.2.1.3	命名规则	(307)
5.2.1.4	语法结构	(311)
5.2.1.5	空格空行	(316)
5.2.1.6	注释编写	(317)
5.2.1.7	模块设计	(320)
5.2.1.8	文件结构	(324)
5.2.2	三种描述方式	(324)
5.2.2.1	结构化描述方式	(324)
5.2.2.2	数据流描述方式	(326)
5.2.2.3	行为级描述方式	(328)
5.2.3	顶层设计模块	(330)
5.2.3.1	概念浅析	(330)
5.2.3.2	特点说明	(331)
5.2.4	设计的分类	(332)
5.2.4.1	按功能分	(332)
5.2.4.2	按面向分	(332)
5.2.4.3	按速度和规模分	(333)
5.2.5	编写纯净的组合或时序逻辑	(334)
5.2.5.1	组合逻辑描述方法	(334)
5.2.5.2	纯时序逻辑描述方法	(336)
5.2.5.3	清晰的时序逻辑描述方法	(338)

5.2.5.4	有隐患的混写逻辑	(340)
5.2.6	正确的变量访问思路	(345)
5.2.6.1	变量访问思路概述	(345)
5.2.6.2	赋值冲突	(348)
5.2.6.3	总线是怎么回事	(353)
5.2.7	数字电路中的隐患	(361)
5.2.7.1	寄存器输出的不稳定态	(361)
5.2.7.2	组合逻辑中的竞争与险象	(369)
5.2.8	时钟及时钟域	(385)
5.2.8.1	时钟,时序逻辑的心跳.....	(385)
5.2.8.2	时钟信号基本特征	(385)
5.2.8.3	时钟信号的分类	(386)
5.2.8.4	时钟域	(389)
5.2.8.5	跨时钟域问题	(394)
5.2.9	DCM 与 PLL	(403)
5.2.9.1	PLL 模块基本端口简介	(403)
5.2.9.2	DCM 模块基本端口简介	(406)
5.2.9.3	应用场合	(409)
5.2.10	复位的设计.....	(416)
5.2.10.1	为什么 FPGA 设计中要有复位	(416)
5.2.10.2	复位方式的分类.....	(417)
5.2.10.3	复位的设计方法.....	(418)
5.2.11	数据的存储.....	(429)
5.2.11.1	为什么需要数据存储.....	(429)
5.2.11.2	数据存储的载体.....	(429)
5.2.11.3	数据存储的形式、实现及应用场合	(430)
5.2.11.4	寄存器的 HDL 描述与用法	(433)
5.2.11.5	单口 RAM 的 HDL 描述与用法	(434)
5.2.11.6	双口 RAM 的 HDL 描述与用法	(441)
5.2.11.7	ROM 的 HDL 描述与用法	(443)
5.2.11.8	同步 FIFO 的 HDL 描述与用法	(446)
5.2.11.9	异步 FIFO 的 HDL 描述与用法	(448)
5.2.11.10	FIFO 使用小技巧之冗余法	(461)
5.2.11.11	STACK 的 HDL 描述与用法	(465)
5.2.11.12	外部存储芯片的 HDL 描述与用法	(467)
5.2.11.13	数据存储的使用思路	(467)
5.2.12	状态机,FPGA 的灵魂	(469)
5.2.12.1	状态机的概念.....	(469)
5.2.12.2	状态机的模型.....	(472)

5.2.12.3	状态机的设计	(477)
5.2.12.4	状态机的 HDL 模板	(492)
5.2.12.5	状态的编码方式	(511)
5.2.12.6	状态机的实现方式	(513)
5.2.12.7	显式状态机与隐式状态机	(517)
5.2.13	关于外界接口的编程思路	(523)
5.2.13.1	按传递方向分类	(523)
5.2.13.2	按电气特性分类	(528)
5.2.13.3	按功能特性分类	(531)
5.2.13.4	按时钟特性分类	(532)
5.2.13.5	按数据位宽分类	(551)
5.2.13.6	按速度分类	(555)
5.2.14	数的表示与数的运算	(557)
5.2.14.1	FPGA 中数的存储形式	(557)
5.2.14.2	整数在 FPGA 中的表示形式	(557)
5.2.14.3	整数的加、减运算	(563)
5.2.14.4	整数的乘法运算	(572)
5.2.14.5	整数的除法运算	(651)
5.2.14.6	整数的其他类型运算简介	(667)
5.2.14.7	非整数运算浅析	(674)
5.2.15	时空变换	(684)
5.2.15.1	时空变换之基本概念	(684)
5.2.15.2	时空变换之时域优化	(686)
5.2.15.3	时空变换之空域优化	(702)
5.2.15.4	时空变换之时间换空间	(707)
5.2.15.5	时空变换之空间换时间	(719)
5.2.16	“万能”的查表法	(732)
5.2.16.1	正弦波发生器示例	(732)
5.2.16.2	Gamma 校正示例	(735)
5.2.17	代码中的约束信息	(741)
5.2.17.1	为什么要在 HDL 中加入约束信息	(741)
5.2.17.2	HDL 中的常用约束示例	(741)
5.2.18	原语的使用	(756)
5.2.18.1	什么是原语	(756)
5.2.18.2	需要使用原语的情况	(759)
5.2.18.3	用原语表示 IP 核的好处	(763)
5.2.18.4	UDP 简介	(767)
5.2.19	提高设计的综合性能	(768)
5.2.19.1	提高设计的鲁棒性	(768)

5.2.19.2	提高设计的自测性	(775)
5.2.19.3	提高设计的重用性	(790)
5.2.19.4	提高设计的易改性	(790)
5.2.19.5	提高设计的移植性	(806)
5.2.19.6	提高设计的保密性	(807)
5.2.20	设计方法学讨论	(808)
5.2.21	FPGA 程序设计的境界	(809)
6	功能仿真	(811)
6.1	仿真原理	(811)
6.1.1	串行模仿并行思路分析	(811)
6.1.2	有限模仿无限思路分析	(812)
6.1.3	组合逻辑仿真原理	(812)
6.1.4	时序逻辑仿真原理	(813)
6.1.5	HDL 的仿真原理	(813)
6.1.6	仿真时间与物理时间	(814)
6.2	仿真语法	(815)
6.2.1	Graphic Waveform	(815)
6.2.1.1	数字波形简介	(815)
6.2.1.2	从实际到仿真	(817)
6.2.1.3	“Hello world”之 Graphic Waveform	(818)
6.2.1.4	一些绘制波形的操作	(818)
6.2.1.5	一些观察波形的操作	(820)
6.2.1.6	波形仿真结果分析及重要注意事项	(820)
6.2.2	VHDL Test Bench	(822)
6.2.2.1	“Hello World”之 VHDL Test Bench	(822)
6.2.2.2	继承描述语法	(825)
6.2.2.3	时间相关语法	(825)
6.2.2.4	时钟激励语法	(827)
6.2.2.5	循环仿真语法	(829)
6.2.2.6	进程语句体	(831)
6.2.2.7	文件操作语法	(832)
6.2.3	Verilog Test Fixture	(837)
6.2.3.1	“Hello world”之 Verilog Test Fixture	(837)
6.2.3.2	继承描述语法	(839)
6.2.3.3	时间相关语法	(839)
6.2.3.4	时钟激励语法	(843)
6.2.3.5	循环仿真语法	(846)
6.2.3.6	程序块语句体	(847)

6.2.3.7	字符显示语法	(848)
6.2.3.8	系统存储载入函数	(849)
6.2.3.9	系统文件操作函数	(850)
6.2.3.10	系统屏幕输出函数	(855)
6.2.3.11	系统随机数生成函数	(855)
6.2.3.12	系统仿真时间函数	(857)
6.2.3.13	系统仿真进度控制任务	(858)
6.2.4	System Verilog SVT	(859)
6.2.4.1	“Hello World”之面向过程 SV Test	(859)
6.2.4.2	“Hello World”之面向对象 SV Test	(868)
6.2.4.3	测试平台相关语法	(878)
6.2.4.4	System Verilog 的数据类型	(894)
6.2.4.5	System Verilog 的数组类型	(906)
6.2.4.6	System Verilog 的操作符号	(920)
6.2.4.7	System Verilog 的过程语句	(922)
6.2.4.8	System Verilog 的进程语句	(930)
6.2.4.9	System Verilog 的任务和函数	(930)
6.2.4.10	System Verilog 的线程及线程间通信	(938)
6.2.4.11	System Verilog 的系统函数、任务语法	(960)
6.2.4.12	System Verilog 的面向对象编程语法	(969)
6.2.4.13	System Verilog 的随机化语法	(995)
6.2.5	Assertion	(1016)
6.2.5.1	Why Assertion?	(1017)
6.2.5.2	SVA 分类	(1017)
6.2.5.3	SVA expression	(1023)
6.2.5.4	SVA sequence	(1026)
6.2.5.5	SVA property	(1033)
6.2.5.6	SVA assert	(1035)
6.2.5.7	并发断言进阶	(1036)
6.3	仿真环境	(1048)
6.3.1	待测设计准备	(1049)
6.3.1.1	寻找所需的模块	(1049)
6.3.1.2	寻找所需的文件	(1050)
6.3.1.3	Clean 工程的概念	(1051)
6.3.2	仿真库文件准备	(1051)
6.3.2.1	建立 Xilinx 平台相关的仿真库	(1052)
6.3.2.2	建立 Altera 平台相关的仿真库	(1069)
6.3.3	仿真平台的结构设计	(1069)
6.3.4	仿真平台的变形	(1070)

6.3.4.1	直接观察波形法	(1071)
6.3.4.2	用多仿真平台替换任务生成模块	(1071)
6.3.4.3	用参考设计替换记分板模块	(1071)
6.3.4.4	添加逆变换模块	(1073)
6.3.4.5	第三方器件模拟	(1074)
6.4	仿真思路	(1075)
6.4.1	如何围绕 FPGA 项目展开仿真	(1075)
6.4.1.1	仿真介入的时机	(1075)
6.4.1.2	自底向上的仿真	(1076)
6.4.1.3	自顶向下的仿真	(1076)
6.4.1.4	回归测试	(1077)
6.4.2	仿真模块的实现	(1077)
6.4.3	测试用例的分类	(1078)
6.4.4	定位问题的手段浅析	(1079)
6.4.4.1	找出所有的问题	(1079)
6.4.4.2	如何具体进行问题排查	(1080)
6.4.4.3	如何加速问题的出现	(1081)
6.5	仿真覆盖率	(1081)
6.5.1	覆盖率浅析	(1081)
6.5.2	代码覆盖率简介	(1082)
6.5.2.1	代码覆盖率的种类	(1083)
6.5.2.2	代码覆盖率的示例	(1086)
6.5.3	功能覆盖率简介	(1096)
6.5.3.1	覆盖组简介	(1097)
6.5.3.2	覆盖组数据的收集规则	(1102)
6.5.3.3	覆盖组的选项	(1119)
6.5.3.4	在仿真中进行覆盖率统计	(1123)
6.5.4	覆盖率的合并	(1124)
7	时序分析	(1128)
7.1	基本概念介绍	(1128)
7.1.1	常用时间参数介绍	(1128)
7.1.2	线延迟与门延迟	(1129)
7.1.3	影响延迟的因素	(1130)
7.1.3.1	温度与电压	(1130)
7.1.3.2	三种工况	(1131)
7.1.4	时钟信号的偏差描述	(1132)
7.2	时序分析的原理	(1133)
7.2.1	一道时序分析的例题	(1133)

7.2.1.1	解答一:能否正确工作分析	(1134)
7.2.1.2	解答二:最大时钟速率分析	(1136)
7.2.1.3	延伸二:最小时钟速率?	(1137)
7.2.1.4	解答三:保持时间不足情形分析	(1138)
7.2.2	同步时序逻辑的分析原理	(1138)
7.2.2.1	逻辑锥的概念	(1138)
7.2.2.2	逻辑锥的划分	(1138)
7.2.2.3	逻辑锥的求解	(1139)
7.2.3	其他常见逻辑情形的应对方法	(1143)
7.2.3.1	内部时钟相关时序分析	(1143)
7.2.3.2	外部接口相关时序分析	(1151)
7.3	常用时序约束介绍	(1167)
7.3.1	时序环境约束	(1168)
7.3.2	分组时序约束	(1168)
7.3.3	常用时序约束	(1169)
7.3.2.1	周期约束	(1169)
7.3.2.2	输入约束	(1171)
7.3.2.3	组间约束	(1175)
7.3.2.4	输出约束	(1177)
7.4	根据时序报告修改设计	(1179)
7.4.1	时序分析报告示例	(1179)
7.4.1.1	待分析设计	(1179)
7.4.1.2	时序约束文件	(1180)
7.4.1.3	时序报告简介	(1181)
7.4.2	常见问题及修改	(1189)
7.4.2.1	可以不改的一些问题	(1189)
7.4.2.2	常见时序收敛的手段	(1189)
8	梦想启航	(1193)
8.1	你懂了吗?	(1193)
8.2	趁热打铁吧	(1193)
8.3	常回来看看	(1193)
8.4	共同成长	(1193)
8.5	祝你好运	(1194)
8.6	致谢	(1195)
参考文献		(1196)

1 与我同行

1.1 个人简介

笔者是陕西西安人,本科与研究生都就读于西安交通大学通信与信息系统专业,毕业后先后辗转于北京、上海、西安三地,期间一直从事 FPGA 相关项目的开发与验证工作。截止到写本书之日,笔者接触 FPGA 共有将近八年的时间。而在这八年的时间里,笔者通过不断的学习和项目的磨练,在 FPGA 技术方面也终有一定积累,拥有了比较丰富的项目经验,掌握了一定的学习和科研方法,也总结了不少技巧。当然,学海无涯,技术的发展也是无止境的,笔者在今后的时间里仍然会继续沿着 FPGA 的道路坚持下去!

1.2 “道”的含义

本书名为——《FPGA 之道》,这其中最关键的就是一个“道”字,它有三层含义:一、道路;二、方法;三、精神。这也是贯穿本书始终的三大主线。

道路,是本书的第一条主线。它指的是 FPGA 的技术之路,主要是指笔者从事 FPGA 事业的这八年所走过的道路,感受到的风景,体会到的滋味。其次,它指的也是此时此刻正在阅读本书的“你”的道路,作为本书的读者,想必你也一定是有着 FPGA 相关的项目要做,或者未来打算做 FPGA 相关的项目吧。那么,从无到有,从不懂到精通,从搭建软、硬件环境到成功拿下项目,这之间都是需要我们老老实实、一步一个脚印的沿着 FPGA 技术之路走下来。只要坚持不懈,走得越远,收获就会越大,万里长征路漫漫,请务必加油,大家共勉!

方法,是本书的第二条主线。它指的是 FPGA 的方法学,这其中包括学习 FPGA 的方法和使用 FPGA 的方法。虽然俗话说“勤能补拙”、“笨鸟先飞”,但是如果飞的方向错了,那么可能永远也到达不了目的地,并且悲观情绪的慢慢滋生很可能会令你丧失继续坚持下去的勇气。因此,光靠使蛮劲、耗时间是不一定能学好 FPGA 的,掌握必要的、科学的、可行的方法,是助你在 FPGA 技术之路上少走弯路的必要条件!

精神,是本书的第三条主线。它指的是一名沿着 FPGA 技术之路勇往直前的奋斗者的境界和觉悟!这是一种求知欲的表达和一份对 FPGA 技术的热爱!只有达到这样一种精神层面,才能在当前这个纷繁复杂、物欲横流的世界中找到属于自己的一片“净土”。而只有这片“净土”,才是能够让你静下心来、解放思想、发散思维、不断创新的源泉之地。因此,这也应该是每一个在 FPGA 道路上的长征的“你”所追求的境界!

1.3 写作目的

1.3.1 技术人生,有总结才有提高

不知不觉中,已和 FPGA 打交道近八个年头,在这期间,时而挖空心思、机关算尽、无计可施,时而柳暗花明、醍醐灌顶、才思泉涌。八年的时间,抗战都胜利了,只要智商还算正常,都会有所斩获!但是,时间在慢慢侵蚀我的大脑,随着接触的越多、知道的越多,忘记的也就越多。我们总不能在每次想要用“余弦定理”解决平面几何问题的时候都去重新推导一遍吧?因此,为了让自己不要成为那只“掰苞米的猴子”,我决定从现在开始对我所经历的 FPGA 之“道”进行详细的总结。总结可以让我对 FPGA 的理解更加清晰,因为它可以帮助我纠正错误的臆断、巩固正确的概念,为我清除混淆视听的障碍;总结可以让我对 FPGA 的理解更加迅速,因为它可以帮助我将一些工程技巧与技术经验提炼成方法、规律,方便我随时查阅;总结还可以让我对 FPGA 的理解继续深入,因为它可以帮我打好坚实的技术基础,为我以后在 FPGA 技术大厦上加砖添瓦奠定基础。因此,对于一名像我这样打算以毕生精力从事技术的工程师而言——“技术人生,有总结才有提高”。

1.3.2 从入门到精通,带你走上 FPGA 之路

FPGA 设计所涉及的知识比较多。例如:

就 FPGA 本身来说,它是一款高集成度的数字芯片,要想搞懂它到底是怎么回事,就离不开数字电路、CMOS、超大规模集成电路等的知识;

就 FPGA 应用的硬件系统来说,要想做好 FPGA 设计,就离不开对它周围模拟电路、数字电路、存储芯片、微处理芯片、各种接口电路的学习和了解;

就 FPGA 前期的可行性验证而言,要想确认设计思路是否可行,最终能否达到期望的功能,就离不开对各类软件平台和软件编程语言的学习;

就 FPGA 的设计方法而言,要想做好 FPGA 设计,就离不开对各种硬件描述语言及开发平台的学习;

就 FPGA 的验证方法而言,要想多多找出 FPGA 设计中的 bug,达到预期目标,就少不了对各种仿真语言、仿真环境及相关分析、验证的思路和方法的学习;

就 FPGA 的具体应用领域而言,要想让自己的 FPGA 设计能够实现某一领域的相关要求,就少不了对相关领域的知识的学习,例如图像领域、通信领域、控制领域;

.....

总之,FPGA 涉及的面较广、也较杂,所以学习的时候不能太过盲目,否则很容易还未跨进门槛就已经想要放弃,或者浪费大量的时间在漫无头绪的探索上而导致得不偿失。因此,为了让初学者能够正确地入门,让有一定 FPGA 设计经验的朋友能够进一步提高自己对 FPGA 的认识,也为了能和一些在 FPGA 设计方面经验比较丰富的朋友找到共鸣、共同探讨、共同进步,遂写此书!希望她能够陪伴你从入门到精通,并肩走上这条 FPGA 技术之路!

本书所带给你的,不是游离于底层硬件的 FPGA 开发,也不是应付作业、应付毕设的程序设计速成,更不是千篇一律的开发工具使用手册,而是一种让你能够真正的了解 FPGA,真正

的认识 FPGA,真正的懂得 FPGA 的思考方式! 掌握技巧,掌握方法,掌握思想,掌握精髓! 从而能够在日后的 FPGA 开发中做到游刃有余,能够在编写程序时通过代码看到门电路,能够在与人讨论时做到深入浅出、画龙点睛,能够在分析问题时透过现象看到本质!

1.3.3 既是教材,也是参考

在我这并不算短的 FPGA 技术生涯中,受过很多有丰富经验的老师和前辈们的指导,真是非常感谢! 但更多的时候,钻研技术还只能靠自己,因此这些年来也走过一些弯路,也曾经迷茫过、彷徨过,甚至有时候想过要放弃。不过还好,我没有半途而废,而是一直坚持着“以 FPGA 技术为中心,以外围硬件和软件的研究、学习为基本点”的方针走了下去,直到现在。

有人问我,说他需要做 FPGA 相关的项目,但是没有基础,该怎么办? 需要学些什么? 说实话,这个问题着实难回答,虽然我与 FPGA 经历了“八年抗战”,但是却很少会考虑该怎么样去教别人学 FPGA。我回忆了一下之前学过的相关基础理论课程以及看过的相关参考书籍,由于知识点太多,技术点太杂,一时半会真的很难从中抽象出一套合理的、系统的研习脉络,于是我萌发了写这本书的想法,希望它能给想学习 FPGA 或者需要学习 FPGA 的人以指引,能够真正地帮助到那些在“路上”的人。因此,我希望它首先是一本教材!

但是,我更希望它不仅仅是一本教材,因为无论是对于我自己还是以后看这本书的读者,我都希望这本书还能成为大家案头的一本参考书、工具书,因为这里面我不但会讲一些关于 FPGA 的原理,也会讲很多具体的 FPGA 应用和比较详细的相关编程语法等。因此,当你以后在“路上”碰到一些问题时,例如某一个功能语法记不清楚了或者某一种逻辑结构忘记了,这时候你就可以马上翻开这本书,并且很快地从中找出答案,这就是我想要达到的效果。

1.3.4 一起出生吧!

本书起笔于 2011 年 6 月 12 日,而我的妻子也正值此时怀有身孕。十月怀胎很辛苦,为了我们的孩子,我的妻子需要在此期间付出很多艰辛,承受很多压力,而作为孩子的父亲,我也很希望能够为孩子多做些什么,尤其是一些有意义的事情。于是我决定写这样一本书作为送给即将出世孩子的礼物,而我的妻子也将负责本书的部分工作,希望他们能够一起出生!

写作并非易事,尤其对我这样一个不善于言谈的人来说更是这样。设计好基本目录后,我就感觉至少得 60 万字,还不算插图。最开始写的小节是“VHDL 基本语法”,大概花去近两周业余时间,3 万多字,着实写得很累。而其他部分写作难度更大,并且要写的内容非常之多,由于我只能利用业余时间来写作,所以无形中多了很大的压力。我曾想删去一些章节,减少成书时间,但想来想去,对于哪一部分都割舍不得,我感觉既然书名叫做《FPGA 之道》,就不能做这种偷工减料的事情,并且如果只挑容易的、方便的来写,也是很不负责任的一种表现。于是只好硬着头皮继续创作,可是复杂的内容和膨胀的字数有时真的令我有些丧失斗志,而更加郁闷的是,此时我的宝宝也出现了让人担忧的状况,医生说有先兆流产的症状,和妻子商量后,我们决定先保胎,但是心里也都没底。就这样在惶惶中度过了两周,宝宝的状态仍没有明显好转,于是,我给妻子打气说:“不要担心,要相信我们的宝宝,我们的宝宝不会流产! 我们的书也不会!”

写下这一段的时候,时间是 2011 年 8 月 13 日,本书已完成了 8 万多字。此时我们宝宝的状况也已经好转,但愿一切都顺顺利利的,我会坚持写作,争取让他们一起出生!!!

1.4 内容综述

本书共分为八个篇章,按照由浅入深、自顶向下的思路逐渐展开,各篇章的内容概括如下:

第一篇——与我同行

该篇是本书的开场篇,通过阅读本篇,可以对本书的主旨、内容和写作背景有一个概括性的了解。

第二篇——共同语言

“不积跬步,无以至千里。”学习的过程就好比盖高楼一样,地基的稳固程度直接决定你能达到的高度。要想学好 FPGA,道理是一样的,也必须扎扎实实一步一步的来。而通过阅读本篇,可以让你认识 FPGA、了解 FPGA,并且可以让你了解 FPGA 所基于的科学理论基础知识,从而在后续的章节中做到“知其然,更知其所以然”。

第三篇——知己知彼

古语有云:“知己知彼,百战不殆!”要想驯服 FPGA,让它乖乖地听你的话,实现你所期望的功能,就不能盲目地只去学习一些相关编程语法或者科学理论。在不断提升不断认识自己的过程中,一定也要去了解一些 FPGA 的工作原理、基本架构和相关技术细节。只有这样,才能在与 FPGA 的“抗战”中扬长避短、事半功倍,从而取得“最终的阶段性”胜利!因此本篇较上一篇而言,对于 FPGA 的阐述更具体,更深入,更有针对性。

第四篇——开发流程

经过了《共同语言》与《知己知彼》的预热,我们将在本篇为大家揭秘一个相对全面、科学的 FPGA 项目开发基本流程,并对开发基本流程中的每一个环节进行较为详细的讲解与分析。因此,通过阅读本篇,可以规范化 FPGA 项目的开发,并且可以纠正大家对“FPGA 开发就是编写 HDL 代码”的狭隘认识。阅读本篇无论是对个人开发还是团队协作,都会有一定的帮助。与接下来的几篇相比,本篇好比在讲如何做一名“军师”,通过习得百家兵法从大局出发取得胜利;而后续几个专业篇好比在讲如何成为一名“武林高手”,通过习得一身好武艺面对面地击倒敌人。

第五篇——程序设计

该篇是本书的第一个专业篇,也是最重要、最基础的一个专业篇。在本篇我们将会为大家详细介绍两种当前最流行的 FPGA 开发语言——VHDL 与 Verilog。并且更重要的是,本篇还将为大家介绍大量的、详细的编程思路和一些具体的编程技巧,从而做到不仅“授人以鱼”,更“授人以渔”。

第六篇——功能仿真

该篇是本书的第二个专业篇,其中的内容非常重要,从《开发流程》来看,其代码量和时间消耗可能比《程序设计》的分量还要大,但本篇的内容与最终的 FPGA 硬件实现却几乎毫无关系。这就是《功能仿真》,一个可以确保《程序设计》FPGA 硬件程序功能上正确性、稳定性和可靠性的重要环节。

第七篇——时序分析

该篇是本书的第三个专业篇,其中的内容也是非常重要的。通过阅读本篇,可以确保《程序设计》FPGA 硬件程序在转换为实际芯片电路时不出时序问题。从而,通过与《功能仿真》的

“强强联合”，可以减少在最终上板调试时可能碰到的问题，从而保证项目最终的成功。

第八篇——梦想起航

该篇是本书的收尾篇，如果你一页一页地看到这一篇，那么首先要感谢你选择了本书，真的非常感谢！其次，我相信你也一定已经成为可以独当一面的 FPGA 从业者。那么，祝福的话就不多说了，期待你在今后的日子里能够活用本书所讲的知识，慢慢做到理解本书、超越本书甚至批判本书！

1.5 默认共识

由于本书内容较多，为了方便起见，采用以下默认共识，请读者注意：

1. 共识一

有些时候在一个小节里面讲到的内容可能需要引用另一小节中的知识来解释或参考。为此，本书适用统一的章节引用语法来统一全书的章节引用风格。章节引用语法如下：

【篇名→1 级小标题→2 级小标题→3 级小标题→…】

例如：

请参考【程序设计→编程语法→VHDL 基本语法→VHDL 注释语法】小节，那么我们可以非常方便地通过目录定位被引用章节的位置，从而快速查阅到需要使用到的内容。

如果同时引用一个大章节下的若干个小节，语法如下：

【篇名→…→ n 级小标题】→【 $n+1$ 级小标题 1】、【 $n+1$ 级小标题 2】、…

例如：

请参考【程序设计→编程思路】→【时空变换之基本概念】、【时空变换之时间换空间】、【时空变换之空间换时间】三个小节，表示同时参考【程序设计→编程思路】章节下的三个小节内容。

如果在一个三级小标题的内容中引用位于同一个二级小标题下的另一个三级小标题，那么可以直接通过如下语法来说明引用：

【另一个 3 级小标题】

5. 共识二

在介绍各种程序语法时，对于语法中非语法关键字的部分，采用替换语法表示，意思即在实际的程序编写中，此部分需要根据情况来确定。替换语法如下：

<替换说明>

例如 Verilog 中的变量声明语法之一：

```
reg<range><variable_name>;
```

其中，第一个替换语法中的说明是范围，第二个替换语法中的说明是变量名称，因此，在实际使用中，如果要声明一个 8 位的寄存器变量 a，可以写成：

```
reg [7:0] a;
```

等等。

1.6 3,2,1,我们开始

从下面开始,你就要踏上 FPGA 的征程了。你应该高兴在自己需要学习 FPGA 的时候能够碰上这样一本书,但是这并不代表后面的路你就会走得一帆风顺。FPGA 的技术道路并不平坦,一路上也是布满荆棘,因此你需要有很强的毅力和解决问题的能力来坚持下去。技术无止境,而通往 FPGA 技术之巅的路也没有尽头。所以,我们的目的不是终点,也没有终点,而是尽自己最大的努力走得更远。这一路上,不要轻言放弃,付出的努力总会有回报。那么,如果你已做好准备,请深吸一口气,跟我一起在心中默默倒数——3,2,1,开始!然后,翻到下一页起始你的征程!

2 共同语言

你总得知道点什么吧,否则我们怎么交流呢?

2.1 什么是 FPGA?

2.1.1 名词解释

FPGA,英文全称 Field-Programmable Gate Array,翻译成中文即现场可编程门阵列。它是一种主要以数字电路为主的集成芯片,属于可编程器件中的一种。FPGA 作为 ASIC(专用集成电路)领域中的一种半定制电路而出现,既解决了定制电路缺乏灵活性的不足,又克服了原有可编程器件门电路数容量过小的缺点,因此,在业界得到了广泛的应用。

2.1.2 发展简史

在还没有发明出可编程逻辑器件(PLD;Programmable Logic Device)之前,设计师们只能使用一些专用的小芯片来搭建系统,这些小芯片被称做离散逻辑芯片。每一种离散逻辑芯片实现的功能也就相当于几个逻辑门,例如几个逻辑与门等。因此,为了实现略微复杂一点的逻辑功能,设计者们可能需要同时使用几十个或更多这样的离散逻辑芯片来完成。这会使得电路板的布局、布线难度极大地增加,并且会极大地影响系统的性能。

为了改变这种状况,20 世纪 70 年代早期出现了第一类可编程逻辑器件——PLA。PLA 英文全称 Programmable Logic Array,翻译成中文即可编程逻辑阵列。PLA 中包含了一些固定数量的与门、非门,它们分别组成了“与平面”和“或平面”,以及仅可编程一次的连接矩阵(因为编程基于的是熔丝工艺),即“与连接矩阵”和“或连接矩阵”,因此可以实现一些功能相对复杂的与、或多项表达式的逻辑。

在 PLA 中,如果只有与门的连接矩阵可编程,而或门的连接矩阵是硬件固定的,那么这种芯片叫做 PAL,即可编程阵列逻辑,英文全称 Programmable Array Logic。

在 PAL 的基础上,又发展出了一种叫 GAL 的器件,英文全称 Generic Array Logic,翻译成中文即通用阵列逻辑。它相比于 PAL 有两点改进:①采用了电可擦除的 CMOS 工艺,极大增强了器件的可重配置性和灵活性;②采用了可编程的输出逻辑宏单元 OLMC(Output Logic Macro Cell),通过编程 OLMC 来将 GAL 的输出设置成为不同的状态,从而达到了仅用一个型号的 GAL 就可以实现所有 PAL 器件的输出电路工作模式,从而增强了器件的通用性。

随着编程工艺的更新,在 PLA 的基础上又发展出了 FPLA,英文全称 Field Programmable Logic Array,翻译成中文即现场可编程逻辑阵列。它相比于 PLA 的主要改进就是可以多次编程,增加了芯片的使用灵活性。

早期的可编程逻辑器件主要由上述四种类型的芯片组成,即 PLA、PAL、GAL 和 FPLA,它们的一个共同特点是可以实现速度特性较好的逻辑功能,但由于其过于简单的结构也制约

它们只能实现规模较小的数字电路。随着科技的发展、技术的进步,人们对芯片的集成度要求越来越高。这些早期的 PLD 产品日渐不能满足人们的需求,于是新的可编程逻辑器件诞生了,它的名字叫 CPLD,英文全称 Complex Programmable Logic Device,翻译成中文即复杂可编程逻辑器件。光从名字我们就可以看出它比之前的 PLD 器件都要强大不少。

CPLD 可以看成是 PLA 器件结构的延续。在 CPLD 芯片的四周,分布着一系列称为宏单元的逻辑块,而芯片的中间部分则分布着一个连接矩阵,用于在各个逻辑块之间建立连接。每一个逻辑块的内部结构跟 PLA 非常类似,所以一个 CPLD 器件也可以被看成集成了若干个 PLA 和一个可编程连接矩阵的芯片。

CPLD 的出现将数字电路的设计带到了一个新的高度,但是,随着时间的推移,这种结构还是不能满足日益增大的电路规模需求,于是,本书所关注的重点——FPGA——诞生了。

FPGA 和 CPLD 并称为高密度可编程逻辑器件,但是,它们有着本质的不同。FPGA 芯片的内部架构相对于以往的可编程逻辑器件有着本质的变革,它并没有沿用类似 PLA 的结构,而是采用了逻辑单元阵列 LCA(Logic Cell Array)这样一个概念,一改以往 PLD 器件大量使用与门、非门的思路,而是大量使用查找表和寄存器等元素。目前,CPLD 从集成度、规模、功耗、编程灵活性等方面都远远落后于 FPGA,不过它还是有着它自身的一些特定优势,在今后的一段时间内将还会与 FPGA 并存,原因如下:

- ①CPLD 更适合于触发器有限而乘积项丰富的结构;
- ②由于布线结构不同,CPLD 时序延迟是均匀的和可预测的,而 FPGA 的时序延迟则具有不可预测性,所以 CPLD 的速度会比 FPGA 更快;
- ③CPLD 使用起来较方便,因为配置内容掉电不丢失,一般不需要外加配置芯片,所以 CPLD 保密性好,而 FPGA 的配置数据流容易被黑客截获,所以保密性差。

以上就是 FPGA 芯片的发展简史,为了深入了解这些 PLD 芯片的原理,请参阅【知己知彼→可编程逻辑器件的原理架构】。不过对于没有数字电路基础的读者,建议先看完本篇的后续章节,掌握一些数字电路的基础知识后,再做研究。

2.1.3 应用方向

FPGA 的应用方向非常广泛,按照应用领域来看,FPGA 在通信、数据处理、网络、仪器、工业控制、医学、生物、军事和航空航天等众多领域都已经得到了广泛的应用。FPGA 已经从最早的只应用于辅助功能以及胶合逻辑(连接各种功能块以及集成电路的逻辑电路)的简单器件,发展到现今众多产品的核心器件。并且随着功耗和成本的进一步降低,FPGA 还将进入更多的应用领域。当然了,以上所有这些领域,并不是只有 FPGA 可以做,那么为什么 FPGA 会在这么多领域有这么大的用武之地呢?这要从基于 FPGA 的产品特点来看。下是按照产品特点来分的 FPGA 几大应用方向。

1. ASIC 原型验证

ASIC,英文全称 Application Specific Integrated Circuit,翻译成中文即专用集成电路。它是应特定用户要求和特定电子系统的需要而设计、制造的集成电路。ASIC 的特点是面向特定用户的需求,它在批量生产时与通用集成电路相比具有体积更小、功耗更低、可靠性提高、性能提高、保密性增强、成本降低等优点。现代的 ASIC 产品常包含完整的处理器,类似 ROM、RAM、EEPROM 或 Flash 的存储单元以及其他模块,因此这样的 ASIC 产品也常被称为 SoC,

即 System on Chip, 翻译成中文即片上系统。

由此可见,随着 IC 设计的密度越来越高、产品的需求越来越苛刻,ASIC 芯片的设计也已经变得越来越复杂,其中软件设计或软件协助设计的成分也越来越多。而把软件设计转换为 ASIC 芯片的过程叫做“流片”,在集成电路设计领域,“流片”指的是“试生产”,就是说设计完电路以后,先生产几片几十片,供测试用。如果测试通过,就照着这个样子开始大规模生产了;如果测试失败,可不是简单地回头修改软件,等待再次流片这么简单。要知道,“流片”的成本是十分高昂的,如果连续两次以上“流片”都失败,那么公司很可能就会考虑取消该 ASIC 芯片的制造计划,因为继续下去投入的人力、物力和资金过多,即使最后“流片”成功也挽回不了损失。因此在“流片”之前,对软件设计的正确性进行充分的验证是十分且非常必要的。

而研究表明,60%以上“流片”失败的原因是由于软件设计中存在逻辑或功能性的错误,剩下的才是时序或者功率、功耗问题。由此可见,功能验证已经成为 ASIC 开发周期中最关键的环节。可是,随着 ASIC 密度的提高和设计复杂性的增加,对“流片”一次性成功的需求越来越迫切,而此时设计中出现错误的可能性也会越来越高,所以设计师无疑需要一种能够在短时间内发现复杂芯片设计中错误的高效验证方法。

以典型的移动电话芯片组设计来说:RTL 级软件模拟的方法光导入设计就需要约 30 天时间;采用较高层模型的软硬件协同仿真方法可以将导入设计缩短为 10 天;C 代码模型可以提供更短的运行时间,但差不多也需要 24 小时。由于 ASIC 设计师需要一种接近于 ASIC 运行速度的验证方法,所以上述几种方法都无法投入实际使用。并且这种验证方法不仅能全速运行相关功能仿真,并且还能方便地集成外部系统组件或接口。于是人们想到了 FPGA,事实上 FPGA 针对上述电话芯片组设计来说,只需要 30 秒钟的时间就可以完成原型验证。

从 30 天到 30 秒,FPGA 的性能优势是毋庸置疑的,因此,目前 90%以上的 ASIC 在“流片”之前都部分甚至全部地利用 FPGA 进行原型验证。因此,为 ASIC 提供原型验证平台是 FPGA 应用的一个重要领域。

2. SoPC

SoPC,英文全称 System on a Programmable Chip,翻译成中文即可编程片上系统。SoPC 是一种特殊的 SoC,因为它也是由单个芯片完成整个系统的主要逻辑功能。所不同的是 SoPC 所基于的单个芯片主要是可编程的 FPGA 芯片,所以它具有灵活的设计方式:可裁减、可扩充、可升级,并具备软硬件在系统可编程的功能。

目前 Xilinx 公司的 FPGA 芯片支持嵌入 Power PC 系列 CPU,Altera 公司也推出了自己的 CPU 软核 nios 系列,并且随着 FPGA 资源的爆炸式增长,只要肯尝试,我们可以将越来越多的处理器用 FPGA 内部的资源来实现。在 FPGA 芯片上嵌入软核,再配合上适当的 FPGA IP 核以及必要的外围存储芯片和接口芯片,这样的片上系统兼顾软件、硬件的特点,能够让开发者更加方便、灵活地实现系统需求的各种功能。因此以 FPGA 芯片为核心来搭建一套系统已经成为当前电路设计的一个重要方向。

3. 小规模产品

在介绍 ASIC 的 FPGA 原型验证的重要性时,我们介绍了“流片”的概念,并且了解到“流片”的成本是十分高昂的,因此,如果某个功能模块不是能够大量销售的,或者不存在潜在大规模用户,那么通常就不会采用 ASIC 的形式来实现它,因为小规模的生产无法拉低 ASIC 芯片

生产时的边际成本。例如,PC 机的 CPU 和手机中的相关通信芯片都是 ASIC 的,因为它们的销售市场相当大,可以把 ASIC 芯片的边际成本拉到很低。若没有大批量的后续产能作为支撑,FPGA 芯片将成为这类功能模块的一个很好的承载平台。

除此以外,有些公司由于规模较小,财力有限,也不会采用 ASIC“流片”的冒险形式来实现自己的产品。虽然目前市面上的微处理器和 DSP 芯片发展都比较成熟,但 FPGA 芯片和它们相比则有着得天独厚的并行处理优势,再加上可以基于 FPGA 构建 SOPC,因此这类公司也会对 FPGA 青睐有加。

还有的公司或机构虽然财力充沛,但是由于其行业的特点,导致产品的数量相对较少——例如航天领域,同步卫星轨道资源有限,不可能任由你发射太空垃圾——因此这类行业使用 FPGA 也比较多。

最后还有一些公司,虽然它们的产品规模可能不小,但由于它们出售的产品本身就价格高昂,早就赚得盆满钵盈,因此也就不在乎并且也懒得用 ASIC 替代 FPGA。

4. 要求功能灵活可配置的产品

如果你生产的产品的应用领域不确定,也就是说它不是一个专用的设备,例如测试测量行业的某些测试仪器或测试板卡,你不知道客户会拿它去测试什么信号。而这个行业不可能针对每个客户的测试需求来生产一个专用的测试设备,但如果你的测试设备不能满足客户的要求又卖不出去,这时 FPGA 就派上了用场。这些厂商,一般会使用 FPGA 来搭建一个通用的测试平台,然后再根据客户的要求去适当地对系统进行少量的外围硬件重配置和内部 FPGA 的功能修改,快速地满足客户的需求。

5. 更新换代快的产品

数字产品的更新换代速度是异常快的,2 年前一块 2000 元的伪高端显卡,即使在通货膨胀如此恶劣的环境下,游戏性能也还比不上今天一块 950 元的中低端显卡。当然,显卡中 GPU 芯片的技术非常成熟,并且目前已经基本被 NVIDIA 和 ATI 两家公司垄断,所以显卡没有使用 FPGA,这里只是通过显卡举个例子。但是类似这样的行业,有时候为了抢得先机,抢占客户,就必须使用 FPGA。

6. 科研领域

在科研领域中使用 FPGA 的很多,例如众多高校和科研院所,此时科研工作者们可以利用 FPGA 开发速度快,有很多现成的 IP 核可供使用的特点来快速搭建实验系统。并且可以利用 FPGA 并行的特点来验证某些理论、算法的可行性或者效率。尤其是科研领域最需要的就是创新性、与众不同、灵感闪现,他们要研究的主要都是现在没有的东西或者现在不成熟的东西,所以定制的电路无法为这些充满活力的大脑提供“胡作非为”的平台,此时 FPGA 就能成为他们的好帮手!

2.2 模拟与数字

FPGA 芯片是一款关于数字电路的芯片,那么什么又是“数字”呢?要讲清楚这点,我们必须结合它的孪生大哥——“模拟”,来一起介绍。模拟与数字,这一对孪生兄弟涉及到了太多的知识理论,不是一两本书,一两年时间能够完全搞懂的,因此在这里,做为 FPGA 的入门基础

知识之一,本书将尽可能以简单、易懂的方式来对模拟与数字的相关基础知识进行一个粗略而概括的介绍。

2.2.1 模拟的与数字的

如果非要给模拟的和数字的下个定义,那么大致应该是这样的——在现实生活中,可以用整数完美描述的就是数字的,反之就是模拟的。例如:休息了10分钟、喝了1升水、长胖了1斤、走了10公里路等等,这些描述都是不精确的,因为也许真实情况是你休息了9分59秒又999微秒、喝了1.000 000 2升水、长胖了1.000 000 014斤、走了9.999 998 9公里等等,但是这些所谓的真实情况也是几乎无法获得的,因为受现有的测量工具的测量精度、测量原理的限制,人类永远只能得到一个非常接近真实值但却不是真实值的近似结果,所以我们称时间、质量、体积、面积、长度等等量度为模拟的。但是,买了10个苹果,它确实就是10个苹果,无论个大个小的苹果都算是一个苹果;考试及格,它确实就是及格,没有介于及格与不及格之间的模棱两可的情况。因此,像这样的描述就是绝对精确的,所以我们称这种描述数量、是非判断等的量度就为数字的。

之所以要从现实生活上对模拟的和数字的下定义,是因为如果真要探究到微观或宏观的领域,模拟的和数字的就可能会发生互换。首先,任何物质都是由分子或者原子组成的,甚至时间究其本源也是离散的,如果有兴趣在这方面探究,可以去学习《量子物理》等。例如,如果我们有方法可以数清楚一杯水中水分子个数,那么其质量其实也是可以精确确定的。因此,如果真要探究到微观领域,那么一切都将是数字的。其次,如果从银河系的角度去观察地球上的苹果数目,恐怕也难以得到精确的测量,因此如果真要探究到宏观领域,那么一切又都是模拟的。

因此,我们现在说模拟的或是数字的,是基于我们人类所熟悉的现实生活中的体验、感知。但是即便如此,模拟的和数字的仍然是非常容易混淆的,因为有时候,模拟的和数字的之间很可能会发生互换。比如说,看问题的角度不同,例如,如果关心的是一筐苹果的重量,那它就是模拟的;可如果关心的是一筐苹果的数量,那它就是数字的。

现实生活中,模拟的量度要远远多于数字的量度,但是人们并没有为无法精确描述这些模拟的量度而苦恼,因为人类懂得近似。没错,近似是一种很好的用数字量来描述模拟量的方法,它可以让我们丢掉小数部分的细节,从而用整数近似完美地来表述。例如,当我们允许0.5厘米以内的误差时,可以采用四舍五入的方法来得到数字化的量度,比方说,小明身高172厘米,小强身高172厘米,但他们其实并不是一模一样高的。除此以外,抽象也是一种从模拟到数字的转换方式,例如,地球上的生物是千差万别的,没有哪两个生物是一模一样的,但是人们通过对这些生物形态特征、生活习性等的抽象提取,便将地球上的生物统分为动物、植物两大类,每一类下面又有若干门、纲、目、科、属、种等等,从而可以让我们精确地说出任何一个生物的种类名称。

好了,关于模拟的与数字的简介就讨论到这里,接下来,让我们缩小视野,将精力集中于和FPGA相关的部分去探讨更为具体的模拟与数字。

2.2.2 模拟信号与数字信号

信号是运载消息的工具,是消息的载体。例如,除夕之夜,钟声响起,这属于声信号;早上太阳升起,大地复苏,这属于光信号;夏天热、冬天冷,这属于热信号;等等。总之,无论要表达

什么、要传递什么,都必须利用信号,而根据信号的特征,则又可以分为模拟信号与数字信号,下面就对两者进行一个简单的介绍。

2.2.2.1 模拟信号的概念

如果以时间为横轴、温度为纵轴,所得到的某地区一天内的温度变化曲线类似如图 2-1 所示。

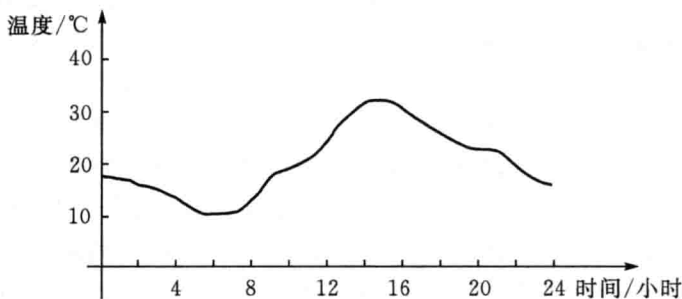


图 2-1

从图 2-1 可以看出,一天内的最低气温通常出现在凌晨,而最高气温则通常出现在午后两点。如果你需要将该地区一天内的温度变化情况告诉别人,那么上图中的曲线便成为了信号,并且还是一个典型的模拟信号,因为曲线上任一点对应的时间和温度都是无法精确描述的。

由此我们可以得出模拟信号的基本特征——时间连续、幅度连续。

所谓连续,其实是数学上的一个定义,即,设函数 $y = f(x)$ 在点 x_0 的某个邻域内有定义。如果当自变量 Δx 趋向于 0 时,若相应的函数改变量 Δy 也趋向于 0,则称函数 $y = f(x)$ 在点 x_0 处连续。如果 $y = f(x)$ 对于 x 所能取值的任意一个点 x 处连续,那么则称 $y = f(x)$ 是连续函数。

由此可见,模拟信号如果用函数形式来表达,即是一个连续函数。

2.2.2.2 数字信号的概念

如果以时间为横轴、人数为纵轴,所得到的某学校六年内的一年级人数变化曲线类似如图 2-2 所示。

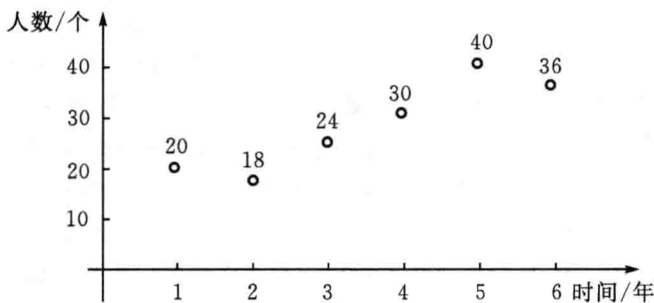


图 2-2

从图 2-2 可以看出,由于刚开始学校名气不大,因此招收学生较少,而后,随着学校的努力,人数慢慢地呈上涨趋势。如果你需要将该校六年内的一年级学生人数变化情况告诉别人,那么上图中的点序列便成为了信号,并且还是一个典型的数字信号,因为序列中任一点对应的时间和人数都是可以精确描述的。

由此我们可以得出数字信号的基本特征——时间离散、幅度离散。
所谓离散,其实也是数学上的一个定义,简单地说,离散即是不连续。

2.2.2.3 模拟信号转换为数字信号

前面讲过,近似是一种很好的用数字量来描述模拟量的方法,那么针对【模拟信号的概念】小节中的例子,先对该模拟信号的时间以小时为单位进行近似,可得如图 2-3 所示的图形。

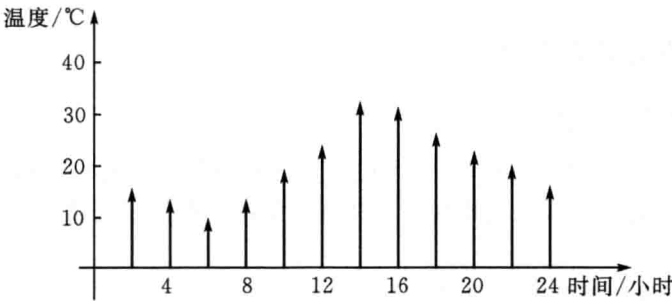


图 2-3

接着再对该上述信号的温度以 1℃为单位进行近似,可得如图 2-4 所示的图形。

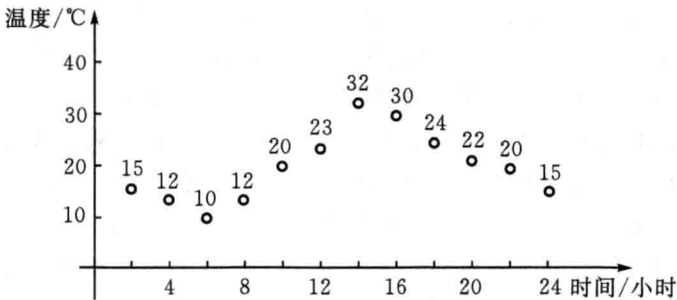


图 2-4

对比【数字信号的概念】小节的例子,可知我们已经成功地将模拟信号转换为数字信号,而所付出的代价就是牺牲了很多细节。

当然了,上述转化过程亦可以颠倒进行,即先对信号的温度以 1℃为单位进行近似,可得如图 2-5 所示的图形。

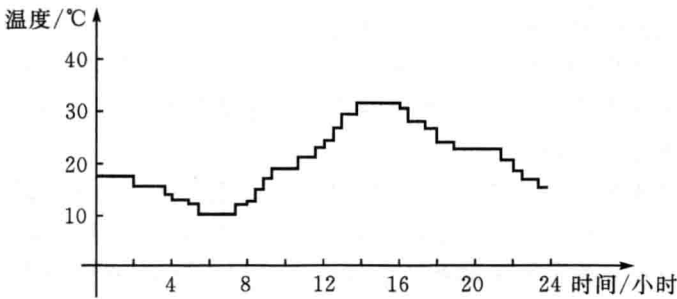


图 2-5

然后再对上述信号进行时间近似,最终得到的结果相同,这里就不再赘述。

采用更加专业一点的描述,上述近似过程中,时间上的近似应该称之为采样,而幅度上的近似应该称之为量化。除了采样以外,积累也是一种时间近似的方法。例如,统计某十字路口以小时为单位的车流量,就必须积累每个小时内通过该十字路口的车辆总数。绝大部分时候,仅仅量化是远远不够的,量化后的编码、处理往往也是重头戏,那么关于这些概念更为具体的描述请参见《数字信号处理》等相关的书籍。

2.2.3 模拟电路与数字电路

2.2.3.1 电信号的本质

电路是电信号的载体,而电信号又分为电流(I)和电压(U)两种互相依存的形式,即若电路中两点之间的电阻为 R ,那么其电流与电压的关系可用公式表述为:

$$U = IR$$

上述公式中,三者的基本概念解释如下:

电压(U):又叫电势差或电位差,这就好比如果U形管两端的水位线不一致而导致U形管中存在着水压进而导致水流一样,如果电路两端存在着电压,那么电路中便会形成电流。如果要解释得更为详细一点,其实是电压的存在会导致空间中形成电场,而电场会驱使空间中的带电载荷发生移动,从而便形成了电流。因此,更为专业的电压概念应该是——在电场中单位正电荷在电场力下从a点移到b点,那么电场力所做的功便等于a、b两点间的电压值或电压强度。电压在国际单位制中的单位是伏特,简称伏,用符号V表示。

电流(I):电流是电荷定向移动的宏观表现。电流的大小称为电流强度(简称电流),是指单位时间内通过导线某一截面的电荷量,单位是安培,简称安,用符号A表示。

电阻(R):如果大家有过堵车经历,那么电阻的概念理解起来就方便了。在车辆数不变的情况下,如果路上比较拥堵,那么单位时间通过某一路段的车辆数就会较少;如果道路比较通畅,那么单位时间通过某一路段的车辆数就会较多。同样的,如果电路载体本身对带电载荷的阻碍性较小,那么电流就会较大,反之电流就会比较小。因此,通常称电阻较小的载体为导体,电阻为0的为超导体,电阻无穷大或接近无穷大的为绝缘体,而那些电阻介于导体和绝缘体之间的称为半导体。电阻的单位是欧姆,简称欧,用符号 Ω 表示。

大部分情况下,电信号是以电压为传输形式的,即我们只从电路的目的端测量电压值的变化曲线来得到电信号中蕴含的信息。不过也有些情况,电信号是以电流为传输形式的,例如某些高速DA芯片的输出,不过我们并不直接测量电流值的变化曲线,而是利用本小节一开始介绍的电流、电压关系公式,通过接入电阻,将电流信号转变为更容易测量的电压信号进行观测、记录。如果要探究其原因,主要是由于电流的形成必须具有闭合回路,而电压的形成则不需如此,这就好比山上、山下各有一个湖泊,两个湖泊之间必然存在水压,但要想形成水流,则必须具有连接两个湖泊的水渠才行。

由于单位电荷的带电量为 $1.602\,177\,33 \times 10^{-19}\text{ C}$,是一个非常小的量,因此在现实世界中,无论是电流信号还是电压信号,都是模拟的。因此电信号的本质是模拟的,电路的本质也是模拟的。但是为什么还有模拟电路和数字电路之分呢?没错,还是近似和抽象的作用。如果将不关心的细节抹去,例如,如果认为电压值大于 1.7 V 表示逻辑1,反之表示逻辑0,那么我们就将原本模拟的电信号抽象为数字的电信号。尽管此时电路中实实在在传递的仍是那个

模拟的电信号,但由于我们主观上的近似与抽象,该模拟电路已然摇身一变成为了数字电路。

2.2.3.2 模拟电路简介

处理模拟信号电子电路即为模拟电路,而组成模拟电路的元素则包括电阻、电容、电感、电源、开关、二极管、三极管以及运算放大器等等。例如,图 2-6 即为一个最基本的模拟 RC 滤波电路。

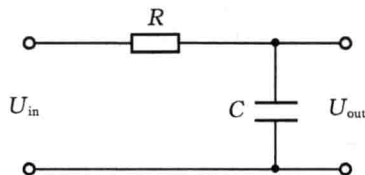


图 2-6

可以得出其幅频特性为:

$$A(f) = \frac{1}{\sqrt{1 + (2\pi fRC)^2}}$$

因此可知该电路的功能为低通滤波器,即 U_{out} 中将会舍去 U_{in} 中的高频部分而保留 U_{in} 中的低频部分。

继续观察上述 RC 电路,可以发现整个电路中没有电源模块出现,输出 U_{out} 完全取决于 U_{in} 和电路的结构特性,因此我们也称这种电路为无源电路。相应地,如果电路中具有电源模块,那么便称之为有源电路,如图 2-7 所示。

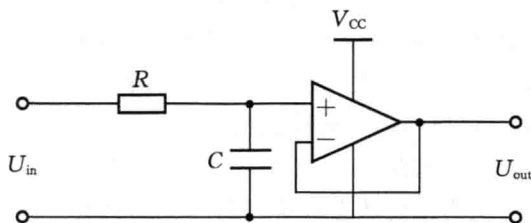


图 2-7

图 2-7 仍然是一个 RC 低通滤波器,所不同的是它利用了运算放大器的跟随特性,使 U_{out} 与 U_{in} 隔离的同时仍等于电容 C 两端的电压,因此,无论 U_{out} 后续又接入了什么样的电路网络,都不会对左半部分的无源电路的状况产生任何影响。而运算放大器之所以具有跟随特性,除了采用特定的电路连接方法外,还必须为其接上特定的电源偏置电路,如图中 V_{cc} 所示。只有放大器得到了合适的电源激励,它才会具有电压跟随以及其他种种特性,因此,凡是电路中包含这种必须供电后才能正常工作的结构或模块,都统称为有源电路。

除了滤波电路之外,典型的模拟电路种类还包括整流电路、放大电路、信号产生电路、直流稳压电路等等;按信号极性又可分为直流电路和交流电路;按信号频率又可分为低频电路、高频电路、射频电路;按密度特性又可分为分离元件电路和集成电路等等。总之,模拟电路千变万化、应用广泛、深不可测,需要长期的学习和经验积累才能够融会贯通,因此在这里,由于作者水平有限,只能点到即止。不过模拟电路并不是本书的重点,我们旨在通过它引出数字电路

的概念,有兴趣的读者可以另行参考模拟电路相关书籍。

2.2.3.3 数字电路简介

处理数字信号电子电路即为数字电路。事实上,所有的数字电路其本质都是模拟的,因为无论是电压信号还是电流信号,其本质都是模拟的电信号,之所以存在数字电路,只不过是人们看问题的角度不同罢了。例如图 2-8 所示电路。

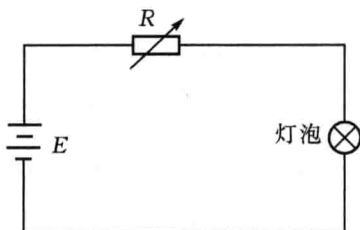


图 2-8

整个电路由一个电压源、一个可调电阻器和一个灯泡组成。如果随机地去改变可调电阻器的电阻值,那么灯泡两端的电压值或者流经灯泡的电流值也会随着改变,并且这两者都是模拟的电信号,但是无论怎样改变可调电阻器的电阻值,灯泡的状况只可能有两种——亮或者灭。如果只关心灯泡的亮、灭而不关心其程度,那么我们从灯泡所获取的信息就是数字的。事实上,点亮灯泡需要一定的条件,当然不同工作原理的灯泡会有所不同,例如灯泡两端的电压值要大于 1 V,或者流过灯泡的电流值要大于 1 A 灯泡才会点亮,因此,灯泡的亮、灭状态也可以转化为对其两端电压或者通过它的电流进行测量得出。这样一来,我们就可以不使用灯泡所传递的光信息来作为携带消息的信号,而转而直接测量其电信号参数来获取信息。例如,若该灯泡的点亮条件为其两端的电压值要大于 1.7 V,那么我们可以认为电压值大于 1.7 V 表示状态亮,反之表示状态灭,这样一来,上述模拟电信号就被近似和抽象为数字信号,整个电路也就瞬间化身为彻头彻尾的数字电路。

采用 1.7 V 这样单一的阈值作为状态转换判断的依据是有一些弊端的,那就是电压值正好在 1.7 V 附近抖动,那么数字化的状态也会不断地发生切换。而事实上,这样的情况多属于噪声干扰,是我们不希望发生的。因此,为了抽象出更加稳定的数字电信号,我们可以规定,大于 2 V 表示状态亮,小于 1 V 表示状态灭,而介于 1~2 V 之间的电压值不对应任何状态。这样做了之后,数字电信号抖动的情况将会得到极大的好转,但是由于 1~2 V 之间的模拟电信号没有数字状态对应,那么便产生了所谓的不定态。不定态是没法消除的,因为模拟电信号在时间和幅度上都是连续的,不可能从 1 V 以下直接跳到 2 V 以上,也不可能从 2 V 以上直接不经过渡地降至 1 V 以下,因此,为了减弱不定态对数字电路的影响,我们只能想办法让模拟电信号尽可能长时间地处于 1 V 以下或者 2 V 以上,并且如果要出现状态转换,希望 1 V 至 2 V (或者反之)这样的过渡时间要尽量地短。因此,为了满足双阈值近似、抽象的要求,在实际的电路中,我们往往需要将三极管、MOS 管等器件通过电源偏置为较符合我们所需数字双阈值电信号特性的状态,所以数字电路通常来说都是有源电路。

除了采用双阈值的方法外,我们也可以采用四阈值或者八阈值来将模拟电信号近似、抽象为三态或者四态的数字信号。不过状态越多,不定态也就越多,而且从最小状态跳至最大状态

所需要的过渡段就会更长并且这期间还不可避免地要经历其他有效状态。再加上电路实现时的复杂度和可行性限制,因此在实际的数字电路中,传递的数字电信号都只有两个状态,即状态 0 和状态 1。虽然数字电路中的电信号仅有两态,但是鉴于所基于电子器件的特性不同,其近似、抽象的阈值标准有着很多种,详细的请参考【模拟信号处理电路系统与数字信号处理电路系统】。

最后,再介绍一些数字电信号的常用术语。针对目前广泛应用的两态数字电信号,如果电信号长期地持续为某一个状态,我们称之为高电平(对应状态 1)或低电平(对应状态 0);如果在某一个状态的中间出现了一小段另一个状态,我们称之为正脉冲(状态 0 中包含了一小段状态 1)或负脉冲(状态 1 中包含了一小段状态 0);如果在某一时刻状态发生了切换,我们称之为上升沿(状态 0 切换至状态 1)或下降沿(状态 1 切换至状态 0)。

2.2.4 模拟信号处理电路系统与数字信号处理电路系统

2.2.4.1 模拟信号处理电路系统

处理模拟信号的电路系统即为模拟信号处理电路系统,简称模拟系统,它的结构框图如图 2-9 所示。

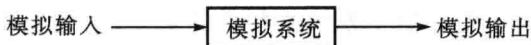


图 2-9

在科学和工程中,碰到的大部分信号均为自然模拟信号,因此可以直接采用合适的模拟系统来改变信号的特征,或者提取有用的信息。例如之前提到的 RC 模拟滤波器,它就可以起到让信号中低频分量通过而抑制高频分量的效果,从宏观上来说,给人的感觉就是信号被平滑了,信号的变化具有比较光滑的过渡。

2.2.4.2 数字信号处理电路系统

处理数字信号的电路系统即为数字信号处理电路系统,简称数字系统,它的结构框图如图 2-10 所示。

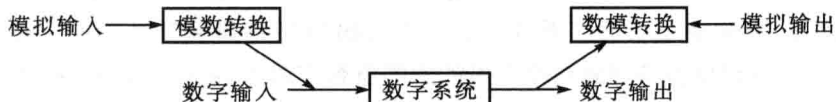


图 2-10

从图 2-10 可以看出,数字系统除了可以处理数字信号外,只要配合上适当的转换电路,就成为了模拟系统的替代版本。例如,先将模拟信号转换为数字信号序列 $\{a[n]\}$,其中 n 、 $a[n]$ 均为整数,如果想要对该数字信号做一个低通滤波,只需使用类似如下公式即可:

$$b[n] = (a[n] + a[n-1] + a[n-2] + a[n-3]) / 4 \quad (\text{除法结果仅保留整数})$$

由于平均具有减缓信号变化趋势的作用,因此经过上述公式的处理,数字信号 $\{a[n]\}$ 中的高频分量得到了抑制,低频信号得以保留,最后输出的序列 $\{b[n]\}$ 即为低通滤波后的 $\{a[n]\}$,接下来如果再经过反向的转换处理,将数字信号 $\{b[n]\}$ 再转换为模拟信号,就完成了

利用数字系统替代原模拟系统的工作。当然了,模拟的和数字的低通滤波器均有很多种不同的实现方法,并且参数不同滤波器的特性曲线也不同,本章节仅仅是使用一种比较简单的实现方式作为示例而已,其中的模拟低通滤波器和数字低通滤波器并没有直接的对应关系。

其实,经过本小节分析,我们也完全可以将模拟系统的框图扩展为如图 2-11 所示的形式。



图 2-11

由此可见,模拟系统除了可以处理模拟信号外,只要配合上适当的转换电路,也可以作为数字系统的替代版本。因此,在解决大部分实际问题时,几乎都是既可以采用模拟系统的方案,也可以采用数字系统的方案。这样一来,我们便面临着选择。所以,接下来我们将讨论一下这两种信号处理电路系统的优缺点。

2.2.4.3 模拟系统与数字系统的优缺点比较

通常来说,我们更倾向于使用数字系统来解决问题,因为相比于模拟系统,它有着以下优点:

第一,随着计算机的普及以及可编程芯片的发展,使得更改一个数字系统的功能变得十分简单,可能仅仅需要修改原有的程序代码即可。而模拟系统的更改往往意味着对硬件的重新设计、实现和调试,因此灵活性以及时效性上都远不及数字系统来得快捷、方便。

第二,数字系统具有更好的容错性。由于模拟电路受噪声等干扰的影响很大,因此系统设计者很难控制模拟电路系统的精度。例如,当我们想要从发送端传递一个电压值为 1.5 V 的模拟电信号,由于受到空间中无法排除的干扰因素影响,接收端最终接收到的电压值很可能为 1.72 V 或 1.43 V 等等,因此接收端无法推算出我们想要表达的真实意思,只能通过接收到的电压信号近似推测出我们想要表达的信息。相比之下,数字系统就要可靠得多,例如,如果我们想要从发送端传递一个数字信号——状态 1,那么实际上我们可能传递的仍是一个 1.5 V 的模拟电压信号,即便由于干扰的影响,使得接收端最终采集到的电压信号为 1.72 V 或 1.43 V 等等,但是根据模拟信号到数字信号的双阈值判定原则,接收端如果将 1 V 以上的模拟电压信号都认为是数字信号的状态 1,那么只要外界干扰对原始模拟电压信号的附加偏差小于 0.5 V,接收端就能完全准确无误地还原出发送端传递出的信息。

第三,数字信号可以很容易地存储于各种存储介质中,这样很方便传输和保存,并且可以留待在以后的任何位置、任意时间进行处理。

第四,对模拟信号进行精确的数学运算一般来说是很困难的,但是对于数字信号来说,无论再复杂的数学运算,都是可以比较容易的实现的。

第五,通常来说,数字系统实现起来的成本更低。

综上所述,随着高性能计算机、可编程芯片等的不断出现和更新换代,数字系统的应用变得越来越广。不过数字实现有它固有的局限性,例如,目前还无法将带宽极宽的模拟信号转换为合适的数字信号。所以,模拟电路在一些方面还是有其不可替代性的。

2.2.4.4 模拟系统与数字系统之间的接口

模拟系统与数字系统之间存在两种接口——模数转换接口以及数模转换接口,它们为数字信号处理的广泛应用起到了重要的桥梁作用。

1. 模数转换接口简介

模数转换接口,顾名思义,是将模拟电信号转换为数字电信号的接口。其转换的大体思路可以参照【本篇→模拟与数字→模拟信号与数字信号→模拟信号转换为数字信号】,一般来说,先时域采样,再幅度近似即可。为了保证转换后的数字信号不丢失其频域方面的某些特性,采样的间隔必须满足一定要求,通常来说,采样频率必须大于原始信号中有用频率的2倍。更为具体的内容请参考数字信号处理相关书籍中关于采样定理的内容,这里主要关注幅度近似的做法。

在【本篇→模拟与数字→模拟电路与数字电路→数字电路简介】中,我们介绍了数字电路中仅存在状态1和状态0,并且是采用双阈值的方法来进行判决的,可现实中,仅有两个状态肯定是无法对模拟信号的幅度进行较好近似的,下面我们就看看数模转换接口是怎么解决这个问题的吧。

假设待转换的模拟信号为电压信号,变化范围 $0\sim 3\text{ V}$ 。

如果我们仅仅关心该模拟电压信号是否大于 1.5 V ,那么数字电路中的两个状态就可以完全表述它了。转换方法很简单,假设此次采样得到的模拟电压值大于 1.5 V ,模数转换接口就输出状态1,反之输出状态0。注意,这里的判定仅采用了一个阈值,但是我们不用担心数字状态会出现非预期的多次翻转,因为在下一次采样值到来之前,本次采样的模拟电压值不会发生变化,即使由于某些特殊原因发生了变化,模数转换接口也仅仅会在两次采样间隔内输出一个数字信号量。通过前面的学习,我们知道,数字电信号的本质也是模拟的,因此模数转换接口实际输出的仍然是模拟电压值,那么为了便于后续数字电路对于该模拟电压值的判决,模数转换接口输出的模拟电压值必须符合双阈值判定规则。例如,如果后续数字电路能够接收的电压范围仍为 $0\sim 3\text{ V}$,那么当需要传递数字状态1时,可以使用一个大于 2.5 V 的模拟电压信号,而当需要传递数字状态0时,可以使用一个小于 0.5 V 的模拟电压信号。这样一来, $1.5\sim 3\text{ V}$ 的输入模拟电压映射到一个 $2.5\sim 3\text{ V}$ 的输出模拟电压, $0\sim 1.5\text{ V}$ 的输入模拟电压映射到了一个 $0\sim 0.5\text{ V}$ 的输出模拟电压,这便完成了模数转换。

现实中,输入的模拟电压信号往往需要更加精确的描述、更加精细的划分。例如,我们现在关心的也许是该模拟电压信号是否接近 0 V 、 1 V 、 2 V 或者 3 V ,那么此时,模数转换器再对输入模拟电压进行判定时,就至少需要3个阈值—— 0.5 V 、 1.5 V 、 2.5 V ,由此可见,要想表达这种要求的输入模拟电压信号,仅用一个数字电信号是不行的。由于一个数字电信号具有两种状态,那么将两个数字电信号A、B组合起来,就有了00、01、10、11四种状态,正好可以用来表达4种不同的输入模拟电压情况。假设后续数字电路能够接收的电压范围仍为 $0\sim 3\text{ V}$,那么,当输入模拟电压信号为 2.1 V 时,按照模拟输入的判定阈值,可知其应该归为表示 2 V 的数字信号,即A、B分别为状态1和状态0。若数字电路中的双阈值判定与上例相同,那么对应到模数转换器的输出端,数字输出端口A输出的应该为一个大于 2.5 V 的模拟电压,而数字输出端口B输出的应该为一个小于 0.5 V 的模拟电压,至此,数模转换器完成了一次模数转换。其他情况的分析类似,若要更加精确的近似输入模拟电压,仅需要增加模数转换器的数字

输出端口个数即可,具体转换过程这里就不再赘述。

综上所述,就是模数转换器的转换过程。关于模数转换器更为具体的介绍,请参考【本篇→硬件扩展】里的相关章节。

2. 数模转换接口简介

数模转换接口与模数转换接口恰恰相反,是将数字电信号转换为模拟电信号的接口,其转换过程也与模数转换恰恰相反,因此可以参考【模数转换接口简介】中的分析来理解数模转换接口的工作方式。

对于单个数字信号的数模转换,由于数字电信号的实质也就是模拟电信号,因此只要电信号取值范围要求一致,甚至可以不需要转换。而对于多个数字信号的模数转换,例如两个数字电信号 A、B 组成的四值数字信号,则可以按照其数字状态而转换为相应的模拟电平,例如,00、01、10、11 分别转换为模拟的 0 V、1 V、2 V、3 V。

最后,需要注意一点,如果需要将一个模拟电信号转换为 4 值数字电信号,仅考虑电信号幅度值是有损失的,例如一个 1.3 V 的模拟电信号,经过模数和数模转换之后,变成了 1 V 的模拟电信号,与转换前比相差了 0.3 V。事实上,介于 0.5~1.5 V 之间的模拟电信号经过模数和数模转换之后均变为了 1 V 的模拟电信号,变得没有区别,因此要想通过数字系统较好地替代模拟系统,模数转换时一定要具有较好的分辨率。如果再考虑到时间上的离散采样,那么数模接口也是无法恢复出漏掉的那些时间段的模拟电信号的。因此数字系统要想较好地代替模拟系统,必须参考数字信号处理等相关书籍中的理论,通过有效地进行采样、内插、滤波等手段来保证,这里就不再赘述。

2.2.4.5 数字系统之间的接口

前面介绍过,数字电路采用的都是双阈值判定准则,但是双阈值的具体取值并没有严格规定,因此不同的数字系统之间如果需要通信,就必须遵循相同的双阈值判定标准。除此以外,作为不同数字系统之间的接口,仅仅电压取值匹配还远远不够,还必须还具有一定的电流驱动能力,这便是数字系统之间的接口。

目前来说,业界中已经形成了很多种数字系统接口电平标准,在这里选择其中几种比较常见的介绍如下。

1. TTL

TTL 是 Transistor-Transistor Logic 的英文缩写,从其命名就可以看出,这种接口电平标准的初衷是用于基于三极管结构的数字系统之间的。

工作于 TTL 接口标准下的数字电路,其内部有源器件的标准电源供给应为 5 V,输出、输入情况如下:

对于输出端,状态 1 的电压要求为大于或等于 2.4 V,状态 0 的电压要求为小于或等于 0.5 V。

对于输入端,状态 1 的判定要求为大于或等于 2.0 V,状态 0 的判定要求为小于或等于 0.8 V。

对比输出、输入端的电压要求,可以看出输出端的电压输出要求要比输入端的双阈值判定标准更加严格,这样做主要是考虑到噪声的干扰以及电信号在输出与输入间的传递速度,从而让双阈值判定标准更加可靠。

2. LVTTL

由于 2.4 V 与 5 V 之间还有很大空间,这对改善噪声干扰并没有什么明显的好处,而且还会增加系统的功耗,并且由于数字状态 1、0 之间电平相差较大,还会影响到数字电路的响应速度。因此后来就把 TTL 的电压范围进行了一些压缩,从而形成了 LVTTL——Low Voltage Transistor—Transistor Logic,也即低压 TTL 电平标准。以下介绍两种目前常用的 LVC-MOS 标准。

(1)LVTTL3V3

LVTTL3V3 的意思,即其内部有源器件的标准电源供给为 3.3 V,输出、输入情况如下:

对于输出端,状态 1 的电压要求为大于或等于 2.4 V,状态 0 的电压要求为小于或等于 0.4 V。

对于输入端,状态 1 的判定要求为大于或等于 2.0 V,状态 0 的判定要求为小于或等于 0.8 V。

对比输出、输入端的电压要求可知,为了保证双阈值判定的稳定性和抗噪性,输出端的电压要求仍比输入端的双阈值判定标准要严格,这点对于所有的数字系统接口标准是一样的,以后不再赘述。

(2)LVTTL2V5

LVTTL2V5 的意思,即其内部有源器件的标准电源供给为 2.5 V,输出、输入情况如下:

对于输出端,状态 1 的电压要求为大于或等于 2.0 V,状态 0 的电压要求为小于或等于 0.2 V;

对于输入端,状态 1 的判定要求为大于或等于 1.7 V,状态 0 的判定要求为小于或等于 0.7 V。

3. CMOS

CMOS 是 Complementary Metal Oxide Semiconductor 的英文缩写,从其命名就可以看出,这种接口电平标准的初衷是用于基于 NMOS、PMOS 组成的 MOS 管结构的数字系统之间的。

工作于 CMOS 接口标准下的数字电路,其内部有源器件的标准电源供给为 5 V,输出、输入情况如下:

对于输出端,状态 1 的电压要求为大于或等于 4.45 V,状态 0 的电压要求为小于或等于 0.5 V;

对于输入端,状态 1 的判定要求为大于或等于 3.5 V,状态 0 的判定要求为小于或等于 1.5 V。

CMOS 与 TTL 接口相比,有了更大的噪声容限,并且其输入阻抗也远大于 TTL 输入阻抗。

4. LVCOMS

同 TTL 一样,鉴于功耗和响应速度的考虑,CMOS 也同样衍生出了 LVCMOS 接口标准,并且由于 MOS 管相对于三极管的导通门限更加低,因此 LVCMOS 比 LVTTL 更容易使用较低的电压进行通信。以下介绍几种目前常用的 LVCMOS 标准。

(1)LVCOMS3V3

LVC MOS3V3 的意思,即其内部有源器件的标准电源供给为 3.3 V,输出、输入情况如下:

对于输出端,状态 1 的电压要求为大于或等于 3.2 V,状态 0 的电压要求为小于或等于 0.4 V;

对于输入端,状态 1 的判定要求为大于或等于 2.0 V,状态 0 的判定要求为小于或等于 0.7 V。

(2)LVCOMS2V5

LVC MOS2V5 的意思,即其内部有源器件的标准电源供给为 2.5 V,输出、输入情况如下:

对于输出端,状态 1 的电压要求为大于或等于 2.0 V,状态 0 的电压要求为小于或等于 0.4 V;

对于输入端,状态 1 的判定要求为大于或等于 1.7 V,状态 0 的判定要求为小于或等于 0.7 V。

(3)LVCOMS1V8

LVC MOS1V8 的意思,即其内部有源器件的标准电源供给为 $V_{CC}=1.8\text{ V}$,当然这是有一定容忍度的,不过与之前介绍的电平标准不同,这个容忍度会影响它的输出、输入情况,介绍如下:

对于输出端,状态 1 的电压要求为大于或等于 $V_{CC}-0.45\text{ V}$ (若 V_{CC} 精确等于 1.8 V,则为 1.35 V),状态 0 的电压要求为小于或等于 0.45 V;

对于输入端,状态 1 的判定要求为大于或等于 0.65 倍的 V_{CC} (若 V_{CC} 精确等于 1.8 V,则为 1.17 V),状态 0 的判定要求为小于或等于 0.35 倍的 V_{CC} (若 V_{CC} 精确等于 1.8 V,则为 0.63 V)。

(4)LVCOMS1V5

LVC MOS1V5 的意思,即其内部有源器件的标准电源供给为 $V_{CC}=1.5\text{ V}$,它的容忍度也会影响到其输出、输入情况,介绍如下:

对于输出端,LVC MOS1V5 没有明确的要求,但是肯定是状态 1 越接近 V_{CC} 越好,状态 0 越接近 0 V 越好;

对于输入端,状态 1 的判定要求为大于或等于 0.65 倍的 V_{CC} (若 V_{CC} 精确等于 1.5 V,则为 0.975 V),状态 0 的判定要求为小于或等于 0.35 倍的 V_{CC} (若 V_{CC} 精确等于 1.5 V,则为 0.525 V)。

(5)LVCOMS1V2

LVC MOS1V2 的意思,即其内部有源器件的标准电源供给为 $V_{CC}=1.2\text{ V}$,它的容忍度也会影响到其输出、输入情况,介绍如下:

对于输出端,LVC MOS1V2 也没有明确的要求,但是肯定是状态 1 越接近 V_{CC} 越好,状态 0 越接近 0 V 越好;

对于输入端,状态 1 的判定要求为大于或等于 0.65 倍的 V_{CC} (若 V_{CC} 精确等于 1.2 V,则为 0.78 V),状态 0 的判定要求为小于或等于 0.35 倍的 V_{CC} (若 V_{CC} 精确等于 1.2 V,则为 0.42 V)。

5. LVDS

LVDS 是 Low Voltage Differential Signaling 的缩写,即低压差分信号,其输入、输出与之前所介绍的接口电平都不同,它需要通过两根线来完成通信。其工作原理如图 2-12 所示。

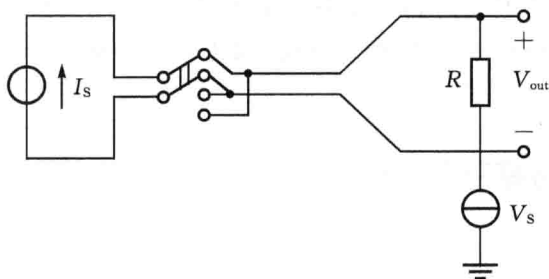


图 2-12

图 2-12 左部为 LVDS 输出端,其内部有一个恒流源 I_s ,大约恒定输出 $3.5\sim 4\text{ mA}$ 的电流值。最右边的 V_{out} 接入 LVDS 的输入端,而在靠近输入端的地方并联接入一个阻值为 100Ω 的匹配电阻 R 。通过改变双刀双掷开关的位置,而改变差分线上电流的方向,以此来表示数字状态 0 和 1,因此,接收端的差分线上将会由于电流方向的不同而表现出来 $\pm 350\text{ mV}$ 的差分电平,并依次作为数字状态的判定依据。上图右侧还有一个直流偏置电压源 V_s ,这主要是用来说明 V_{out} 的两端其实一般都是正电压的,实际电路中并没有该项。

由于 LVDS 的电压摆幅仅有 350 mV 左右,电流也仅有 3.5 mA 左右,而且又是差分传输,因此具有高速、超低功耗、低噪声和低成本等优良特性。

6. RS232

RS232 是美国电子工业协会 EIA(Electronic Industry Association)制定的一种串行物理接口标准。RS 是 Recommended Standard 的缩写,中文意思为推荐标准,232 为标识号。RS232 总线标准共设有 25 条信号线,这里我们仅讨论其数字电平接口判定标准。

RS232 的标准电源供给为 $\pm 12\text{ V}$ 或 $\pm 15\text{ V}$,状态 1 的电压要求为 $-15\sim 3\text{ V}$ 之间,状态 0 的电压要求为 $3\sim 15\text{ V}$ 之间。

7. RS485

RS485 相当于 RS232 的升级版,与 LVDS 类似,RS485 也是采用差分的形式来传递信息的(不过 RS485 是真的传了两路电压信号过去),因此抗干扰性要优于 RS232。这里,我们同样仅关心其数字电平接口判定标准。

RS485 的状态 1,其两线之间的电压差要求为 $2\sim 6\text{ V}$ 之间;状态 0,其两线之间的电压差要求为 $-6\sim -2\text{ V}$ 之间。

8. 不同标准之间能否混连?

上面介绍了多种数字系统之间的接口电平标准,通常在使用的时候,还是强烈建议大家为数字系统接口的双方选择一致的标准。不过有时候受限于两方的一些配置情况,可能并不能找出统一的电平标准来进行通信,那么此时,是不是除了设计接口转换电路板以外就没有别的方法了呢?并不是的,其实,有些不同的接口电平标准是可以兼容的。

首先单端和差分是不可能兼容的,因为从物理连线上它们就不一样。但是对于同种类的接口,如果 A 电平标准的输出符合 B 电平标准的输入,那么就称 A 的输出可驱动 B 的输入,如果反之亦然,那么称 A、B 两种电平标准可相互驱动。

例如,CMOS 的输出是可以驱动 TTL 输入的,但是反之则不行,因为 TTL 的状态 1 输出

仅为大于或等于 2.4 V,并不能达到 CMOS 判决状态 1 所需要的大于或等于 3.5 V;但是 LVTTTL3V3 和 LVCMOS3V3 却可以相互驱动,因为它们的输出都能满足彼此的输入判定要求。

2.3 数字逻辑电路基础知识

2.3.1 数与数制简介

数是表示事物的量的基本数学概念,而数制则是计数进位制的简称。

2.3.1.1 常见数制简介

数,必须结合数制,才能确定其所代表的具体意义,下面就为大家介绍一些常用的数制。

1. 十进制

十进制是日常生活中最为常用的进制,也是我们所最熟悉的进制。

从“数”的角度来说,十进制使用了 0、1、2、3、4、5、6、7、8、9 十个不同的符号来进行数的表示;从“数制”的角度来说,十进制采用了低位向高位“逢十进一”或高位向低位“借一当十”的计数方式。

举例来说,在十进制中,比 79 大 1 的数为 80,比 100 小 1 的数为 99。

2. 二进制

二进制是计算机最容易理解的进制,也是我们在进行数字逻辑相关设计时所最常用的进制。

从“数”的角度来说,二进制仅使用了 0、1 两个符号来进行数的表示;从“数制”的角度来说,二进制采用了低位向高位“逢二进一”或高位向低位“借一当二”的计数方式。

举例来说,在二进制中,比 1 大 1 的数为 10,比 100 小 1 的数为 11。为了区别于其他进制,二进制的数通常用小括号括起来,并配合下标 2 来表示,例如 $(100)_2$ 。

3. 八进制

八进制是早期计算机中比较常用的一种进制,因为那时的计算机还以 12 位或 36 位居多,不过随着 8 位、16 位、32 位甚至 64 位计算机的普及,八进制已经逐渐淡出了历史舞台。

从“数”的角度来说,八进制使用了 0、1、2、3、4、5、6、7 八个符号来进行数的表示;从“数制”的角度来说,八进制采用了低位向高位“逢八进一”或高位向低位“借一当八”的计数方式。

举例来说,在八进制中,比 47 大 1 的数为 50,比 100 小 1 的数为 77。为了区别于其他进制,八进制的数通常用小括号括起来,并配合下标 8 来表示,例如 $(50)_8$ 。

4. 十六进制

十六进制是现今计算机中比较常用的一种进制,也是我们在进行数字逻辑相关设计时所常用的一种进制。

从“数”的角度来说,十六进制相比于十进制,又增加了 a、b、c、d、e、f(大小写均可)六个符号来进行数的表示;从“数制”的角度来说,十六进制采用了低位向高位“逢十六进一”或高位向低位“借一当十六”的计数方式。

举例来说,在十六进制中,比 19 大 1 的数为 1a,比 100 小 1 的数为 FF。为了区别于其他

进制,十六进制的数通常用小括号括起来,并配合下标 16 来表示,例如 $(FF)_{16}$ 。

5. 六十进制

六十进制是日常时间记录中常用的一种进制(秒到分、分到时),除此以外,时间记录中还有二十四进制(时到日)、七进制(日到周)、三十进制(日到月)、十二进制(月到年)等等。这里就以六十进制为代表进行简单的介绍。

从“数”的角度来说,六十进制与十进制一样,使用 0~9 这十个符号来进行数的表示,只不过当某一位的数值大于 9 时,六十进制就采用两位十进制数来表示;从“数制”的角度来说,六十进制采用了低位向高位“逢六十进一”或高位向低位“借一当六十”的计数方式。

举例来说,在六十进制中,比 1 分 59 秒大 1 秒的时间为 2 分 0 秒,比 1 分小 1 秒的时间为 59 秒。

2.3.1.2 数制间的转换

各种各样的不同数制适用各种各样的不同应用,但是不同的应用之间却有着千丝万缕的联系。除此以外,人们往往习惯于使用十进制数,而对其他进制的数不太敏感,可又需要使用不同数制承担各种各样的工作。

鉴于以上两点,我们必须要学会在不同数制间进行转换,因此本节就来介绍一下具体的数制间转换操作。

1. 整数部分

任何数都可以分为整数部分和小数部分,对于进制间的转换,这两部分是稍有不同的,因此我们将分开来讨论。

(1) X 进制整数转换为十进制整数

十进制是人类日常生活中最常用的进制方法,因此我们先来看看如何将一个 X 进制的整数转换为十进制的整数。

我们先来看一看“权值”的概念。当一个数字处于不同的数位时,它所代表的数值是不同的。以十进制数为例,如果 7 处于十位,那么它就代表数值 70;如果 7 处于百位,那么它就代表数值 700;依次类推。因此如果一个十进制数为 756,那么它所代表的数值可以按位展开求得,方法如下:

$$756 = 7 \times 100 + 5 \times 10 + 6 = 756$$

亦可写为

$$756 = 7 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 = 756$$

由此可见,权值即为数位的数值单位,例如,十进制数中十位的权值为十,百位的权值为一百。因此,对于 X 进制整数来说,由于我们在书写的时候习惯将最高位写在左边、最低位写在右边,所以,其右数第 N 位的权值即为 X^{N-1} 。

再看上述等式,可发现其左右两边完全一致,因此十进制的数对于人脑来说,看到即可理解,所见即是所得,这便是人们非常习惯于使用十进制计数的原因。那么利用权值的概念并结合上述等式,便可得出 X 进制整数转换为十进制整数的方法——只需要将 X 进制整数按照权值展开求和即可,例如:

$$(77)_8 = 7 \times 8 + 7 \times 1 = 63$$

$$(110)_2 = 1 \times 4 + 1 \times 2 + 0 \times 1 = 6$$

$$(FF)_{16} = 15 \times 16 + 15 = 255$$

(2) 十进制整数转换为 X 进制整数

了解了如何将 X 进制整数转换为十进制整数,那么接下来我们再来看一看其逆变换是如何实现的。

仍以十进制数 756 为例,我们知道其表示的数值就是 756,那么将上一小节例子中的等式重写如下:

$$756 = 7 \times 10^2 + 5 \times 10^1 + 6 \times 10^0$$

让 756 除以其进制,即除以十,结果写成商和余数的形式,可得:

$$756/10 = 7 \times 10 + 5 \quad \dots\dots 6$$

对商重复上述操作,可得

$$(756/10)/10 = 7 \quad \dots\dots 5$$

$$((756/10)/10)/10 = 0 \quad \dots\dots 7$$

由此可见,当商变为 0 时,只要将得到的余数自下而上串联起来,便可恢复出原本的十进制整数。其实上述方法便是将十进制整数转换为 X 进制整数的方法,其基本原理就是利用了权值的特性——第一次除法中,由于除了目标 X 进制数中最低位的数之外,其余位的数所代表的数值均为 X 的整数次幂(≥ 1)的倍数,因此它们都可以被进制 X 整除,所以第一次除法所得的余数即为 X 进制的最低位。以此类推,每次对上次的商进行同样的处理,便可依次得出次低位、……、次高位、最高位。举例如下:

① 将十进制整数 63 转换为八进制整数

$$63/8 = 7 \quad \dots\dots 7$$

$$7/8 = 0 \quad \dots\dots 7$$

因此 $63 = (77)_8$ 。

② 将十进制整数 6 转换为二进制整数:

$$6/2 = 3 \quad \dots\dots 0$$

$$3/2 = 1 \quad \dots\dots 1$$

$$1/2 = 0 \quad \dots\dots 1$$

因此 $6 = (110)_2$ 。

③ 将十进制整数 255 转换为十六进制整数:

$$255/16 = 15 \quad \dots\dots 15(\text{对应 F})$$

$$15/16 = 0 \quad \dots\dots 15(\text{对应 F})$$

因此 $255 = (FF)_{16}$ 。

对比上一小节中的例子,可见它们的结果一一对应。

(3) X 进制整数转换为 Y 进制整数

本小节,我们来讨论一下更为普遍的进制转换方法,即如何将 X 进制的整数转换为 Y 进制的整数。

其实通过前面的分析,我们很容易得出一种通用的转换方案,即利用前两个小节的知识,先将 X 进制的整数转换为十进制整数,然后再将十进制整数转换为 Y 进制整数即可。该方法利用了十进制数作为转换的媒介,能够胜任任何两个进制整数之间的转换,例如七进制整数 $(51)_7$ 到八进制整数的转换方法如下:

第一步,将七进制数转换为十进制数。

转换后的十进制数 $=5 \times 7 + 1 = 36$

第二步,将十进制数转换为八进制数。

$$36/8=4 \quad \dots\dots 4$$

$$4/8=0 \quad \dots\dots 4$$

因此 $(51)_7 = 36 = (44)_8$,完成了七进制整数与八进制整数之间的转换。

在数字逻辑当中,较常用的进制几乎都是2的整数次幂进制,例如二进制、八进制、十六进制等,因此今后我们会经常碰到这些比较特殊进制之间的转换问题。当然,我们仍可以沿用本小节的方案来解决2的整数次幂进制之间的转换问题,不过在这方面其实存在着更为简便的方法,下一小节将详细展开讨论。

(4)二的整数次幂进制之间的转换

首先,先看一下如何将二的整数次幂进制整数转换为二进制整数。

对于 2^n 进制($n > 1$)来说,它的一位数所表示的内容恰好对应 n 位二进制数所能表示的内容。例如,八进制数的一位可以有8种不同的取值——0~7,那么3位二进制数也正好有8种不同的取值——000、001、010、011、100、101、110、111。因此,如果需要将一个 2^n 进制($n > 1$)的整数转换为二进制的整数,仅需按位将这个进制整数分别转换为 n 位二进制数,然后串联即可得到结果。例如,对于十六进制数 $(F6)_{16}$,由于

$$(F)_{16} = (1111)_2$$

$$(6)_{16} = (0110)_2$$

因此,转换为二进制后应该为

$$(F6)_{16} = (11110110)_2$$

其次,来看一下如何将二进制整数转换为二的整数次幂进制整数。其实思路恰恰与上述相反,即,若要将二进制整数转换为 2^n 进制($n > 1$)整数,则先从该二进制数的右边(最低位)开始,每数 n 位划分为一组,最后一组如果不足 n 位则高位补0,然后将每组二进制数转换为1位进制数值符号,串联后即可得到结果。例如,对于二进制数 $(11110110)_2$,分组后可得 $(1111)_2$ 和 $(0110)_2$,由于

$$(1111)_2 = (F)_{16}$$

$$(0110)_2 = (6)_{16}$$

因此,转换为十六进制后应该为

$$(11110110)_2 = (F6)_{16}$$

最后,来看一下如何在两个任意的二的整数次幂进制之间进行整数转换。其思路与【X进制整数转换为Y进制整数】小节介绍的类似,只不过此处的媒介变为了二进制数。即,若要将 2^n 进制($n > 1$)整数转换为 2^m 进制($m > 1$)整数,可以先将 2^n 进制整数转换为二进制数,然后再将该二进制数转换为 2^m 进制整数即可。例如,将八进制数 $(263)_8$ 转换为十六进制数,方法如下:

第一步,将八进制数转换为二进制数。由于

$$(2)_8 = (010)_2$$

$$(6)_8 = (110)_2$$

$$(3)_8 = (011)_2$$

所以转换后的二进制数 $= (10110011)_2$ (省略了高位的零)。

第二步, 将二进制数转换为十六进制数。分组可得 $(1011)_2$ 和 $(0011)_2$, 由于

$$(1011)_2 = (B)_{16}$$

$$(0011)_2 = (3)_{16}$$

因此, 转换为十六进制后应为 $(B3)_{16}$, 故

$$(236)_8 = (B3)_{16}$$

2. 小数部分

下面来看一下小数部分的进制转换。

(1) X 进制小数转换为十进制小数

首先仍然需要关注权值, 不过此处关心的是小数部分的权值罢了。仿照整数部分的概念, 可以得出, 对于 X 进制小数来说, 其小数点往右数第 N 位的权值即为 X^{-N} 。例如十进制小数中, 小数点右数第一位的权值为十分之一, 第二位为百分之一。那么仿照整数部分的转换, 我们可以得出 X 进制小数转换为十进制小数的方法——只需要将 X 进制小数按照权值展开求和即可, 例如:

$$(0.1)_8 = 1 \times 8^{-1} = 0.125$$

$$(0.11)_2 = 1 \times 2^{-1} + 1 \times 2^{-2} = 0.75$$

$$(0.8)_{16} = 8 \times 16^{-1} = 0.5$$

注意, 与整数部分的转换不同, X 进制的小数并不一定都可以完全精确的转换为有限位十进制的小数, 例如:

$$(0.1)_3 = 1 \times 3^{-1} = 0.333333333\ldots$$

(2) 十进制小数转换为 X 进制小数

十进制小数转换为 X 进制小数的思路与整数恰恰相反, 因为此处用到的是乘法, 即乘以目标的进制。其基本原理也是利用了权值的特性——第一次乘法中, 由于除了目标 X 进制数中小数点右数第一位的数之外, 其余位的数所代表的数值均为 X 的负整数次幂 (>1) 的倍数 ($<X$ 倍), 因此它们乘以 X 后仍为小数, 所以第一次乘法所得的整数部分即为 X 进制小数的右数第一位。以此类推, 每次对上次得到的小数部分做同样的处理, 便可依次得出右数第二位、第三位、…。举例如下:

① 将十进制小数 0.125 转换为八进制整数:

$$0.125 \times 8 = 1.00$$

因此 $0.125 = (0.1)_8$ 。

② 将十进制小数 0.75 转换为二进制小数:

$$0.75 \times 2 = 1.5$$

$$0.5 \times 2 = 1$$

因此 $0.75 = (0.11)_2$ 。

③ 将十进制小数 0.5 转换为十六进制小数:

$$0.5 \times 16 = 8$$

因此 $0.5 = (0.8)_{16}$ 。

注意, 与整数部分的转换不同, 十进制的小数并不一定都可以完全精确的转换为有限位 X 进制的小数, 例如若需要将 0.6 转换为六进制数, 由于

$$0.6 \times 6 = 3.6$$

$$0.6 \times 6 = 3.6$$

...

因此 $0.6 = (0.333333\cdots)_6$ 。

(3) X 进制小数转换为 Y 进制小数

本小节,我们来讨论一下更为普遍的进制转换方法,即如何将 X 进制的小数转换为 Y 进制的小数。

其实通过前面的分析,我们很容易得出一种通用的转换方案,即利用前两个小节的知识,先将 X 进制的小数转换为十进制小数,然后再将十进制小数转换为 Y 进制小数即可。该方法利用了十进制数作为转换的媒介,能够胜任任何两个进制小数之间的转换,例如六进制小数 $(0.3)_6$ 到八进制小数的转换方法如下:

第一步,将六进制数转换为十进制数。

$$\text{转换后的十进制小数} = 0.3 \times 6^{-1} = 0.5$$

第二步,将十进制数转换为八进制数。

$$0.5 \times 8 = 4$$

因此 $(0.3)_6 = 0.5 = (0.4)_8$, 完成了六进制小数与八进制小数之间的转换。

(4) 二的整数次幂进制之间的转换

与整数部分类似,二的整数次幂进制之间的小数转换也存在一些简便方法。

首先,先看一下如何将二的整数次幂进制小数转换为二进制小数。

参考整数部分的分析,这里可得类似的结果。即,如果需要将一个 2^n 进制 ($n > 1$) 的小数转换为二进制的小数,仅需按位将这个 2^n 进制小数分别转换为 n 位二进制数,然后串联即可得到结果。例如,对于十六进制小数 $(0.8F)_{16}$,由于

$$(8)_{16} = (1000)_2$$

$$(F)_{16} = (1111)_2$$

因此,转换为二进制后应该为

$$(0.8F)_{16} = (0.10001111)_2$$

其次,来看一下如何将二进制小数转换为二的整数次幂进制小数。其实思路恰恰与上述相反,即,若要将二进制小数转换为 2^n 进制 ($n > 1$) 小数,则先从该二进制小数的小数点右边第一位开始,每数 n 位划分为一组,最后一组如果不足 n 位则低位补 0,然后将每组二进制数转换为 1 位 2^n 进制数值符号,串联后即可得到结果。例如,对于二进制小数 $(0.10001111)_2$,分组后可得 $(1000)_2$ 和 $(1111)_2$,由于

$$(1000)_2 = (8)_{16}$$

$$(1111)_2 = (F)_{16}$$

因此,转换为十六进制后应该为

$$(0.10001111)_2 = (0.8F)_{16}$$

最后,来看一下如何在两个任意的二的整数次幂进制之间进行整数转换。其思路与【X 进制小数转换为 Y 进制小数】小节介绍的类似,只不过此处的媒介变为了二进制数。即,若要将 2^n 进制 ($n > 1$) 小数转换为 2^m 进制 ($m > 1$) 小数,可以先将 2^n 进制小数转换为二进制小数,然后再将该二进制小数转换为 2^m 进制小数即可。例如,将八进制数 $(0.47)_8$ 转换为十六进制

小数,方法如下:

第一步,将八进制数转换为二进制数。由于

$$(4)_8 = (100)_2$$

$$(7)_8 = (111)_2$$

所以转换后的二进制数 $= (0.100111)_2$ 。

第二步,将二进制数转换为十六进制数。分组可得 $(1001)_2$ 和 $(1100)_2$ (低位补 0), 由于

$$(1001)_2 = (9)_{16}$$

$$(1100)_2 = (C)_{16}$$

因此,转换为十六进制后应为 $(0.9C)_{16}$, 故

$$(0.47)_8 = (0.9C)_{16}$$

3. 完整的转换

一个数可以分解为整数部分与小数部分之和,因此完整的数制间转换也可以分解为整数部分的转换和小数部分的转换两个部分,然后将两者相加(串联)即可。例如,若要将 $(F.8)_{16}$ 换为六进制,可以按照如下方法进行:

第一步,整数部分转换。

$$(F)_{16} = 15 = (23)_6$$

第二步,小数部分转换。

$$(0.8)_{16} = 0.5 = (0.3)_6$$

因此,串联后可得

$$(F.8)_{16} = 15.5 = (23.3)_6$$

2.3.1.3 二进制编码简介

数字逻辑只认识二进制,因此要想利用数字逻辑电路做事情,我们就必须了解二进制的编码知识。

1. 自然二进制编码

以二进制数的形式存在的编码即为自然二进制编码。例如,用 101 代表数值 5,1000 代表数值 8,由于这种表示形式和二进制数一模一样,即 $5 = (101)_2$, $8 = (1000)_2$, 因此称之为自然二进制编码。注意,二进制编码并不一定表示二进制数,它的含义是由人们预先预定所赋予的,后面我们将会介绍一些其他二进制编码形式。

自然二进制编码的方式有其优点,那就是非常利于数字逻辑电路的理解,因此其在数字逻辑电路中有着广泛的应用,不过其也有着不可忽视的缺点,那就是不利于人脑理解。

2. 十进制数的二进制编码方法

数字逻辑电路所完成的功能最终还是要为人类服务的,因此它少不了与人脑进行交互,可是人脑习惯使用的是十进制码。如果采用自然二进制编码,那么每次向数字电路输入信息,都需要使用除法取余的形式做一次十进制数到二进制数的转换;而每一次从数字电路中读取信息,又要再做一次除法取余的二进制到十进制的转换(除非想自己演算,否则二进制到十进制的转换都是由数字逻辑来承担的,而对于数字逻辑来说二进制是它们习惯的进制,因此从二进制转换为十进制需要用到除法取余的方式)。由于除法取余这项操作的效率比较低,因此采用自然二进制码非常不利于高效的人机交互,因此本小节将为大家介绍一些用二进制数来表示

十进制数的编码方法,即 BCD 码。

BCD 码,英文全称 Binary Coded Decimal,中文简称二—十进制码。由于十进制数共有 0~9 十个数码,因此为了能够全面地表述它们,必须使用至少 4 位二进制数码来表示 1 位十进制数码。4 位二进制数共有 0000~1111,共 16 种编码形式,可以从中任选 10 个来分别代表十进制数中的 10 个数码,因此方法有非常多种。下表中列举了一些比较常用的 BCD 编码形式,供大家参考。

十进制	8421 码	2421 码	余三码	单位距离码	余三循环码
0	0000	0000	0011	0000	0010
1	0001	0001	0100	0001	0110
2	0010	0010	0101	0011	0111
3	0011	0011	0110	0010	0101
4	0100	0100	0111	0110	0100
5	0101	1011	1000	0111	1100
6	0110	1100	1001	0101	1101
7	0111	1101	1010	0100	1111
8	1000	1110	1011	1100	1110
9	1001	1111	1100	1110	1010

(1) 有权 BCD 码

有权 BCD 码是指表示十进制数码的 4 位二进制编码中,每一位二进制编码都有一定的权值,例如上表中的 8421 码和 2421 码。对于有权 BCD 码,可以根据权值按位展开求得其所代表的十进制数,例如:

$$\begin{aligned}[0111]_{8421\text{BCD}} &= 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 7 \\ [1101]_{2421\text{BCD}} &= 1 \times 2 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 7\end{aligned}$$

8421 码是最常用的有权 BCD 码,由于其位权值与自然二进制码一样,因此通常也叫自然权码。而 2421 码中,0 和 9、1 和 8、2 和 7、3 和 6、4 和 5 互为反码,这种特性被称为自补性,在数字逻辑中也是很有用的。

(2) 无权 BCD 码

无权 BCD 码中,每一位二进制编码都没有固定的权值,因此不能通过按位展开乘权相加的方法来得出它们所表示的十进制数。由于它们不受权值的限制,因此它们的形式更加灵活,并且都有着各自比较独特的优点。例如,

余三码是在 8421 码的基础上,对以每个码字加 3 得到的,采用余三码进行运算比较方便,比如说在进行加法时,无须额外的判断逻辑,余三码就能正确地产生进位,不过对于求和后的值需要进行一些修正。

而单位间距码和循环余三码,它们的两个相邻的码字之间仅有一位不一样,利用这一点可以避免计数过程中所产生的中间不定态。

(3) BCD 码与十进制数之间的转换

在 BCD 码中,4 位二进制数对应 1 位十进制数,因此若要将一个多位十进制数转换为

BCD 码,只需按位进行编码后串联即可,例如:

将十进制 73 用 8421BCD 码表示,由于

$$7=[0111]_{8421BCD}, 3=[0011]_{8421BCD}$$

因此

$$73=[01110011]_{8421BCD}$$

反之,如果我们要从 BCD 码中获得十进制数的数值信息,需要先对 BCD 码进行分组,每组 4 个 2 进制码字,然后按照相应算法或者参考相应表格得出每组对应的十进制码字,串联即得结果,例如:

将单位距离码 00110010 转换为十进制数值,先分组为 0011、0010,由于

$$[0011]_{\text{单位距离码}}=2, [0010]_{\text{单位距离码}}=3$$

因此

$$[00110010]_{\text{单位距离码}}=23$$

2.3.2 数字逻辑的基本运算

2.3.2.1 集合与数字逻辑

1. 集合的基本概念和运算

(1) 基本概念

集合是具有某种特定性质的事物的总体。通常,我们用如下方式来表示集合:

$$A=\{\text{元素 } 1, \text{元素 } 2, \dots\}$$

如果属于集合 A 的元素全部都属于集合 B ,那么我们称集合 A 是 B 的子集,记做:

$$A \subseteq B$$

若集合 B 中至少存在一个不属于集合 A 的元素,则称 A 是 B 的真子集,记做:

$$A \subset B$$

有时候,我们分析问题,往往限定在一个集合范围内,那么这个集合称为总集。例如,我们只在整数范围内讨论问题,那么整数集就是当前的总集。而我们在讨论问题时得到的所有直接或间接集合都是总集的子集,例如奇数集合、偶数集合、素数集合等等都是整数集的子集。

如果一个集合什么元素都不包括,那么这样的集合称为空集,它也是任何集合的子集。

(2) 基本运算之交集

如果一个元素属于集合 A ,又属于集合 B ,那么所有具有这种特性的元素组成一个新的集合 C ,我们称 C 为 A 与 B 的交集,记做:

$$C=A \cap B$$

求解交集的过程可以通过图 2-13 来理解。

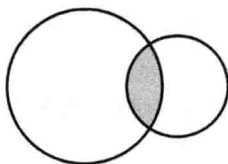


图 2-13

图 2-13 中,两个圆圈分别代表集合 A 和集合 B ,而中间阴影部分即为其交集。例如,有些人会用电脑、有些人会开车,那么它们的交集就是既会用电脑又会开车的人。

(3) 基本运算之并集

如果一个元素满足属于集合 A 或者属于集合 B 两者中至少一个,那么所有具有这种特性的元素组成一个新的集合 C ,我们称 C 为 A 与 B 的并集,记做:

$$C = A \cup B$$

求解并集的过程可以通过图 2-14 来理解。

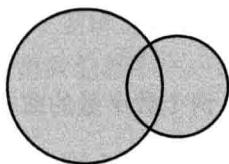


图 2-14

图 2-14 中,两个圆圈分别代表集合 A 和集合 B ,而所有阴影部分即为其并集。例如,有些人喜欢踢足球、有些人喜欢打篮球、有些人喜欢游泳……那么它们的并集就是所有热爱运动的人。

(4) 基本运算之补集

如果 S 是一个总集,其内部一个元素不属于集合 A ,那么那么所有具有这种特性的元素组成一个新的集合 C ,我们称 C 为 A 的补集,记做:

$$C = \bar{A}$$

求解补集的过程可以通过图 2-15 来理解。

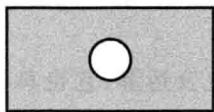


图 2-15

图 2-15 中,圆圈代表集合 A ,方框代表总集 S ,介于方框和圆圈之间的阴影部分即为其补集。例如,总集为整数,那么奇数集合的补集即为偶数集合。

2. 从集合到数字逻辑

如果要讨论数字逻辑,相当于总集的元素空间只包含一个元素,那么其空间下的集合要么包含这个元素,要么是空集,即“有”或者“无”的概念。如果用 1 表示包含这个元素的集合,0 表示不包含这个元素的集合(即空集),那么便可得出数字逻辑空间下的运算规则与集合的运算对应如下:

$$A \cdot B \Leftrightarrow A \cap B \quad (\cdot \text{号通常可省略})$$

$$A + B \Leftrightarrow A \cup B$$

$$\bar{A} \Leftrightarrow \bar{A}$$

2.3.2.2 数字逻辑的基本运算

1. 与、或、非运算

与、或、非是数字逻辑中最基本的三个运算(也称为逻辑乘、逻辑加、逻辑非),通过前一小节介绍,我们知道它们分别与集合的交集、并集、补集一一对应。通俗点来讲,“与”就是两者

皆“有”才为“有”，“或”就是两者皆“无”才算“无”，“非”就是从“无”到“有”或从“有”到“无”。

2. 简单复合运算

在数字逻辑中，除了以上三种基本运算外，还有一些简单的复合运算也非常地常用，因此人们对这些简单的复合运算进行了单独的命名，介绍如下。

(1) 与非

“与非”是“与”和“非”的复合运算，两个数字量的与非表示如下：

$$\overline{A \cdot B}$$

通俗点来讲，“与非”就是两者皆“有”才为“无”。

(2) 或非

“或非”是“或”和“非”的复合运算，两个数字量的或非表示如下：

$$\overline{A+B}$$

通俗点来讲，“或非”就是两者皆“无”才为“有”。

(3) 与或非

“与或非”是“与”、“或”和“非”的复合运算，四个数字量的与或非表示如下：

$$\overline{A \cdot B + C \cdot D}$$

(4) 异或

“异或”也是“与”、“或”和“非”的复合运算，它具有单独的运算符号，一般用“ \oplus ”表示。两个数字量的异或表示如下：

$$A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$$

通俗点来讲，“异或”就是两者不同才为“有”，两者相同即为“无”。

(5) 同或

“同或”也是“与”、“或”和“非”的复合运算，它也具有单独的运算符号，一般用“ \odot ”表示。两个数字量的同或表示如下：

$$A \odot B = A \cdot B + \overline{A} \cdot \overline{B}$$

通俗点来讲，“同或”就是两者相同才为“有”，两者不同即为“无”。

2.3.2.3 真值表

对于稍微复杂一些的情况，往往很难直接给出其逻辑表达式，因此，通常我们先穷举出所有的输入情况，然后再列出它们对应的输出情况，如果再将这些输入、输出情况制成表，那么该表就称为真值表。通常来说，逻辑的输入一般列在真值表的左边，而输出则列在右边，由于每一个输入都有 0、1 两种取值可能，所以一个 N 输入逻辑，其真值表必然包含 2^N 次幂个表项。

事实上，真值表并不是仅用于复杂逻辑的，任何逻辑函数都有其对应的真值表，例如与逻辑的真值表如下：

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

上表中,A、B为输入,F为输出。

由此可见,真值表与逻辑函数表达式,在表示逻辑功能方面是完全等价的,因此它们也是可以相互转化的。不过表示同一逻辑可以有多种不同的逻辑表达式形式,但却只能存在一种真值表,这便是真值表的唯一性,因此,真值表可以非常方便地用在逻辑函数相等判断上。除此以外,由于逻辑表达式与真值表之间的等价转换关系,使得真值表在逻辑函数的化简中也扮演着非常重要的角色,具体内容请参见后续的【数字逻辑的化简】。

2.3.2.4 数字逻辑运算的基本公式与规律

本章节将为大家介绍一些常用的数字逻辑运算的基本公式与规律,它们的正确性都是可以通过公式推导或真值表来验证的,这里我们就不一一证明了。

1. 变量与常量

以下列举出一些常用的变量与常量之间在进行逻辑运算时的公式:

$$A \cdot 0 = 0; A \cdot 1 = 1$$

$$A + 0 = A; A + 1 = 1$$

$$A \cdot \bar{A} = 0; A + \bar{A} = 1$$

$$A \oplus 0 = A; A \oplus 1 = \bar{A}$$

$$A \odot 0 = \bar{A}; A \odot 1 = A$$

$$A \oplus \bar{A} = 1; A \odot \bar{A} = 0$$

2. 交换律

$$A \cdot B = B \cdot A$$

$$A + B = B + A$$

$$A \oplus B = B \oplus A$$

$$A \odot B = B \odot A$$

3. 结合律

$$A \cdot B \cdot C = (A \cdot B) \cdot C$$

$$A + B + C = (A + B) + C$$

$$A \oplus B \oplus C = (A \oplus B) \oplus C$$

$$A \odot B \odot C = (A \odot B) \odot C$$

4. 分配律

$$A(B + C) = AB + AC$$

$$A + BC = (A + B)(A + C)$$

$$A(B \oplus C) = AB \oplus AC$$

$$A + (B \odot C) = (A + B) \odot (A + C)$$

5. 重叠律

$$A \cdot A = A$$

$$A + A = A$$

$$A \oplus A = 0$$

$$A \odot A = 1$$

6. 反演律

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

$$\overline{A \oplus B} = A \odot B$$

$$\overline{A \odot B} = A \oplus B$$

7. 调换律

若 $A \oplus B = C$, 则必有

$$A \oplus C = B, B \oplus C = A$$

若 $A \odot B = C$, 则必有

$$A \odot C = B, B \odot C = A$$

简要证明同或的调换律如下:

$$A \odot C = A \odot (A \odot B) = A \odot A \odot B = 1 \odot B = B$$

根据调换律还可推出以下四个公式:

$$A \cdot B = A \odot B \odot (A + B)$$

$$A + B = A \oplus B \oplus (A \cdot B)$$

$$A \cdot B = A \oplus B \oplus (A + B)$$

$$A + B = A \odot B \odot (A \cdot B)$$

这是因为

$$(A \cdot B) \odot (A + B) = A \odot B \quad (\text{按照同或逻辑的与、或、非展开可得})$$

$$(A \cdot B) \oplus (A + B) = A \oplus B \quad (\text{按照异或逻辑的与、或、非展开可得})$$

2.3.2.5 数字逻辑运算中的三个规则

1. 代入规则

数字逻辑中的代入规则是指,任何一个含有变量 A 的等式,如果将所有出现 A 的地方都代之以一个逻辑函数 F ,则等式仍然成立。

这个结论是显然的,因为任何一个逻辑函数其实都和一个逻辑变量是一样的,只可能有 0、1 两种取值。例如,根据反演律,有如下等式:

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

令 $A = C + D$, 则代入上式后,等式仍然成立:

$$\overline{(C + D) \cdot B} = \overline{C + D} + \overline{B}$$

2. 反演规则

数字逻辑中的反演规则其实就是对反演律的一种总结,它使得我们可以简便、快速地求解出逻辑函数 F 的反函数,而不用利用反演律进行一步步的推导。

反演规则的内容即是:对于一个全部用与、或、非三种基本运算表示的逻辑函数 F ,如果将其内部的所有与操作符换为或操作符,所有或操作符换为与操作符;所有的原变量换为反变量,所有的反变量换为原变量;所有的常量 0 换为常量 1,所有的常量 1 换为常量 0。那么这样所得到的新函数表达式就是。

例如,我们可以直接用反演验证异或运算的反演律:

$$\overline{A \oplus B} = A \odot B$$

令

$$F = A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$$

那么利用反演规则,可得

$$\overline{F} = (\overline{A+B}) \cdot (A+\overline{B}) = \overline{A} \cdot A + \overline{A} \cdot \overline{B} + A \cdot B + \overline{B} \cdot B = \overline{A} \cdot \overline{B} + A \cdot B = A \odot B$$

3. 对偶规则

对偶规则的内容是:对于一个全部用与、或、非三种基本运算表示的逻辑函数 F ,如果将其内部的所有与操作符换为或操作符,所有或操作符换为与操作符;所有的常量 0 换为常量 1,所有的常量 1 换为常量 0。那么这样所得到的新函数表达式就是 F^* ,称为 F 的对偶函数。

可以证明,如果逻辑函数 $F=G$,那么 $F^*=G^*$ 。

因此利用对偶规则我们也可以简单地证明【数字逻辑运算的基本公式与规律】章节中的多个等式。例如,对于

$$A+BC=(A+B)(A+C)$$

如果对等式两边同时求对偶,则有

$$A(B+C)=AB+AC$$

2.3.3 数字逻辑的化简

同样功能的逻辑函数可以有繁简不同的表示形式,当表达形式简单时,其对应的数字电路所使用的元器件就会相应地少一些;当表达式复杂时,其对应的数字电路所使用的元器件就会相应地多一些。因此,为了得到更加简洁的数字电路,我们很有必要对待实现的逻辑函数首先进行一下化简操作。那么本章节就来介绍几种常用的逻辑化简方法。

2.3.3.1 公式化简法

本小节介绍几个常用的、用于逻辑化简的公式。

1. 公式一

$$AB+A\overline{B}=A$$

该公式也称为吸收律,它的意义为:如果两个乘积项,除了共有因子外,不同因子恰好互补,则这两个乘积项可以合并为一个由公有因子组成的乘积项。例如:

$$ABCD+AB\overline{C}D=AB$$

那么利用对偶性质,可得另一吸收律公式:

$$(A+B) \cdot (A+\overline{B})=A$$

2. 公式二

$$A+AB=A$$

这个公式的意义是,如果两个乘积项求和,其中一个的部分因子恰好是另一个乘积项的全部,那么该乘积项即是多余的,这是因为 AB 肯定是 A 的子集。

那么利用对偶性质,可得另一公式如下:

$$A(A+B)=A$$

3. 公式三

$$A+\overline{A}B=A+B$$

该公式可以通过分配律轻松得证,当然了,也可以利用公式二、公式一联合推出,即

$$A + \overline{A}B = A + AB + \overline{A}B = A + (AB + \overline{A}B) = A + B$$

这个公式的意义是,如果两个乘积项求和,其中一个乘积项的部分因子恰好是另一个乘积项的补,则该乘积项中的这部分因子是多余的。

那么利用对偶性质,可得另一公式如下:

$$A(\overline{A} + B) = AB$$

4. 公式四

$$AB + \overline{A}C + BC = AB + \overline{A}C$$

该公式可以利用公式二证明如下:

$$AB + \overline{A}C + BC = AB + \overline{A}C + (A + \overline{A})BC = AB + \overline{A}C + ABC + \overline{A}BC = AB + \overline{A}C$$

按照上述思路,可轻松得证该公式的一个推论,即:

$$AB + \overline{A}C + BCDE = AB + \overline{A}C$$

该推论的意义为,三个乘积项求和时,如果两个乘积项中的部分因子恰好互补,而这两个乘积项中的其他因子都是第三个乘积项中的因子,那么这第三个乘积项是多余的。

那么利用对偶性质,可得另一公式如下:

$$(A + B)(\overline{A} + C)(B + C) = (A + B)(\overline{A} + C)$$

2.3.3.2 卡诺图化简法

1. 逻辑函数的标准形式

从之前介绍的各种规律及公式可以了解到,对于一个固定功能的逻辑函数来说,其表达式并不是唯一的。那么,逻辑函数标准形式的出现就是为了规范、统一逻辑函数的表达形式。

(1) SOP 和 POS

SOP 是 sum of product 的缩写,意思为积之和表达式,也即与-或表达式。SOP 表达式共分为两级,低一级为若干个纯粹的纯“与”表达式(乘积项),而高一级则为所有的低一级“与”表达式求“或”(求和),进而得出逻辑函数的结果。例如:

$$F = AB + CD$$

$$F = ABC + \overline{CD}$$

$$F = AB + CD + EF + MN$$

等等,这些都属于 SOP 表达式。

POS 是 product of sum 的缩写,意思为和之积表达式,也即或-与表达式。POS 表达式共分为两级,低一级为若干个纯粹的纯“或”表达式(求和项),而高一级则为所有的低一级“或”表达式求“与”(求积),进而得出逻辑函数的结果。例如:

$$F = (A + B)(C + D)$$

$$F = (A + B + C)(C + \overline{D})$$

$$F = (A + B)(C + D)(E + F)(M + N)$$

等等,这些都属于 POS 表达式。

而类似下列逻辑函数的表达式,则既不属于 SOP 也不属于 POS:

$$F = (A + B)C + D$$

$$F = (AB + BC)\overline{D}$$

(2) 最小项与最大项

对于 SOP 表达式来说,若其中的某个乘积项包含了所有的输入变量(每个输入变量以原变量或反变量的形式出现),则该乘积项称做最小项。

例如,对于一个三输入的逻辑函数 $F1(A, B, C)$ 来说,以下列出了其全部最小项:

$$ABC, ABC, \overline{A}BC, \overline{A}BC, \overline{A}BC, \overline{A}BC, \overline{A}BC, \overline{A}BC$$

之所以称做最小项,是因为这样的乘积项,在输入随机的情况下,结果为真(即为 1)的可能性最小。例如,对于上述三输入逻辑函数 $F1$ 来说,输入的组合情况共有 8 种,但是对于每一个最小项来说,仅有 1 种输入情况能够使其为真。以 ABC 为例,仅有当所有输入均为 1 时,该乘积项的结果才为真,否则其结果为假(即为 0)。

如果按照 A, B, C 的取值能使各项为真的二进制编码情况,上述最小项也可依次称为:

$$m7, m6, m5, m4, m3, m2, m1, m0$$

对于 POS 表达式来说,若其中的某个求和项包含了所有的输入变量(每个输入变量以原变量或反变量的形式出现),则该求和项称做最大项。

例如,对于一个三输入的逻辑函数 $F2(A, B, C)$ 来说,以下列出了其全部最大项:

$$(A+B+C), (A+B+\overline{C}), (A+\overline{B}+C), (A+\overline{B}+\overline{C}), \\ (\overline{A}+B+C), (\overline{A}+B+\overline{C}), (\overline{A}+\overline{B}+C), (\overline{A}+\overline{B}+\overline{C})$$

之所以称做最大项,是因为这样的求和项,在输入随机的情况下,结果为真(即为 1)的可能性最大。例如,对于上述三输入逻辑函数 $F2$ 来说,输入的组合情况共有 8 种,但是对于每一个最大项来说,仅有 1 种输入情况能够使其为假。以 $(A+B+C)$ 为例,仅有当所有输入均为 0 时,该求和项的结果才为假,否则其结果为真。

如果按照 A, B, C 的取值能使各项为假的二进制编码情况,上述最大项也可依次称为:

$$M0, M1, M2, M3, M4, M5, M6, M7$$

(3) 标准式

如果一个逻辑函数 F 的表达式为 SOP 形式,且其中的乘积项均为最小项,那么该表达式称为 SOP 标准式,也即标准与-或式。

如果一个逻辑函数 F 的表达式为 POS 形式,且其中的求和项均为最大项,那么该表达式称为 POS 标准式,也即标准或-与式。

因此,逻辑函数的标准形式主要有两种,即 SOP 标准式和 POS 标准式,而在今后的讨论中,我们将更多地围绕更为常用的 SOP 标准式展开介绍。

2. 从真值表到标准式

从真值表可以轻松写出逻辑函数的标准形式。因为真值表中的每一个表项就对应了所有输入组成的一个最小项,而真值表中除了某一个表项之外的其他所有表项则对应了所有输入组成的一个最大项。

因此,当拿到一个真值表后,找出所有输出为 1 的表项,按照输入的情况,为 1 用原变量表示,为 0 则用反变量表示,得出若干乘积项,然后求和,这样得到的函数表达式即为 SOP 标准式;如若找出所有输出为 0 的表项,按照输入的情况,为 1 用反变量表示,为 0 则用原变量表示,得出若干求和项,然后求积,这样得到的函数表达式即为 POS 标准式。

例如,有如下真值表:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

若针对所有 F=1 的表项,可轻松写出 SOP 标准式如下:

$$F=ABC+\overline{A}B\overline{C}+\overline{A}B\overline{C}+\overline{A}BC$$

通过【最小项与最大项】中的学习,上式也可以简写作:

$$F=\sum m(3,4,6,7)$$
 (1)

若针对所有 F=0 的表项,可轻松写出 POS 标准式如下:

$$F=(\overline{A}+B+\overline{C})(A+\overline{B}+C)(A+B+\overline{C})(A+B+C)$$

通过【最小项与最大项】中的学习,上式也可以简写作:

$$F=\prod m(0,1,2,5)$$
 (2)

对比(1)、(2)两式,可以看出,同一函数的 SOP 与 POS 标准式的索引具有互补性,可以利用这点轻松地在两者之间进行变换。

3. 从标准 SOP 式到最简 SOP 式

标准表达式并非最简表达式,例如,针对前面的 SOP 标准式,其中的前两个乘积项 ABC 和 $\overline{A}B\overline{C}$,就可以利用【公式化简法→公式一】中介绍的公式来进行化简合并。那么,从标准 SOP 式到最简 SOP 式的求解过程,即为一个标准的逻辑化简的过程。

如果单纯利用公式法,化简的程度受限于个人的判断能力,当乘积项较多时,就更难剔除出所有的冗余。因此,相比于公式法的不确定性,我们接下来将重点介绍一种基于卡诺图的逻辑化简方法。

4. 从真值表到卡诺图

通过之前的介绍,我们知道,要想从真值表推出标准 SOP 式,那是非常简单的,但是要想从标准 SOP 式利用公式法推出最简 SOP 式,就不那么简单了,甚至有时候还会有些困难。因此,必须寻找别的途径来得出最简 SOP 式。

通过【公式化简法】中的介绍,我们发现其实从标准 SOP 式到最简 SOP 式,主要就是利用“公式一”来进行最小项的不断合并,当然,一个最小项可以被多次利用。而可以被合并的两个最小项,它们之间的差别仅仅是有一个变量的表现形式从原变量转换到反变量而已。因此,如果我们能够建立一张关于最小项的表格,使得表格中相邻的这些最小项都仅有一个变量的表现形式不同,那么将更容易看出最小项的合并规律,从而可以轻易的进行化简工作。没错,这个表格就是卡诺图!

卡诺图在安排最小项时,利用了循环码的特性,即相邻之间仅有一位不同的编码方法,并采用特定的填表顺序,使得所有相邻的最小项之间都仅有一个变量的表现形式不同。下面列举出二输入、三输入、四输入的卡诺图分布:

2 输入	A=0	A=1
B=0	m0	m2
B=1	m1	m3

3 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0	m0	m2	m6	m4
C=1	m1	m3	m7	m5

4 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0	m0	m4	m12	m8
D=0				
C=0	m1	m5	m13	m9
D=1				
C=1	m3	m7	m15	m11
D=1				
C=1	m2	m6	m14	m10
D=0				

需要指出,对于 5 输入以上的卡诺图,由于受限于二维空间的表格形式,因此其最小项之间的相邻关系就不像 4 输入及以下情况那样一目了然,因此卡诺图通常只适合于 4 输入及以下的逻辑函数化简。

5. 卡诺图的相邻情况

如果相邻的两个最小项均出现在标准 SOP 式中,那么它们就可以合并化简;同理,如果合并后的两个次小项也相邻,那么它们可以进一步进行合并化简。因此,下面列举出 4 输入及以下的卡诺图中所有的相邻情况。

(1) 2 输入卡诺图相邻示例

2 个相邻 1 项合并的例子:

2 输入	A=0	A=1
B=0	1	1
B=1		

2 输入	A=0	A=1
B=0	1	
B=1	1	

4 个相邻 1 项合并的例子:

2 输入	A=0	A=1
B=0	1	1
B=1	1	1

(2)3 输入卡诺图相邻示例
2 个相邻 1 项合并的例子：

3 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0	1	1		
C=1				

3 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0	1			
C=1	1			

3 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0	1			1
C=1				

4 个相邻 1 项合并的例子：

3 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0	1	1		
C=1	1	1		

3 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0	1	1	1	1
C=1				

3 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0	1			1
C=1	1			1

8 个相邻项合并的例子：

3 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0	1	1	1	1
C=1	1	1	1	1

(3)4 输入卡诺图相邻示例
2 个相邻 1 项合并的例子：

4 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0 D=0	1	1	1	
C=0 D=1	1			1
C=1 D=1				1
C=1 D=0			1	1

4 个相邻 1 项合并的例子：

4 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0 D=0	1			1
C=0 D=1		1	1	
C=1 D=1		1	1	
C=1 D=0	1			1

4 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0 D=0		1	1	
C=0 D=1	1			1
C=1 D=1	1			1
C=1 D=0		1	1	

4 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0 D=0			1	
C=0 D=1	1	1	1	1
C=1 D=1			1	
C=1 D=0			1	

8 个相邻 1 项合并的例子：

4 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0 D=0	1	1	1	1
C=0 D=1		1	1	
C=1 D=1		1	1	
C=1 D=0	1	1	1	1

4 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0 D=0	1			1
C=0 D=1	1			1
C=1 D=1	1	1	1	1
C=1 D=0	1	1	1	1

16 个相邻 1 项合并的例子：

4 输入	A=0	A=0	A=1	A=1
	B=0	B=1	B=1	B=0
C=0 D=0	1	1	1	1
C=0 D=1	1	1	1	1
C=1 D=1	1	1	1	1
C=1 D=0	1	1	1	1

6. 卡诺图的化简

简单地说,卡诺图的化简分为两个主要步骤——填图和合并,接下来分别进行介绍。

(1) 填图

填图,即根据逻辑函数 F 的表达式,得出其所对应的卡诺图。

一个比较中规中矩的方法是,先根据 F 的表达式得出其对应的真值表,这也就相当于得到了标准 SOP 式,这样便可以根据【逻辑函数的标准形式→从真值表到卡诺图】完成卡诺图的填图。

不过实际操作中,上述操作有些过于繁琐,其实我们只需要将 F 的表达式利用分配律等规律展开成普通的 SOP 式,这样一来,我们便可以轻松将其中的每一个乘积项分别对应填充到卡诺图中,因为根据乘积项所包含的变量数不同,其实每一个乘积项就对应 1 个最小项、2 个相邻的最小项、4 个相邻的最小项、8 个相邻的最小项、甚至 16 个相邻的最小项。当然了,由于此时的乘积项不一定是最小项,因此填图时很可能出现重叠现象,不过这并不会影响填图的结果。

例如,对于如下 4 输入逻辑函数:

$$F = ABC + ABC\bar{C} + D + \bar{A}BCD$$

对每一个乘积项进行填图,可得最终的卡诺图如下所示:

4 输入	A=0 B=0	A=0 B=1	A=1 B=1	A=1 B=0
C=0 D=0			1	
C=0 D=1	1	1	1	1
C=1 D=1	1	1	1	1
C=1 D=0			1	

(2) 合并

得到卡诺图后,就可以依据【卡诺图的相邻情况】中介绍的相邻情况进行逻辑化简了。其主旨就是利用最少、最大的圆圈来覆盖掉所有为 1 的最小项。具体的做法可以参考以下步骤。

步骤一:依次遍历每一个最小项,并以该最小项为必要包含项,画出尽可能大的相邻关系圈(再大就包含 0 了),当然,对于每一个最小项来说,其衍生出的最大相邻关系圈可能不止 1 个。绘制过程中,如果以某一最小项为基础所得到的某个最大相邻关系圈和之前遍历过的某个最小项所得到的某个最大相邻关系圈完全重叠,则无需画出。注意,此次所得到的若干个最大相邻关系圈也称做卡诺图的主要项。

步骤二:依次遍历上一步所得到的那些最大相邻关系圈,也即主要项,如果某一主要项中至少包含一个不被其他主要项包含的 1 格,那么该主要项称做必要项或实质素项、实质本源蕴含项等;否则,该主要项称为多余项,将其从卡诺图中删除。

步骤三:针对卡诺图中剩余的那些最大相邻关系圈,写出化简后的逻辑函数表达式。

例如,接【填图】中所举的例子,经过步骤一后,可得:

4 输入	A=0 B=0	A=0 B=1	A=1 B=1	A=1 B=0
C=0 D=0			1	
C=0 D=1	1	1	1	1
C=1 D=1	1	1	1	1
C=1 D=0			1	

经过步骤二后,发现两个主要项均为必要项,因此在步骤三即可得到化简后的逻辑表达式为

$$F = AB + D$$

2.3.3.3 系统化简法

卡诺图化简法仅适用于变量个数较少的逻辑函数化简情况,因此人们又总结出了名为系统化简法的逻辑化简方法。

系统化简法又称奎恩—麦克洛斯基法,简称 Q-M 法,相比于卡诺图法,它的计算要繁琐很多,但是它有着严格的算法,适用于化简任意多变量的逻辑函数,因此就可以将其化简思路编写成程序,利用计算机来解决问题。

接下来,就来分步讲解一下系统化简法的思路。

1. 求出函数的 SOP 标准式

系统化简法的第一步,是先从逻辑函数的表达式得出其 SOP 标准式,这可以简单地通过列真值表或乘积项分解等方法求得,这里就不再赘述。

例如,对于函数:

$$F = ABC + ABC + D + \overline{A}BCD$$

可写出其标准 SOP 式为

$$F = \sum m(1, 3, 5, 7, 9, 11, 12, 13, 14, 15)$$

2. 求出函数的全部主要项

系统化简法的第二步,是利用【公式化简法→公式一】小节介绍的合并规律,来对 SOP 标准式中的所有最小项进行不断的合并,从而求出函数的全部主要项。其具体操作步骤紧接上一小节的例子说明如下:

对于标准式

$$F = \sum m(1, 3, 5, 7, 9, 11, 12, 13, 14, 15)$$

首先,将最小项按其内部包含 1 的个数多少进行排列、分组,可得下表:

m	<u>ABCD</u>	标记
1	<u>0001</u>	✓
3	0011	✓
5	0101	✓
9	1001	✓
12	<u>1100</u>	✓
7	0111	✓
11	1011	✓
13	1101	✓
14	<u>1110</u>	✓
15	1111	✓

其次,根据该表,可以发现能够利用“公式一”进行合并的两个最小项必定位于相邻的两组,因此从最低组开始,和相邻高位组逐个运算合并,并仍按照乘积项中1的个数进行排列可得新的表如下:(注意:①在进行合并的同时,需要在之前的表中用“✓”标注出被使用过的最小项;②如果合并结果与之前的某次相同,则不需再列出来。)

m	<u>ABCD</u>	标记
1,3	00—1	✓
1,5	0—01	✓
1,9	—001	✓
3,7	0—11	✓
3,11	—011	✓
5,7	01—1	✓
5,13	—101	✓
9,11	10—1	✓
12,13	110—	✓
12,14	<u>11—0</u>	✓
7,15	—111	✓
11,15	1—11	✓
13,15	11—1	✓
14,15	111—	✓

第三,参考前两个步骤,继续对表项进行合并,直至无法合并为止。注意,此后的合并,有很多规律可循,不过最实用的就是“—”的位置要一样。继续合并的结果如下:

m	<u>ABCD</u>	标记
1,3,5,7	0— —1	✓
1,3,9,11	—0—1	✓
1,5,9,13	<u>— —01</u>	✓
3,7,11,15	— —11	✓
5,7,13,15	—1—1	✓

9,11,13,15 1--1 ✓

12,13,14,15 11-- P0

m

ABCD 标记

1,3,5,7,9,11,13,15 ---1 P1

第四,上述各表中,凡是没被“✓”标记的合并项,就是主要项。对于该例,主要项就是: $P0=AB$ 和 $P1=D$ 。

3. 求出必要项、列出化简结果

与卡诺图类似,系统化简法所得出的主要项,其中也肯定包含必要项和多余项。除此以外,由于系统化简法针对的是任意多变量的逻辑化简,因此可能还会存在一些比较模棱两可的主要项,例如,有时候,两个主要项单独来看可能都是多余项,但是如果删掉一个,另一个就变成了必要项。因此,在删除多余项时需要一个一个来进行,删除一个之后需重新审视修改后的列表一遍。当然,这也从侧面说明这两个主要项是等价的,取其一即可。

正因为如此,系统化简法最后得出的必要项可以有多种可能,其中任何一种都是正确的求解。那么,求出所有必要项的方法介绍如下。

求必要项的过程也是不断列表求解,仍接上例,列表如下:

	$m1$	$m3$	$m5$	$m7$	$m9$	$m11$	$m12$	$m13$	$m14$	$m15$
P0						
P1

上表中,横向表头列举出所有该逻辑函数的最小项,纵向表头列出所有的主要项,而在表中,我们通过“.”号标注出每一个主要项所包含的最小项。如果某一个主要项中,至少包含一个其他项不包含的最小项,则它必然是必要项。针对上例,可见 $P0$ 、 $P1$ 均是必要项,因此上例的最终化简结果自然也是 $F=AB+D$ 。

下面,列举一个存在等价主要项的例子,为了简便起见,直接列出其求必要项的初始列表如下:

	$m0$	$m3$	$m4$	$m5$	$m6$	$m7$	$m8$	$m10$	$m11$
P0					
P1								.	.
P2		.							.
P3		.				.			
P4							.	.	
P5	.						.		
P6	.		.						

先看 $P0$,它显然是一个必要项,因为 $m5$ 、 $m6$ 都只属于它。

再看 $P1$,由于 $m10$ 也存在于 $P4$ 中, $m11$ 也存在于 $P2$ 中,因此认为它是一个多余项,将其删掉后得新表如下:

	<i>m</i> 0	<i>m</i> 3	<i>m</i> 4	<i>m</i> 5	<i>m</i> 6	<i>m</i> 7	<i>m</i> 8	<i>m</i> 10	<i>m</i> 11
P0			•	•	•	•			
P2		•							•
P3		•				•			
P4							•	•	
P5	•						•		
P6	•		•						

接着看 P2,此刻它已然成为一个必要项,因为 *m*11 只属于它,但是如果再回顾最开始的列表,可以发现其实它也可以被判断为多余项,这便可以看出上述主要项中存在着等价主要项,只不过由于先删除了一个,才导致另一个或几个成为了必要项。当然了,先删除哪个完全取决于遍历上表的顺序。

然后看 P3,由于 *m*3 也属于 P2,*m*7 也属于 P0,因此这是一个多余项,删除后得新表:

	<i>m</i> 0	<i>m</i> 3	<i>m</i> 4	<i>m</i> 5	<i>m</i> 6	<i>m</i> 7	<i>m</i> 8	<i>m</i> 10	<i>m</i> 11
P0			•	•	•	•			
P2		•							•
P4							•	•	
P5	•						•		
P6	•		•						

继续看 P4,由于此刻 *m*10 仅属于它,因此它现在是一个必要项。

再看 P5,由于 *m*0 也属于 P6,*m*8 也属于 P4,因此这是一个多余项,删除后得新表:

	<i>m</i> 0	<i>m</i> 3	<i>m</i> 4	<i>m</i> 5	<i>m</i> 6	<i>m</i> 7	<i>m</i> 8	<i>m</i> 10	<i>m</i> 11
P0			•	•	•	•			
P2		•							•
P4							•	•	
P6	•		•						

最后看 P6,发现此刻它也成为了一个必要项,最后,表中剩余的即为表示该逻辑函数的一种所有必要项组合,可写出其化简后的表达式:

$$F=P0+P2+P4+P6=\overline{A}B+\overline{B}CD+A\overline{B}\overline{D}+\overline{A}\overline{C}\overline{D}$$

上述求解过程为笔者自创,暂称其为“逐行消去法”,相比于其他书本中提到的“行列消去法”,这种方法更为简单和实用。

2.3.4 数字逻辑功能单元

数字逻辑最终是需要通过数字电路的形式来实现的,那么为了方便对数字电路进行图形化的描述,就好比使用电阻、电容、电感等图形化原件来描述模拟电路一样,我们需要为数字电路引入一些逻辑功能单元。

提供各种书籍的pd电子版代找服务，如果你找不到自己想要的书的pdf电子版，我们可以帮您找到，如有需要，请联系QQ1779903665.

PDF代找说明：

本人可以帮助你找到你要的PDF电子书，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签索引和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我QQ1779903665。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ，大部分我都可以找到，而且每本100%带书签索引目录。因PDF电子书都有版权，请不要随意传播，如果您有经济购买能力，请尽量购买正版。

声明：本人只提供代找服务，每本100%索引书签和目录，因寻找pdf电子书有一定难度，仅收取代找费用。如因PDF产生的版权纠纷，与本人无关，我们仅仅只是帮助你寻找到你要的pdf而已。