轻松实现高速串行I/O

FPGA 应用设计者指南

作者：

Abhijit Athavale

Carl Christensen

XILINX®

轻松实现高速串行I/O

*FPGA**应用设计者指南***

作者:

Abhijit Athavale

Xilinx公司连接功能解决方案部市场营销经理

和

Carl Christensen

Xilinx公司技术营销部

获取本书复印件，请致信：

Xilinx Connectivity Solutions
Product Solutions Marketing/Xilinx Worldwide Marketing
Dept. 2450, 2100 Logic Drive, San Jose, CA 95124
电话：408.879.6889，传真：308.371.8283
serialio@xilinx.com

初版1.0
2005年**4**月
PN0402399

致谢

在此，请允许我们向Paul Galloway和Craig Abramson致以最诚挚的谢意。我们能够完成这个项目，离不开他们的激励、指导和鼓励。

同时，我们也非常感谢Ryan Carlson为本书的编写所提供的宝贵帮助，以及Chuck Berry所给与的大力支持和为本书的销售所做出的努力。

向包括Matt DiPaolo、Mike Degerstrom和Scott Davidson在内的一大批校阅者表达我们的谢意。是他们让我们具备了诚实、准确和与时俱进的作风。

感谢Babak Hedayati和Tim Erjavec给与我们的坚定支持和鼓励。最后，特别感谢Ray Johnson，他不仅为我们提供了全力支持，而且还审阅了全书并撰写了开篇序言。

千兆位串行 I/O 设计

Xilinx—您的设计合作伙伴

SERDES示例资料 - RocketIO X 收发器概述

序言

“军队的入侵可以抵抗，思想的入侵防不胜防。”

- Victor Hugo (1802 –1885) 'Histoire d'un crime,' 1852

一生中有幸成为新发现或新思想的一部分的机会屈指可数。某些思想或革新会极大地改变我们所生活的世界。想一想如果生物科学家完成了整个人类基因的绘制 — 确定了DNA结构的最后一个基因，美国国立卫生研究院实验室会作何感想。或者当Bardeen、Brattain和Shockley演示第一个引发通信变革的晶体管时，贝尔实验室会有什么反应。

在过去的50年里，科学家和工程师取得了数量惊人的科技突破。他们提出的思想改变了我们的思维方式和几乎每一件事情的做事方法。例如，连接研发中心计算机的愿望演变成了今天的互联网 - 对于这项创新，很多人认为这是我们一生中所看到的最重要的、改变了商业、社会和政治状况的工具。

如今，我们能够再一次见证并分享这些罕见的技术发现。电子行业正在经历一场根本性的转变 - 从并行I/O电路到串行I/O连接功能解决方案的转变。作为一种能够降低系统成本、简化系统设计并提供所需的扩展性，从而满足新的带宽需求的手段，这种转变受到了各行业企业的推动。

Xilinx坚定地相信串行连接功能解决方案最终将应用到可能的电子产品的方方面面。简单地举几个例子，这种解决方案可用于芯片到芯片的接口、背板连接功能和系统板、以及盒到盒的通信。我们支持这种信念，并发起了“高速串行创新行动”，从而加速了产业从并行技术向高速串行I/O技术的转变。这一行动包括为系统设计提供新一代连接功能解决方案，来满足从622Mb/s到11.1Gb/s，甚至更高的带宽要求。

行业分析师均认为高速串行创新行动是不可避免的，因为在数据速率超过1Gb/s而且不再能够为保持信号同步提供可靠、经济的方法时，并行I/O电路达到了其物理极限。与传统的并行实现方法相比，基于串行I/O的设计具有很多优势，包括：器件引脚数较少、降低了板空间要求、印刷电路板(PCB)层数较少、可以轻松实现PCB设计、连接器较小、电磁干扰降低并具有较好的抗噪能力。

从并行向串行的转变不会没有工程挑战。其中最大的挑战在于理解设计高速串行I/O解决方案具有很大的难度和复杂度，以至于尽管现有的并行技术有很多的劣势，系统工程师也宁愿继续使用它。为了解决这一挑战，我们编写了《*串行I/O入门指南*》。它为那些对快速发展的串行技术感兴趣而又对迈出第一步犹豫不决的设计者提供了援助。对于那些已经开始利用串行技术进行设计的读者而言，本书可作为深入的进修课程使用。

通过高速串行创新行动，Xilinx为一系列串行系统结构提供了专业技术支持和完善的、预先设计好的解决方案。这些系统结构包括：网络、电信和企业存储市场，Xilinx为这些市场提供了终极连接功能平台 - 集成有串行I/O收发器的Xilinx平台FPGA。

除了撰写本书，Xilinx还积极参与主要的行业组织，帮助推动串行技术标准。而且，Xilinx还提供了广泛的“生态系统”合作伙伴网络（EDA、参考设计、IP、设计服务等），从而保证了互操作性和对最新技术、

技巧和设计工具的使用。

世界拥护串行技术。高速串行I/O包含的器件（如FPGA内的RocketIO ™收发器）使串行技术成为首选的系统连接功能解决方案。我们也承认对于熟悉并行I/O技术的大多数设计者而言，高速串行设计的很多挑战难题仍然很陌生。《*串行I/O入门指南*》提供了串行I/O设计的基本原理，从而使所有人都能正确地应用这项创新技术。

Raymond R. Johnson

Xilinx公司通信技术部副总裁兼总经理

关于作者

Abhijit Athavale

Abhijit Athavale是Xilinx公司连接功能解决方案部市场营销经理，其职责包括为公司的高速串行和并行连接功能产品完成战略开发、产品定位和营销计划。自1995年加入Xilinx以来，他担任过营销、应用和软件工程方面的多种职务。之前，Athavale曾任Meltron公司研发工程师之职，主要设计通信产品。他拥有印度旁尼大学电子工程学士学位和德克萨斯农业大学电子工程硕士学位。他是一名很有造诣的演说家和作家，发表了数篇论文。

Carl Christensen

Carl从事硬件和软件设计已超过16年，目前专攻用于Thomson（品牌名称，包括RCA、Technicolor和Grass Valley）的领先FPGA设计和系统架构。

Carl的著作包括在Synopsys User Group大会（SNUG）、全国广播工作者联合会（NAB）大会和Xilinx专家级用户座谈会上发表的技术论文。目前，他拥有前向纠错和广播传送系统领域的16项专利/应用。Carl拥有犹他州立大学电子工程学士学位，并在计算机科学方面取得了优异成绩，曾在行业内和大学中传授HDL设计和编程课程。

第一章

介绍

数字I/O信号处理方法概要

I/O性能极限

输入/输出（I/O）在计算机和工业应用中一直扮演着关键角色。但是，随着信号处理越来越复杂，I/O通信会变得不可靠。在早期的并行I/O总线中，接口的数据对齐问题影响着与外部设备的有效通信。并且，随着更高的传输速度在数字设计中日渐普及，对信号延迟的管理也变得困难重重。

针对I/O的数字设计解决方案

数字电路设计者采用了一系列方法来提高信号速度和消除I/O问题。例如，采用差分信号处理来提高芯片间的通信速度。信号同步、源同步和自同步之类的设计方法改善了内部IC（集成电路）通信，在满足计算机行业所需速度的前提下，提供了可靠的输入/输出。

千兆位级串行技术介绍

图1-1为典型的数字信号。

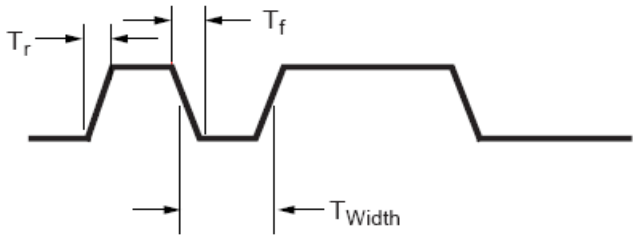


图1-1：**标准数字信号**

注意图中列出的时间测量值：

T_R = 20 ps

T_F = 20 ps

T_{WIDTH} = 0.10 ns

这些值描绘出了一个变化*很快的*波形。图1-2添加了作为参考的历史信号，以便说明该波形的变化有多快。

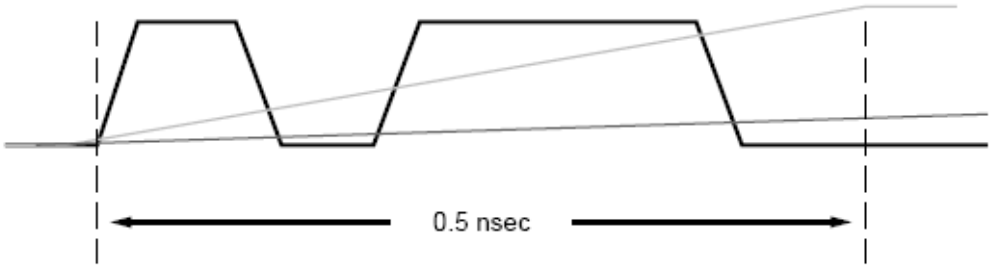


图1-2：**添加历史信号**

大多数信号的上升时间甚至不能在这个信号的五个比特周期内结束。那么，为什么要讨论这个信号呢？因为它代表了数字 I/O 领域最热门的潮流——千兆位级串行通信。

这类信号在市场上引起轩然大波。它被广泛采用，从局域网（LAN）设备到尖端医疗成像设备，再到先进的战斗机技术，不一而足。千兆位级信号迅速成为延伸信息化时代的关键因素。为了解这一飞速发展的科技进步技术，让我们首先回顾一下I/O设计的历史。

数字电子通信的历史

晶体管-晶体管逻辑电路（TTL）曾是首选的设计方法。分立的逻辑门芯片可以相互通信从而组成更大规模的电路，这些更大规模的电路后来又集成为复杂的芯片，例如多比特寄存器和计数器。多年来，并行通信一直是印刷电路板（PCB）所采用的主要通信方法。但对于外部通信而言，数据对齐问题太难解决。因此，串行端口成为了设备盒到设备盒之间通信的主要方法，这一点在早期的计算机串行打印机端口中得到了证实。

> 了解一项新技术的关键是了解它的词汇。整本书中，您都会发现这种对**关键术语**的定义。

数据对齐问题最终得以解决。随着并行技术的发展，高速并行打印机端口激增。这些并行端口技术包括工业标准架构（ISA）、扩展工业标准架构（EISA）和小型计算机系统接口（SCSI）、外围部件互联（PCI）以及更小的个人电脑存储器卡行业协会（PCMCIA）标准。

串行技术继续与并行技术共存。以太网和令牌环在许多应用中占主导地位。最终，以太网在使用五类线（Cat5）的应用中取代了令牌环。

并行技术竭力适应新的接口需求。随着对特殊信令越来越多的需求，PCI 33之类的标准也发展成PCI 66。目前，低摆幅标准（如高速晶体管逻辑（HSTL））试图支持并行技术。同时，以太网速率从10Mb/s提高到100Mb/s，进而达到1000Mb/s。这类速度使得以太网非常适于应用在桌上型电脑中。

这时，**分数**相位检测器（fractional phase detector）面世了。这项技术将串行接口速度提高到了千兆位级的范围。串行技术被证明是快速而功能强大的，可作为背板技术采用。随着串行引脚数和同步开关输出（SSO）的提高，千兆位级串行技术在PCB中获得了主导地位，取代了并行技术。

基本I/O概念

单端I/O成为标准已有数年光景。单端系统中，2个IC间仅用单一的信号连接。该信号与指定的电压范围（TTL CMOS [互补金属氧化物半导体]）或参考电压（HSTL）进行比较。这些方法的指标示例如图1-3所示。

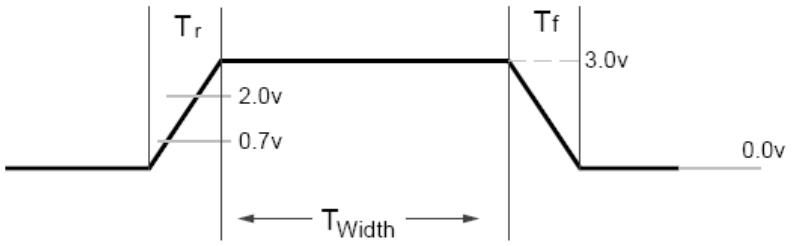


图1-3：**TTL波形**

表1-1：**LVCMOS（低压CMOS）电压指标**

参数	最小值	典型值	最大值
V_{CCO}	2.3	2.5	2.7
V_{REF}	-	-	-
V_{TT}	-	-	-
V_{IH}	1.7	-	3.6
V_{IL}	-0.5	-	0.7
V_{OH}	1.9	-	-
V_{OL}	-	-	0.4
V_{OH}时的I_{OH} (mA)	-12	-	-
V_{OL}时的I_{OL} (mA)	12	-	-

表1-2：**HSTL电压指标**

参数	最小值	典型值	最大值
V_{CCO}	1.4	1.5	1.6
V_{REF}	0.68	0.75	0.90
V_{TT}	-	V_{CCO} x0.5	-
V_{IH}	V_{REF}+0.1	-	-
V_{IL}	-	-	V_{REF}-0.1
V_{OH}	V_{CCO}-0.4	-	-
V_{OL}	-	-	0.4
V_{OH}时的I_{OH}(mA)	-8	-	-
V_{OL}时的I_{OL} (mA)	8	-	-

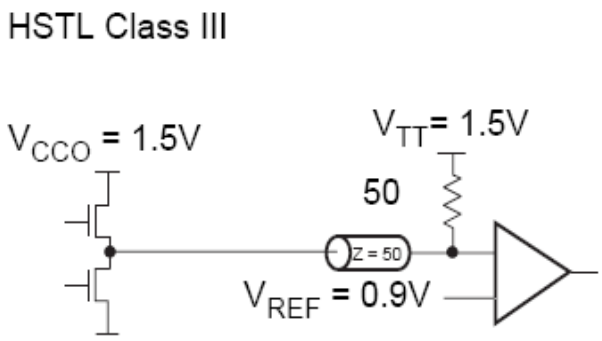


图1-4：**HSTL电压结构图**

差分信号

大约在HSTL以及其他低电压摆幅标准流行起来的时候，一种差分信号方式开始出现在芯片间通信中。差分信号出现很久了，但是它们主要用于长距离传输，而不用于PCB上的芯片间通信（图1-5）。

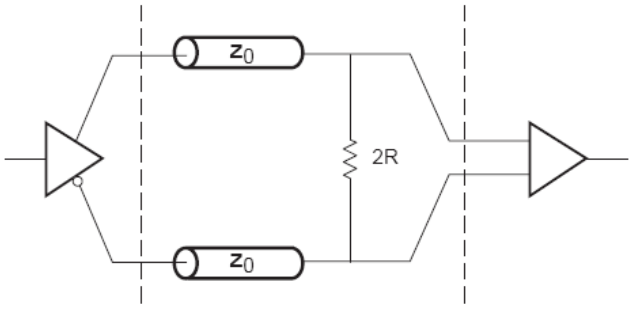


图1-5：**差分信号方法**

随着IC通信速度的提高，系统和IC设计者开始寻找可以处理更高速度的信令方法（图1-6）。差分信令就是这样的方法。与单端信令相比，差分信令有几点优势。例如，它受噪声的影响很小，这有助于保持恒定的IC驱动电流。另外，其信号并不同给定电压值或参考电压进行比较，而是在2个信号之间互相进行比较。这样，如果作为正参考电压的信号电压比作为负参考电压的信号电压高，则信号为高，或为“1”。如果作为负参考电压的信号电压较高，则信号为低，或为“0”。如下图所示，正、负引脚受精确互补信号驱动。

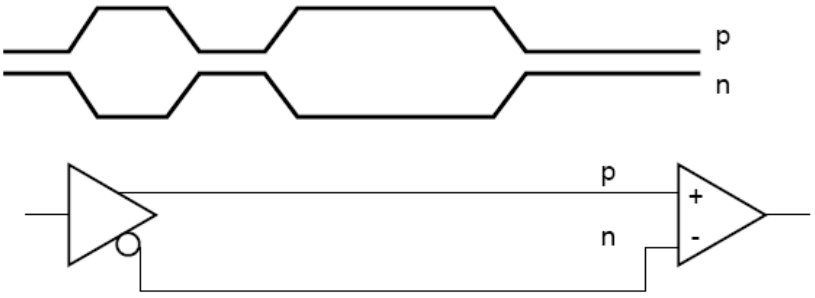


图1-6: **信令方法**

系统同步、源同步和自同步

有三种用于两个IC间通信的时序模型——系统同步、源同步和自同步。

系统同步

如图1-7，这种方法是多年来最常用的。乍一看会觉得它很简单，但看过图1-8中的时序模型，您就不会这么认为了。阴影框部分表示：为确保可靠，接收电路必须予以处理和抵消的延迟。

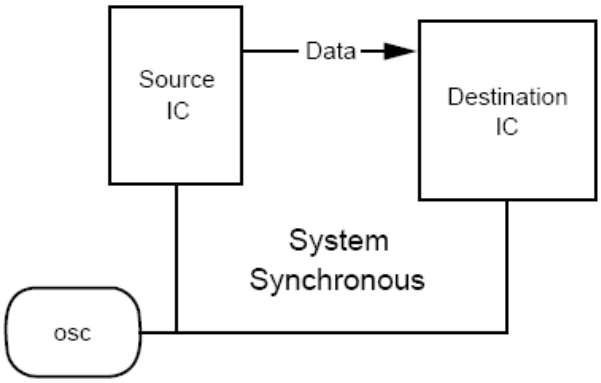


图1-7：**系统同步结构图**

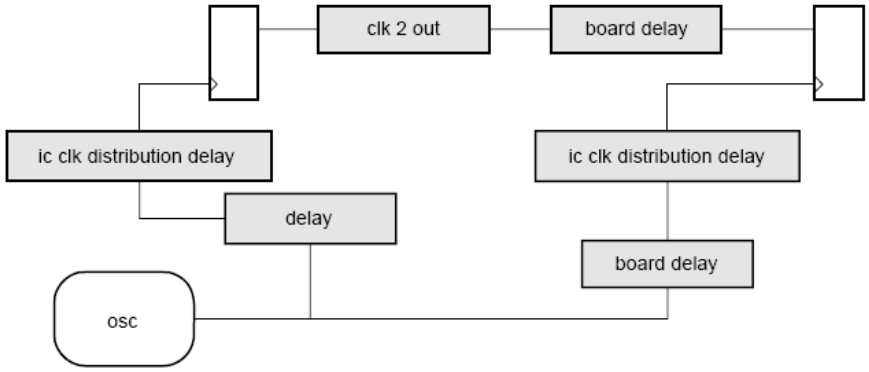


图1-8：**系统同步时序模型**

系统同步：两片IC之间进行通信时，使用一个共用时钟，用于数据发送和接收。

源同步

多年来，大多数的信号延迟都被忽略了，因为与有效时间相比，延迟时间很短。但是，随着速度的提高，管理延迟变得越来越困难，甚至最终变得不可能。改善问题的方法之一就是在发送数据的同时发送一个时钟副本。这种方法叫源同步（图1-9），它极大地简化时序参数。

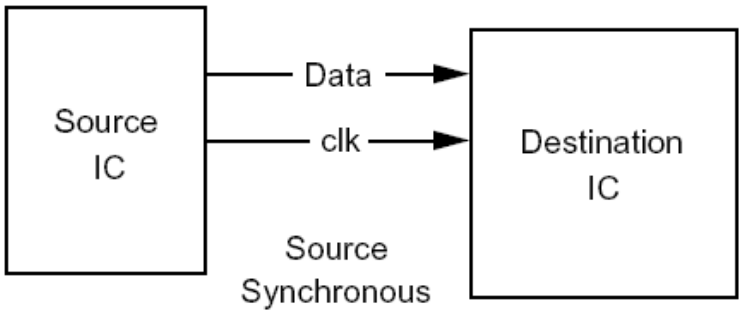


图1-9：**源同步结构图**

调节转发时钟的输出时间，使时钟在数据单元的中间位置发生翻转。因此，数据线和时钟线的长度需要互相匹配。但是，这种方法存在一些缺点。在目的芯片接收到的数据必须从接收时钟域转移到全局芯片时钟域中。

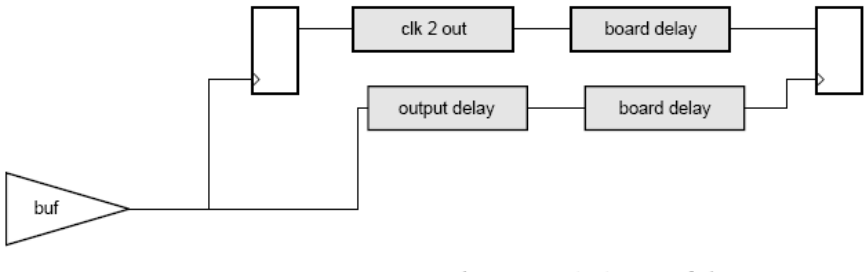


图1-10：**源同步时序模型**

源同步：两个IC间进行通信时，发送IC生成一个伴随发送数据的时钟信号。接收IC利用该转发时钟进行数据接收。

转发时钟：转发时钟（cf）或时钟转发是用于源同步的另一个技术术语。

源同步设计导致时钟域数量的剧增。对于具有有限时钟缓冲器的现场可编程门阵列（FPGA）和必须量身定制每个时钟树的专用集成电路（ASIC）等器件来说，这将带来时序约束和分析难题。对于采用大型并行总线的设计来说，该问题会进一步加重：由于电路板的设计限制，每条数据总线通常需要采用1个以上的转发时钟。因此，一条32位总线可能需要4个、甚至是8个转发时钟。

自同步

自同步模型如图1-11所示。这里，数据流包含数据和时钟。

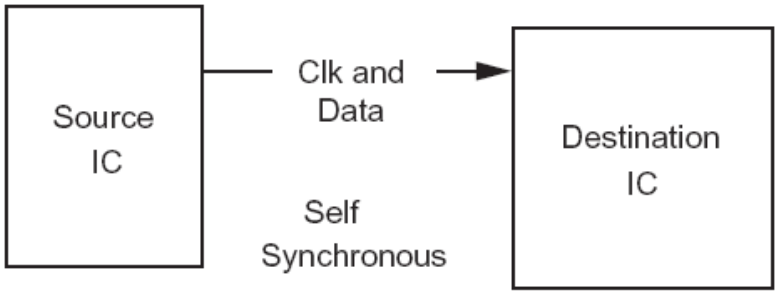


图1-11：**自同步结构图**

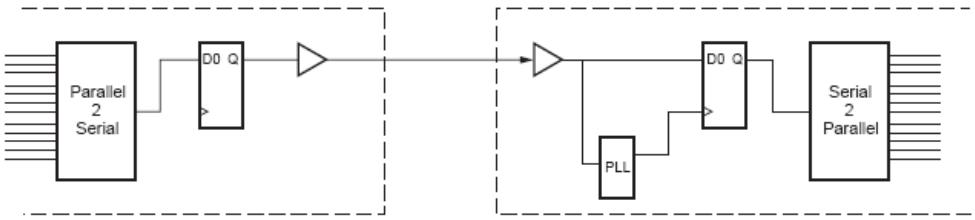


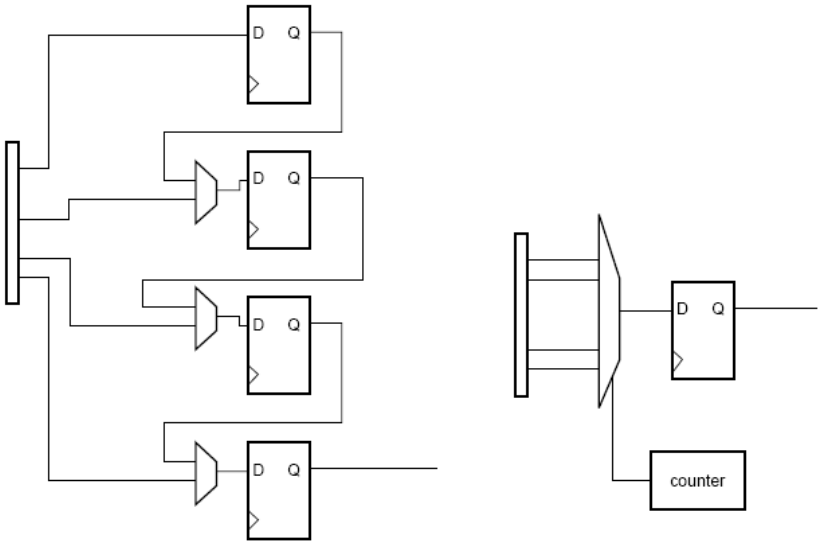
图1-12：**自同步时序模型**

自同步：两块芯片之间的通信，其中发送芯片产生的数据流同时包括数据和时钟信息。

自同步接口的三个主要模块分别是并串转换、串并转换和时钟数据恢复。

并串转换

有两种主要的并串转换方式——可装载移位寄存器和回转选择器。这些方法的简单逻辑图如图1-13所示。

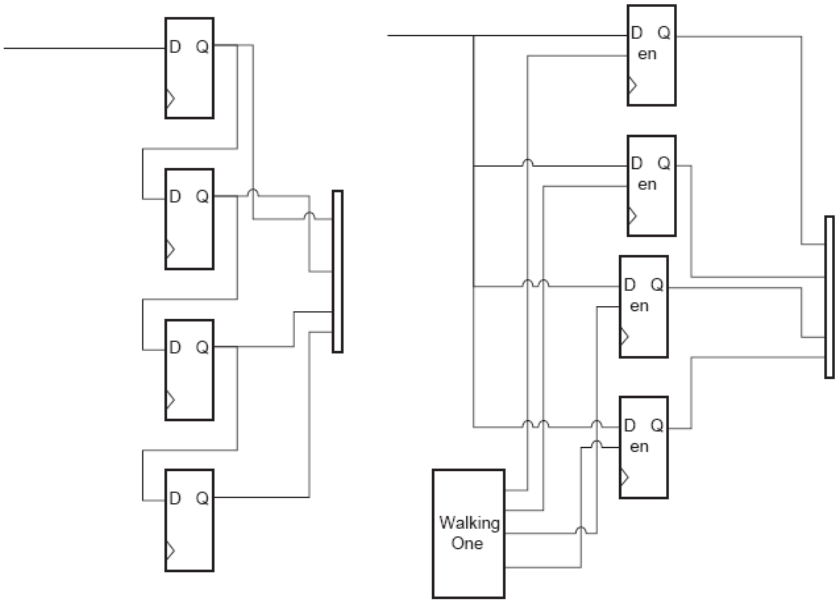


可装载移位寄存器　　回转选择器

图1-13：**并串转换步骤**

串并转换

串并转换与并串转换的步骤刚好相反，如图1-14所示。



移位寄存器　　回转使能

图1-14：**串并转换步骤**

时钟/数据恢复

时钟恢复过程（图1-15）无法产生一个共用时钟或者同数据一起发送时钟。作为替代，由锁相环（PLL）合成出一个与输入串行信号的时钟频率一致的时钟。

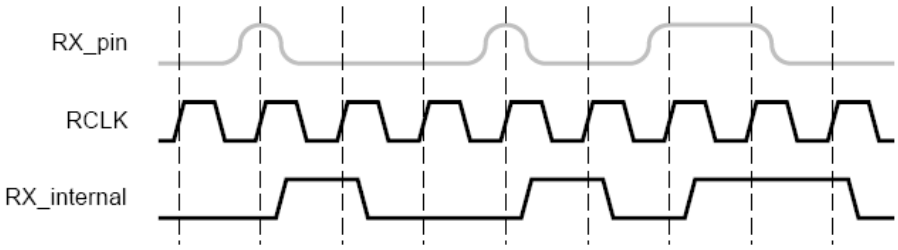


图1-15：**时钟/数据恢复波形**

锁相环：锁相环是这样一种电路，它能根据参考时钟和输入信号来产生锁定于输入信号的新时钟。

在并行数据传输中，经常使用额外的控制信号线为数据赋予不同的意义。例如数据使能信号，以及在同一总线上对数据和控制信号的多路选择。

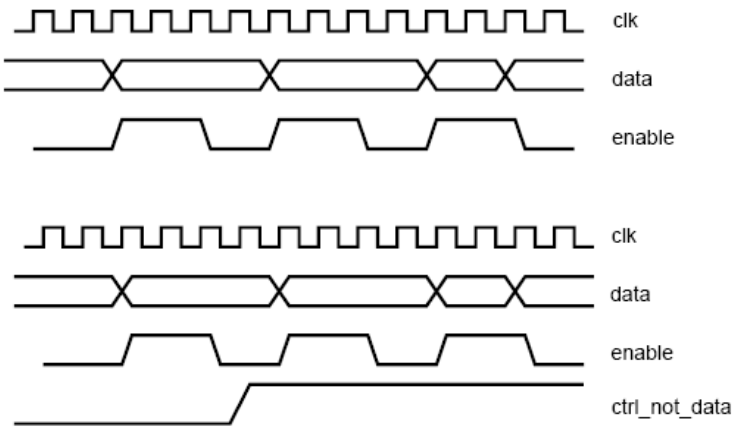


图1-16：**并行传输示例**

串行域中，标志或标记用于将数据与非数据（通常指空闲数据）区分开来。标志还可用来表示不同的信息类型，如数据信息和控制信息。

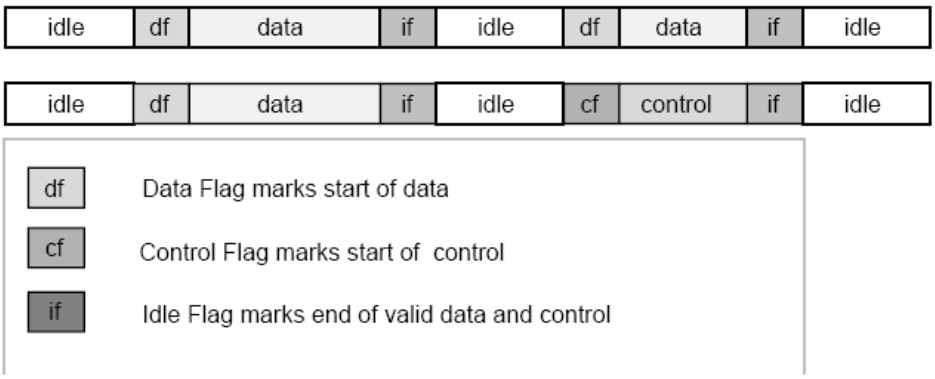


图1-17：**串行域传输示例**

I/O技术的不断改进

对带宽和速度的行业需求要求不断地改进I/O设计。并行和串行I/O在争夺芯片和器件通信领域主导权的同时，均受益于大幅提高速度的设计方法。采用数字设计方法（如差分和同步信号处理以及并行传输），可以保证家用和工业用I/O性能的不断改进。

第二章

为何需要千兆位串行I/O?

回顾千兆位串行I/O的设计优势

设计考虑

通常设计工程师都处于进退两难的境地。一方面，他希望能坚持使用已经过验证的、可靠的解决方案，因为这些方案的结果可靠并能够预见。另一方面，他也必须努力改进各项参数性能，如：数据流、引脚数、电磁干扰（EMI）、成本和背板效率等。那么，他会考虑使用千兆位串行输入/输出（I/O）吗？

千兆位串行I/O的优势

千兆位串行I/O的主要优势是什么？答案是：速度。在从片内/片外、板内/板外或盒内/盒外获取数据时，没有什么技术可以超过高速串行链路。这种技术的线速范围为1Gb/s~12Gb/s，有效负载范围为0.8Gb~10Gb，因此可以进行*大量*的数据传送。由于引脚数较少、没有大量的同时开关输出（SSO）问题、EMI较低且成本较低，所以高速串行就成为了理所当然的选择。当需要进行大量数据的快速传输时，使用千兆位级收发器（MGT）是个不错的方法。让我们首先分析一下千兆位串行I/O的优势。

MGT：千兆位级收发器——千兆位级串行器/解串器(SERDES)的别名。接收并行数据，并允许在串行链路上进行大带宽数据传输。

最大数据流

某些大型可编程逻辑器件具有20个或更多个10Gb串行收发器，可以实现总带宽为200Gb/s的输入和输出。不过那只是极端情况，我们来看一个应用实例，它向我们展示了串行I/O的速度是如何帮助系统架构师、电路板设计师和逻辑设计师的。

图2-1是高分辨率画质视频混频器的结构图。

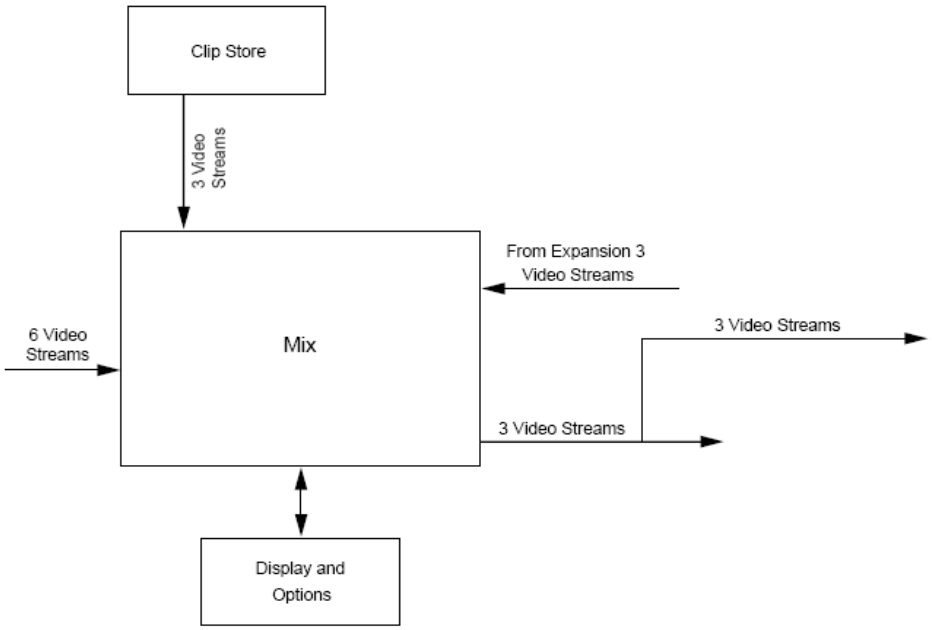


图2-1：**高分辨率视频混频器**

以基带信号形式或无压缩格式传输时，每一路高分辨率视频流的码率都需要1.5 Gb/s。创建这种系统的一种方式是采用分离的芯片对串行视频流进行解串行化和串行化，以及并行接口来处理扩展总线和素材库。另一种方式是使用逻辑电路内部的千兆位级收发器对串行数据流进行解码和编码。更快速的数据流扮演着扩展连接器和素材库接口的角色。

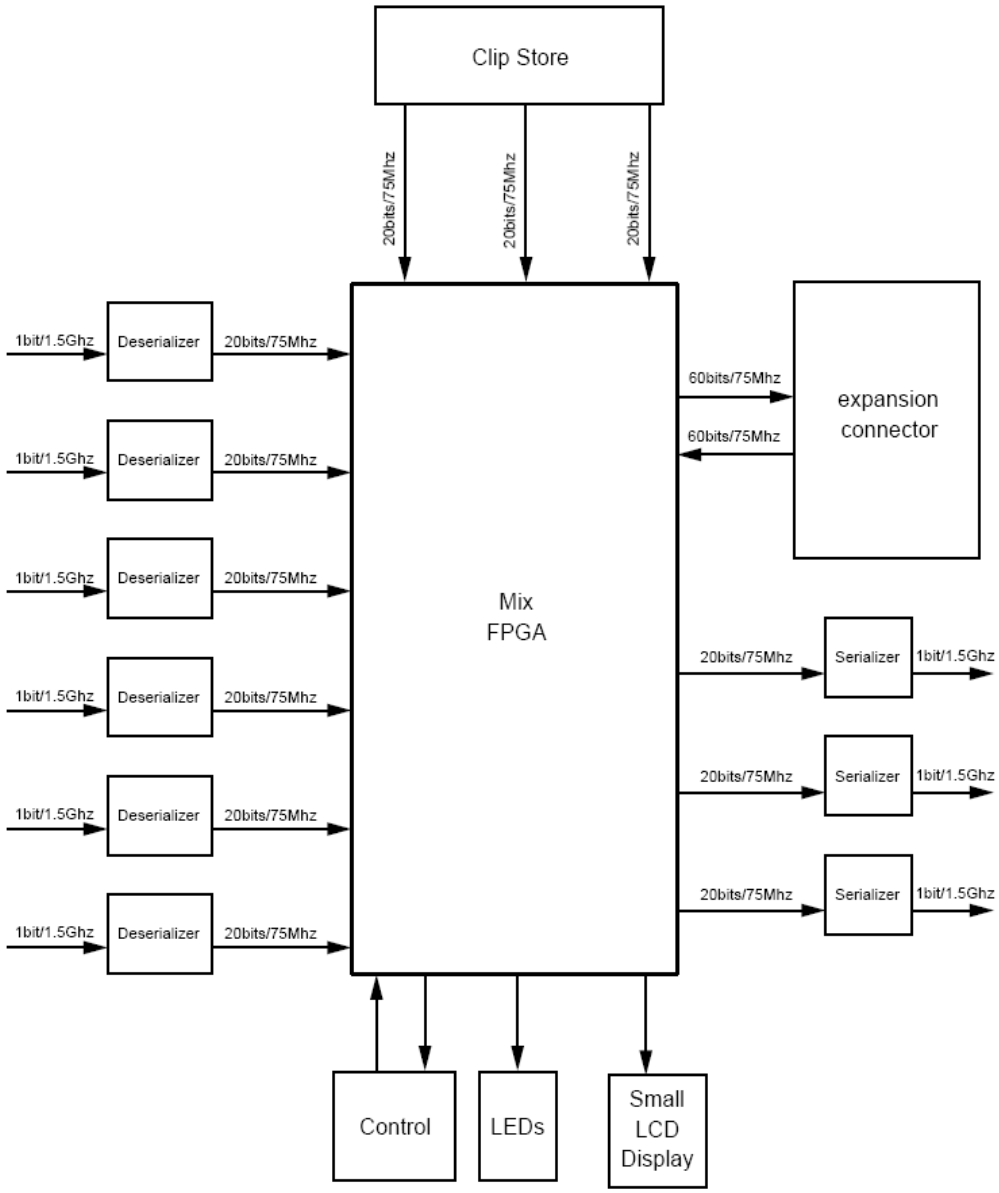


图2-2：采用解串行化和串行化的芯片（并行）

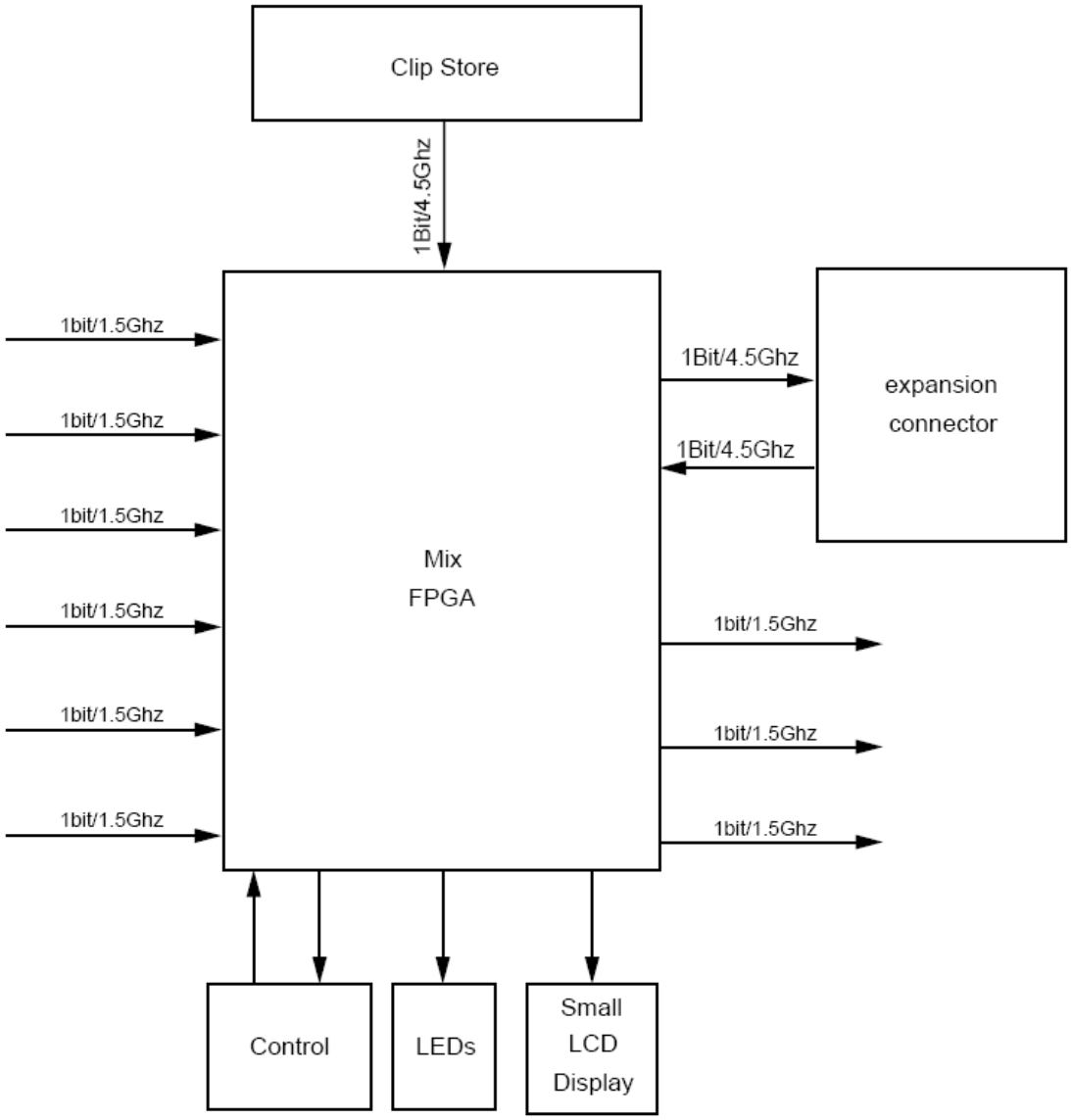


图2-3：采用千兆位收发器（串行）

引脚数

将大量数据传入或传出芯片或电路板时所遇到的第一个问题是引脚数。通常，输入和输出引脚数是有限的。虽然引脚数会随着时间而增加，但却总是不够用。

表2-1给出了2种方式下所需的引脚。

表2-1：**引脚数：串行和并行对比**

	方向	并行	串行
输入 1-6	IN	120	12
素材库	IN	60	2
扩展输入	IN	60	2
扩展输出	OUT	60	2
输出 1-3	OUT	60	6
控制/状态输入	IN	48	48
控制/状态输出	OUT	52	52
LED	OUT	12	12
LCD 驱动	OUT	48	48
总计		520	184

当然，还有一些我们没有说明的引脚问题。例如，某些MGT比一对传输较慢的管脚需要更多的电源和接地引脚。而且，并行接口可能需要特殊的参考引脚。但是，仅就比较应用而言，本例已足够了。

使用大量引脚时，电路板设计时间和成本会急剧增加。考虑连接器及电缆的选择和可用性时，连接器的引脚数也非常重要。全部使用现有的球形栅格阵列（BGA）封装可能会不太方便。

同步转换输出

采用单端并行总线时，设计者应考虑同步转换输出（SSO）。不过，其中的某些输出会在同一时间翻转。如果出现太多的同步转换，触地反弹会产生大量噪声。

设计者还可以在所有I/O上都使用差分信号处理技术，以此来消除SSO问题，但是，这样做就会使引脚数翻倍。如果数据流需求比较适中，设计者可以使用具有适当引脚数的并行接口。

EMI

经验表明：时钟越快，放射测试就越难进行，因此，千兆位设计看起来近乎不可能。但是，通常高速串行链路的辐射量比以较低速度工作的大型总线低。这是因为运行时的千兆位链路需要出色的信号完整性。正如一位专家所言：“辐射问题实际上就是信号完整性问题。”

成本

采用MGT通常会降低系统总成本。连接器采用较小、较经济的封装时，引脚数较少，电路板设计也更简单。视频混频器应用中，并行解决方案使用的IC（集成电路）数量比串行解决方案多9个。本例中，串行解决方案的成本比并行解决方案的成本低数百美元。

预设协议

采用MGT的另一个好处是可以使用预先定义好的协议和接口标准。从Aurora到XAUI，满足多种需求的设计已经存在。

缺点是什么？

在我们认为千兆位级串行I/O技术出色的近乎不真实之前，来看看它的弊端吧。设计中，首先我

们必须密切注意信号完整性问题。例如，有个供应商报告说，他们第一次试图将高速、千兆位级串行设计用于某种特定应用时，失败率为90%。为了提高成功率，我们可能需要进行模拟仿真，并采用更复杂的新型旁路电路。事实上，我们甚至需要对旁路电路进行仿真和建模。

而且，阻抗控制的 PC(印刷电路)板、高速连接器和电缆的费用较高。我们必须处理数字仿真中的复杂性和时基较小的问题。并且，在利用预设协议的时候，必须为集成过程计划时间，并且为协议的开销安排额外的逻辑电路或 CPU 时钟周期。

千兆位I/O用于何处？

起初，千兆位级串行器/解串器（SERDES）仅局限于用在电信行业和少数缝隙市场（如广播视频）。如今，MGT应用出现在电子行业的各个角落——军事、医疗、网络、视频、通信等等。MGT也可以用于背板或机箱之间的PCB上。对于电子行业的发展前景而言，MGT至关重要。下面是采用千兆位级SERDES的行业标准示例。

- 光纤通道（FC）
- PCI Express
- RapidIO串行
- 先进交换互连（Advanced Switching Interface）
- 串行ATA
- 1-Gb以太网
- 10-Gb以太网(XAUI)
- Infiniband 1X、4X、12X

芯片到芯片

SERDES最初用于盒间通信。但是，因为它能出色地处理同一块电路板上的芯片间通信，因而在市场上引起了轰动。先前，芯片间通信仅采用并行技术。用于串行化和解串行化的逻辑门数量远远超过了因引脚数目减少而节省的逻辑门数量。

但是，采用深亚微米结构，就可以在极小的芯片上获得数量惊人的逻辑门电路，从而使SERDES也能够以极低的芯片成本实现。除此之外，对I/O带宽日益增长的需求使得SERDES迅速成为进行芯片间大量数据传输的合理选择。使用SERDES进行芯片间通信具有如下好处：

- **引脚数：**更小、更经济的封装。
- **引脚数：**PCB层数减少。
- **更小的封装：**电路板更小、更经济；设计更紧凑。
- **SSO：**较少的引脚和差分信令消除了SSO问题。
- **功耗：**通常，高速串行链路的功耗要小于并行链路。这一特点在一些有源偏置/终端的高速并行标准，例如高速晶体管逻辑（HSTL）中尤为明显。
- **内含控制线路：**通常，并行接口除了数据线外，还需要一些控制线和使能线。大多数协议下使能和控制性能都可以嵌入到串行链路中。

板到板/背板

虽然并行结构曾经是最好的结构，但是它们已达到了极限。由于连接器上引脚数量的限制，大多数并行总线协议已发展到不可能增加数据位的地步了。时钟歪斜、数据歪斜、上升和下降时间以及抖动都限制了更进一步提高时钟频率。将数据速率加倍可能会有所帮助，但是它通常需要使用差分信令，而这将会引起引脚数量的急剧增加。而且，控制并行总线上的串扰问题也很困难。

新的串行背板与并行背板稍有不同。它们从一个节点到另一个节点间有专用的串行链路。图2-4对旧的并行总线和新的串行总线的基本结构进行了说明。

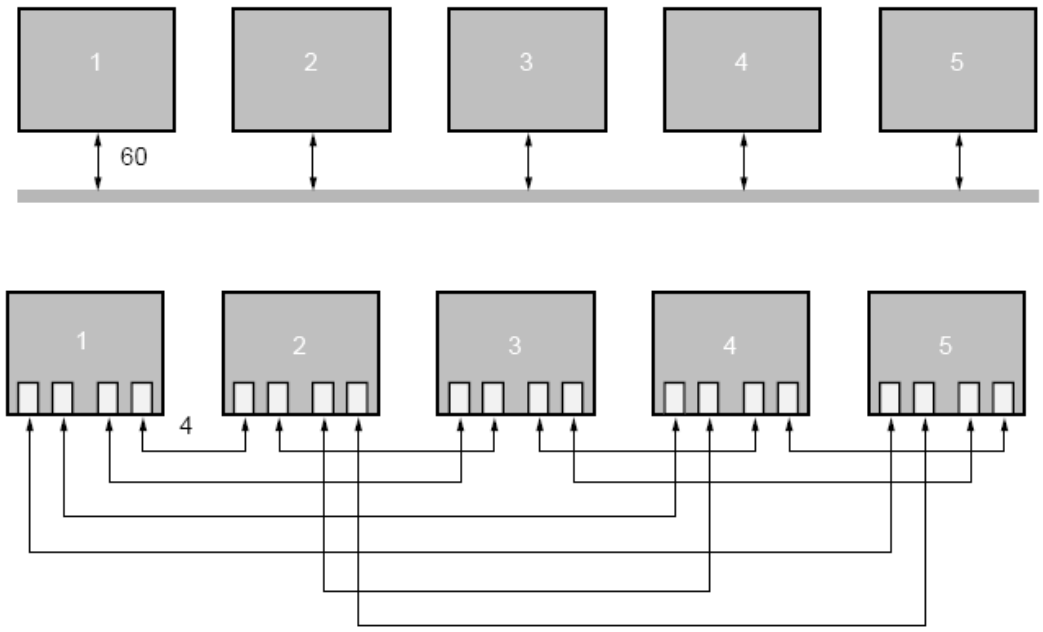


图2-4：旧的并行总线对比新的串行总线

串行总线结构可以提供很多功能。串行总线的引脚数是节点数的函数。对最常用的节点数而言，串行结构的引脚数比旧的并行结构少。

也许两者之间最重要的差异在于带宽接入方法。并行结构中，一个节点向一个或多个节点发送信号。但是，该节点在发送信号时，其它全部节点都是锁定的。传输带宽由全部节点共享。

串行总线中，每个节点到其它节点都有专用链路。因此，在一个节点通话时，另一个节点可以同时和一个或全部节点通话。事实上，全部节点都可以同时同其它全部节点通话。当然，节点必须按照先进先出（FIFO）的原则进行缓冲和存储，以便能够处理接收到的全部信息。

串行总线的结构优势在于：

- 带宽更大
- 引脚数更少
- 按节点到节点的方式检测带宽 (无需共享带宽)

- 解决方案内置到SERDES中
- 轻松实现协议支持

盒到盒

SERDES开始用于盒间连接时，很多设计者并未考虑使用千兆位级串行链路来实现盒间通信。常见的误解在于：不用光纤，盒间通信的速度会很慢。但是，有很多链路可以通过铜电缆系统进行盒间短距离通信。适于这些链路的标准之一就是Infiniband。Infiniband规范支持1、4或12个通道的串行数据通信（每个串行流的速度为2.5Gb/s）。该标准实现商业应用已有数年，电缆、连接器及协议等都得到了良好定义和测试。

图 2-5：**盒到盒的连线**

千兆位级设计的未来

乍一看，千兆位级通信似乎强加了一些不受欢迎的限制。串行设计者必须考虑信号完整性、较小的时基以及可能出现的对额外的门电路和CPU周期的需求。但是，在盒间以及芯片间通信中采用千兆位级技术的优势远远超过了那些可以察觉到的缺点。例如：高速、引脚数少、低EMI和低成本等，这些都使它成为了众多通信设计的理想之选，并保证了其在未来的通信应用中仍能继续得到广泛使用。

第三章

技术

实现千兆位串行I/O的相关技术

实际的串行I/O

在前面的章节中，我们分析了输入/输出(I/O)设计将面临的一些挑战。同时，我们也注意到串行I/O可以提供很多优点。但是，一名设计工程师怎样才能真正充分利用串行I/O的各种技术呢？在开始设计之前，我们需要知道什么对于实现串行I/O是有益的。我们需要研究一些基于串行设计的单元器件，从而了解一下是否有现成的工具可以帮助实现串行I/O。

千兆位串行的实现

在本章中，我们讨论了千兆位（multi-gigabit）链路设计的相关技术，同时也会介绍一下串行器/解串器（SERDES）及其基本构成单元，最后我们还讨论了所有这些速率是如何达到的（图3-1）。我们还将从逻辑和物理两个层面回顾串行数据流的格式。本章的内容是千兆位I/O设计的基础。

串行器/解串器

串行器/解串器和CDR的历史

串并转换和并串转换一开始就是I/O设计的一部分。时钟恢复，或“把时钟锁定在输入比特流上”也早就是I/O设计的一部分。那么为什么SERDES突然变得这么重要呢？

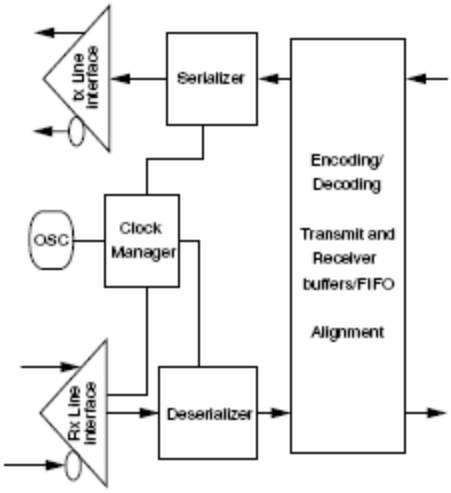


图3-1 SERDES结构框图

随着集成电路尺寸的变小，最大翻转速率（Fmax）的增大，I/O的带宽需求也日益增加。事实上，一些技术甚至允许I/O的频率比Fmax还要快。

Fmax：在给定技术或领域中，触发器的最大翻转速率

基本工作原理和总体框图

让我们来看看SERDES的基本构成模块（图3-2）。

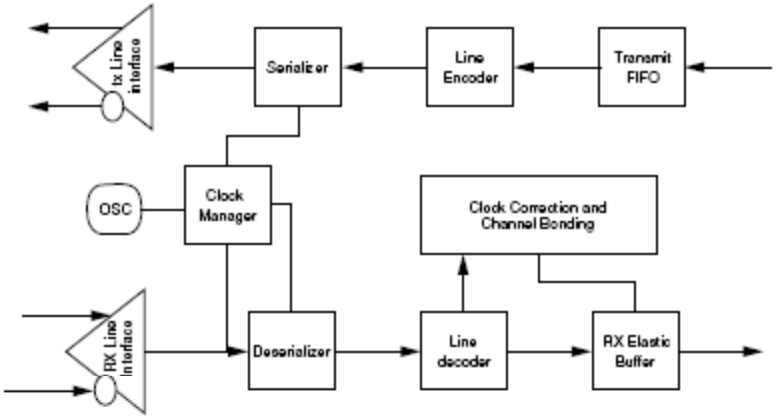


图 3-2 SERDES总体结构框图

- **串行器**：将速率为y的n位宽并行数据转变成速率为n*y的串行数据。
- **解串器**：将速率为n*y的串行数据转变成速率为y的n位宽并行数据。
- **Rx（接收）对齐**：将接收的数据对齐到合适的字边界。可以使用不同的方法，从自动检测和对齐特殊的预留比特序列（通常也称作comma字符），到用户控制的比特调整。
- **时钟管理器**：管理各种时钟操作，包括时钟倍频，时钟分频，时钟恢复。
- **发送FIFO（先进先出）**：在输入数据发送之前，暂时保存数据。
- **接收FIFO**：在接收数据被提取之前，暂时保存数据。在需要时钟修正的系统中，接收FIFO是必须的。
- **接收线路接口**：模拟接收电路，包括差分接收器，还可能包括有源或者无源均衡电路。
- **发送线路接口**：模拟发送电路，可以支持多种驱动负荷。通常还带有转换的预加重部分。
- **线路编码器**：将数据编码成适应不同线路的格式。编码器通常会消除长的无转变位的序列，同时还可以平衡数据中0、1的出现次数。（这是一个可选模块，某些SERDES可能没有。）
- **线路译码器**：将线路上的编码数据分解成原始数据。（这是一个可选模块，编码可能在SERDES外完成。）
- **时钟修正和通道绑定**：修正发送时钟和接收时钟之间的偏差，同时也可实现多通道间的歪斜修正。（通道绑定是可选的，并不一定包含在SERDES中。）

其他可能包括的功能模块有：循环冗余检测（CRC）码生成器、CRC检测器、多种编码和解码（4b/5b,8b/10b,64b/66b）、可调的扰码器、各种对齐和菊花链选项，以及可配置的时钟前端和后端。

SERDES的常用功能还包括不同等级的自环。市场上有多种商用的SERDES。图 3-3和图 3-4给出了几种SERDES的结构框图。

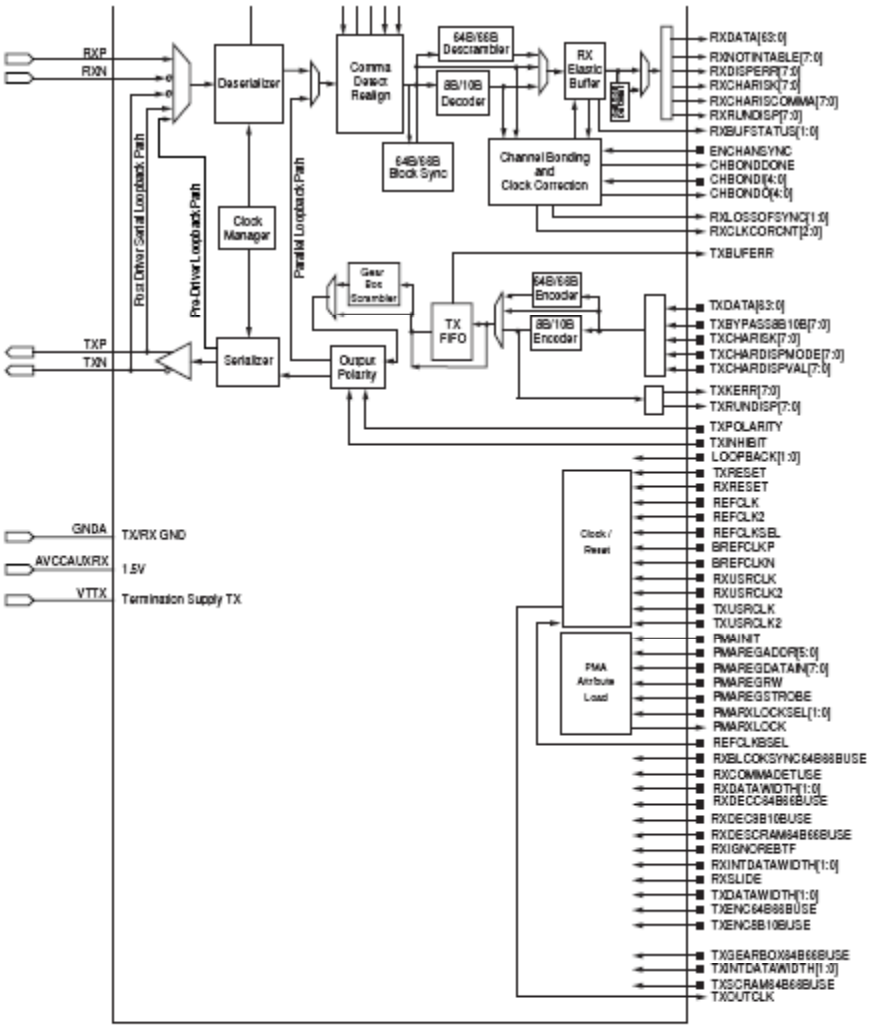


图3-3 :Virtex™-Ⅱ Pro X RocketIO™结构框图

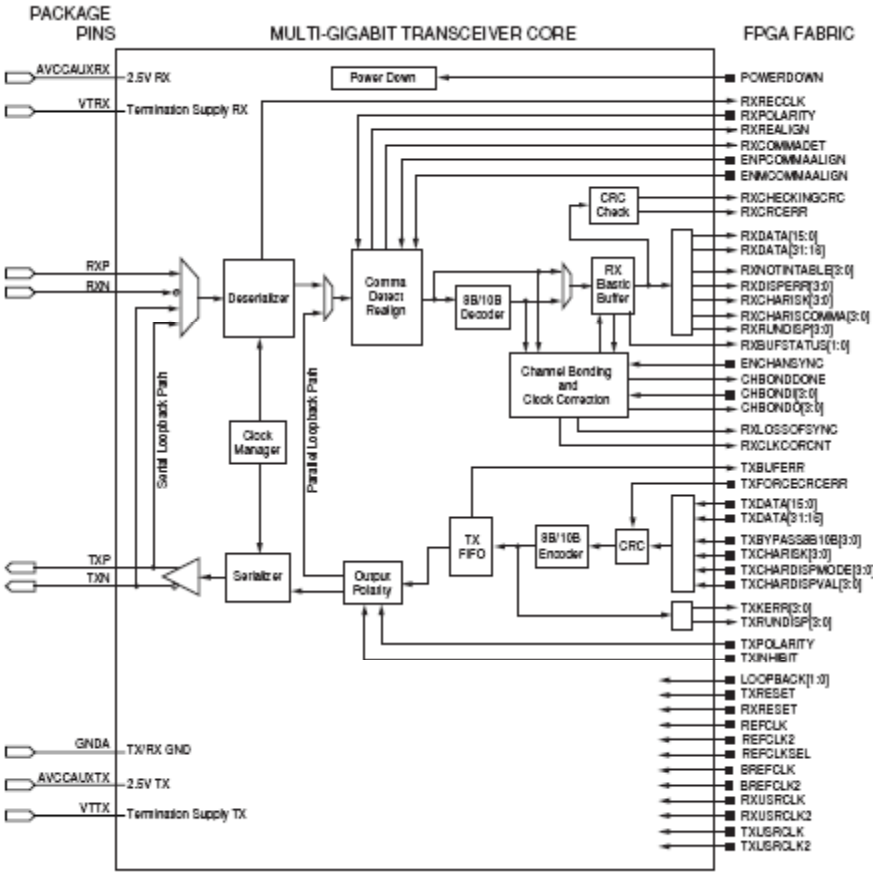


图3-4 :Virtex™-Ⅱ Pro X RocketIO™结构框图

为何SERDES速度如此之快？

人们多少会觉得千兆位SERDES是不可思议：它们就像魔法一样。千兆位SERDES可以工作在3、5甚至10Gb/s。它怎么可能达到这么高的速度呢？千兆位SERDES可以使用多种技术来实现这种工作速度。

这些技术中多数都采用了多重相位技术（图3-5和图3-6）。通过分析多重相位数据提取电路，我们可以知道多重相位有何帮助。如果输入的串行数据流比特率为x，那么我们可以使用多重相位以x/4的低速时钟来重新组织数据流。输入的数据流直接连接到4个触发器，每一个触发器运行在时钟的不同相位上（0、90、180 以及270）。

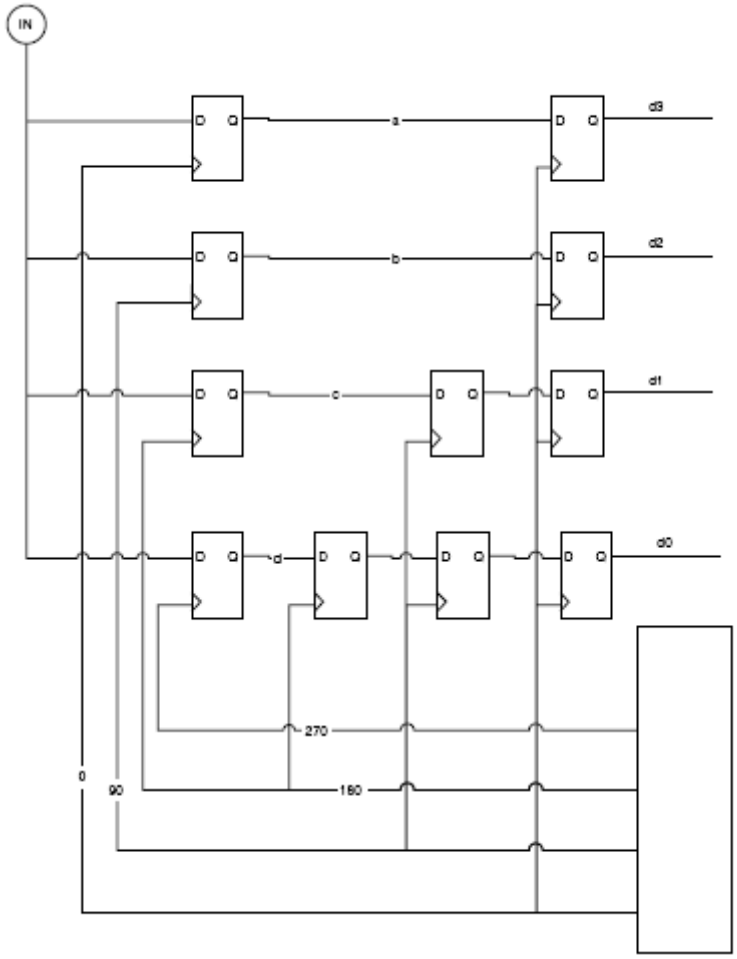


图 3-5 多重相位数据提取电路

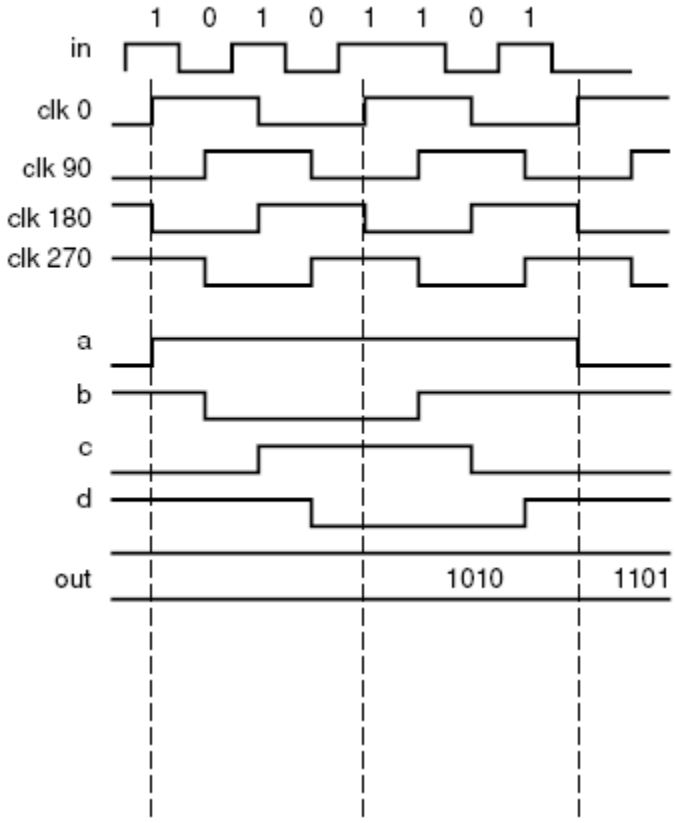


图3-6 多重相位提取电路的示例波形

每个触发器的输出连接到时钟相位小90度的触发器，直到到达时钟相位为0 的触发器。这样，输入数据流就被分解成了1/4输入速率，4bit宽度的并行数据流。

在上述的示例电路中，相位等差排列，时钟频率严格等于输入数据流速率的1/4。怎样才能实现呢？我们必须和输入的数据流保持锁定。我们可以使用典型的锁相环来实现这一点，但是锁相环需要一个全速率的时钟，这是很难满足的。锁相环是高速SERDES设计中最重大的改进之一，它主要用于时钟和数据恢复。一般的锁相环需要有运行在数据速率上的时钟，不过可以通过多种技术来避免这种要求，包括分数鉴相器、多重相位锁相环、并行采样以及过采样数据恢复。

线路编码机制

线路编码机制将输入的原始数据转变成接收器可以接收的格式。同时，线路编码机制还必须保证有足够的切换提供给时钟恢复电路。编码器还提供一种将数据对齐到字的方法，同时线路可以保持良好的直流平衡。

线路编码机制也可选择用来实现时钟修正、块同步、通道绑定和将带宽划分到子通道。

线路编码机制主要有两种－数值查找机制和自修改数据流或扰码器机制。

8b/10b编码/解码

8b/10b编码机制是由IBM开发的，已经被广泛采用。8b/10b编码机制是Infiniband，千兆位以太网，FiberChannel以及XAUI 10G以太网接口采用的编码机制。它是一种数值查找类型的编码机制，可将8位的字转化为10位符号。这些符号可以保证有足够的跳变用于时钟恢复。表3-1给出了几个例子，例子中的8位数值会导致很长时间不出现切换。8b/10b将12个特殊字符编码成12个控制字符，通常也称作“K”字符。我们将对K字符详细了解，但首先让我们看一下8b/10b是如何保证了良好的直流平衡性能。

表3-1 8位数值

8位数值	10位符号
00000000	1001110100
00000001	0111010100

运行不一致性（Running Disparity）

8b/10b中的直流平衡是通过一种称作“运行不一致性”的方法来实现的。实现直流平衡的最简单办法是：只使用有相同个数0和1的符号，但是这将限制符号的数量。

而8b/10b则为各个数值分配了两个不同的符号。在大多数情况下，其中一个符号有6个0和4个1，另一个符号则有4个0和6个1。编码器检测0和1的数量，并根据需求选择下一个符号，以保证线路的直流平衡。这两个符号通常也称作+和-符号。表3-2给出了一些符号示例。

表3-2　8b/10b 符号示例

名称	16进制	8位	负运行不一致性	正运行不一致性
D10.7	EA	11101010	0101011110	0101010001
D31.7	FF	11111111	1010110001	0101001110
D4.5	A4	10100100	1101011010	0010101010
D0.0	00	00000000	1001110100	0110001011
D23.0	17	00010111	1110100100	0001011011

“运行不一致性”的另一个优点是接收器可以通过监控运行不一致性，并检测输入数据中的错误，因为此时数据违反了运行不一致性规则。

控制字符

表 3-3列出了12个称作控制字符或K字符的特殊符号

表 3-3 有效的控制K字符

名称	16进制	8位	负运行不一致性	正运行不一致性
K28.0	1C	00011100	0011110100	1100001011
K28.1	3C	00111100	0011111001	1100000110
K28.2	5C	01011100	0011110101	1100001010
K28.3	7C	01111100	0011110011	1100001100
K28.4	9C	10011100	0011110010	1100001101
K28.5	BC	10111100	0011111010	1100000101
K28.6	DC	11011100	0011110110	1100001001
K28.7	FC	11111100	0011111000	1100000111
K23.7	F7	11110111	1110101000	0001010111
K27.7	FB	11111011	1101101000	0010010111
K29.7	FD	11111101	1011101000	0100010111
K30.7	FE	11111110	0111101000	1000010111

这些控制字符用于对齐，控制以及将带宽划分成为子通道。

Comma字符检测

数据的对齐是解串器的一项重要功能，图3-7给出了串行流中的有效8b/10b数据示例。

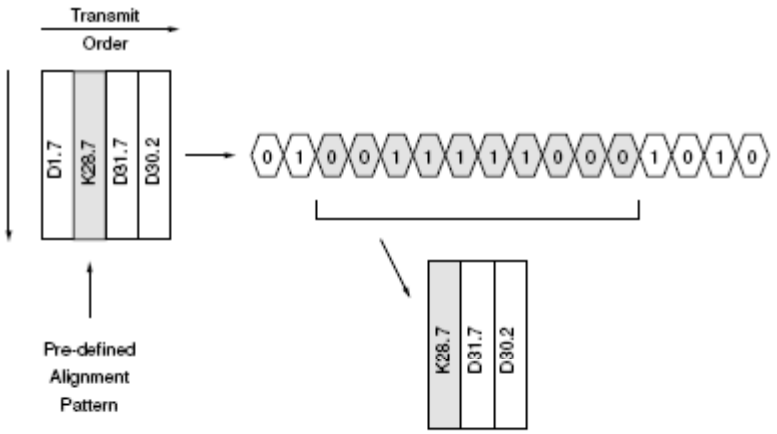


图3-7 串行数据流中的有效8b/10b符号

我们如何知道符号的边界呢？符号被comma字符隔开。此处comma字符包括一个或两个符号，用于指定comma或对齐序列。收发器中这个序列通常是可以设置的，不过有时序列是预先定义好的。

comma字符：用于表示对齐序列的一个或两个符号

接收器在输入数据流中扫描搜寻特定的比特序列。如果找到序列，解串器调整字符边界以匹配检测到的comma字符序列。扫描是连续进行的。一旦对齐确定，所有后续的comma字符均会发现对齐已经确定。当然，在任意的序列组合里comma字符序列必须是唯一的。

举个例子，如果我们使用符号c作为comma字符，则我们必须确保任意有序符号集xy中都不包含比特序列c。使用预定义协议时是不出现这个问题的，因为comma字符都是已经定义好的。

常用的K字符是全部控制字符中的一个或多个特定子集。这些子集中包含K28.1 K28.5 K28.7，这些字符的头7位都是1100000。这种比特序列模式只可以在这些控制字符中出现。其他任意的字符序列或者其他K字符都不包含这一比特序列。因此，这些控制字符是非常理想的对齐序列。在使用自定义协议的情况下，最安全且最常用的解决方案是从比较著名的协议中“借”一个序列。千兆位以太网使用K28.5作为comma字符。鉴于这个原因，尽管在技术上还有其他的选择，这个字符还是经常被当作comma字符。

控制字符的命名方式源于编码器和解码器构建方式，例如：D0.3和K28.5，图3-8 描绘了这种方法。

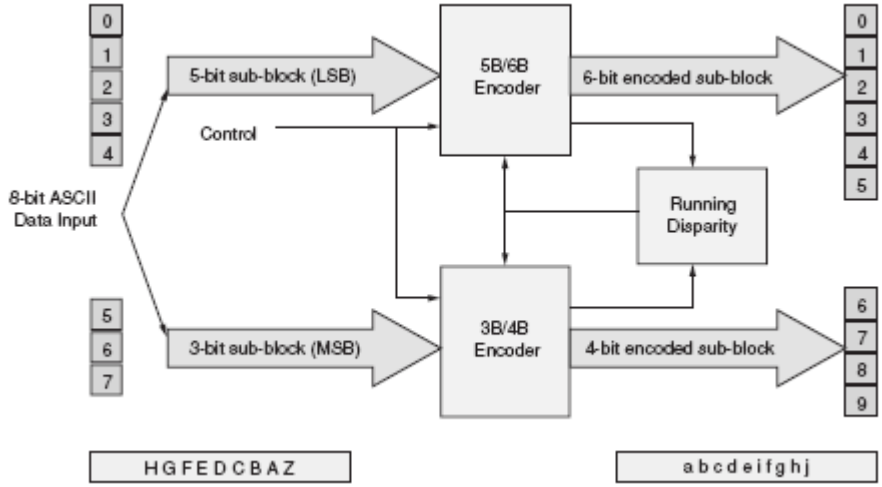


图3-8 编码器/解码器结构框图

输入的8位比特被分割到5位和3位的总线上，这就是其名字的来源。举例说明，Dx.y表示最低5位的数值对应十进制值x，而最高3位的数值对应十进制值y的输入字节。

K表示控制字符。3位转化为4位，5位转化为6位。另一种命名方式中， 8位比特对应于HGFEDCBA，而10位比特对应于abcdeifghj。

开销是8b/10b机制的一个缺陷。为了获得2.5Gbit的带宽，需要3.125Gb/s的线路速率。使用扰码技术可以很容易地解决时钟发送和直流偏置问题，并且不需要额外的带宽。

扰码

扰码是一种将数据重新排列或者进行编码以使其随机化的方法，但是必须能够解扰恢复。我们希望打乱长的连0和长的连1序列，将数据随机化。显然，我们希望解扰器在解扰时不需要额外的对齐信息。具有这种特性的码称作自同步码。

扰码：一种将数据重新排列或者编码以使其随机化的方法，但必须能解扰恢复。

一个简单的扰码器包含一组排列好的触发器，用于移位数据流。大部分的触发器只需要简单地输出下一个比特即可，但是某些触发器需要和数据流中的历史比特相与或者相或。图3-9 描述了此概念。

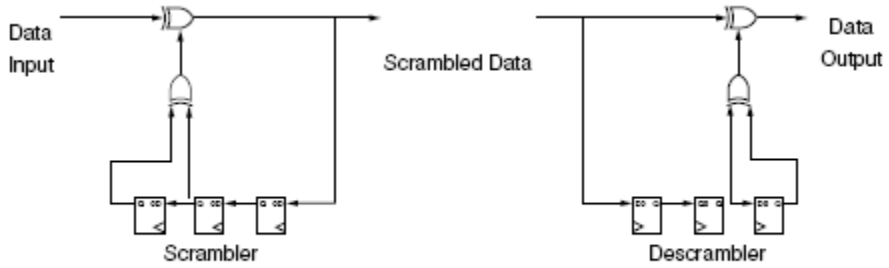


图3-9 基本扰码电路

扰码方法通常是伴随着多项式出现的，因为扰码的数学原理中使用了多项式。多项式的选择通常是基于扰码的特性，包括生成数据的随机度，以及打乱长的连0、连1的能力。扰码必须避免生成长的连0或连1序列。

我们希望能够加快触发器的时钟速率。但是想要达到诸如10Gb/s的高速率不是能轻易办到的。但是，还是有方法可以将任意形式的串行数据并行化为y位宽度的并行数据，从而加快进程。如图3-10所示。

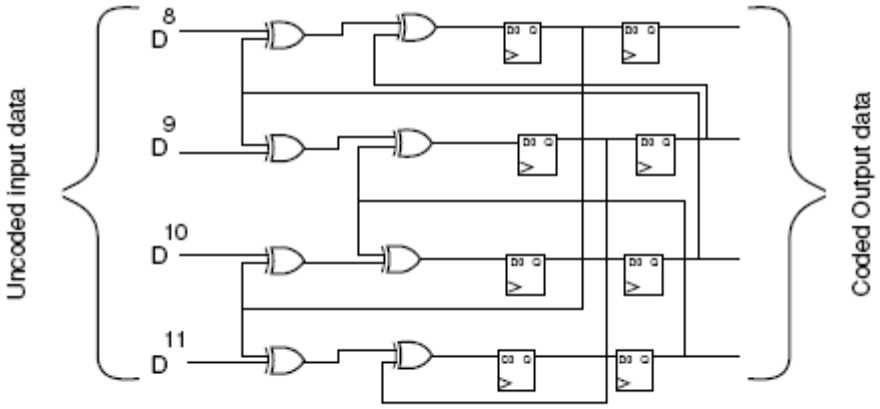


图3-10 并行扰码电路

扰码器消除了长连0、连1序列以及其他会对接收器接收能力有负面影响的序列。但是线路编码机制（例如：8b/10b）的其他功能不是由扰码器提供的，包括：

- 字对齐
- 时钟修正机制
- 通道绑定机制
- 子通道生成

某些情况下后三种功能可能是不需要的，而字对齐通常是必要的。如果线路编码使用了扰码，那么字对齐也必须采用相应的方法。例如，我们可以将某些数值排除在容许的数据（或者有效载荷）数值之外。此后，我们可以使用这些禁用的值来创建一个比特流，这个比特流不会在序列的数据部分中出现（图3-11）。

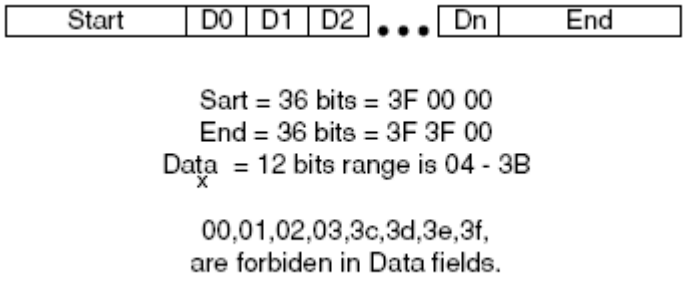


图 3-11 为扰码设计的数据帧格式

通常，因为存在不允许的数值，所以需要设计数据流中不能出现连0或连1的长度。长的连0、连1会被扰码器打乱，并在解扰时进行恢复。接收数据流的解扰逻辑在数据流中搜寻这些符号并对齐数据。类似的技术还可用于建立其他特性。

4b/5b 64b/66b

4b/5b和8b/10b是类似的，但是要简单些。顾名思义，这种机制将4个比特编码成5个比特。4b/5b的编码器和解码器要比8b/10b简单一些。但是4b/5b的控制字符要少一些，并且不能处理直流平衡和不一致性问题。由于编码开销相同但是功能却比较少，4b/5b编码机制并不经常使用。它的最大优势是设计的尺寸，不过随着逻辑门价格的降低这个优势也不再明显。4b/5b仍用在各种标准中，包括低速率版本的FiberChannel、音频工程协会-10（AES-10）以及多通道数字音频接口（MADI，一种数字音频复接标准）。

还有一种新的编码方式称作64b/66b 。我们可以认为64b/66b是8b/10b的简化版本，它具有更低的编码开销，但是实现细节相当不同。

在现有的技术用户需求下，人们开发出了64b/66b机制。10G以太网协会要求实现基于以太网的10Gb/s通信。他们可以通过使用4条有效载荷速率为2.5Gb/s、线路速率为3.125Gb/s的链路来实现，但此时SERDES已经可以在单个链路上实现10Gb的解决方案。此时新型SERDES的运行速率已经可以略高于10Gb/s了，但还不能达到12.5Gb/s以支持8b/10b的开销。

激光驱动二极管是另一个问题。电信标准同步光网络（SONET）使用的激光器性能刚好超过10Gb。而高速激光器的价格要昂贵的多。千兆位以太网协会要么选择放弃，要么开发出一种低开销的新方法来取代8b/10b。所以他们选择了64b/66b。

64b/66b：一种为10G以太网开发的新型线路编码机制，它使用了带有非扰码同步字符和控制字符的扰码方式。

不同于8b/10b的查找表方式，64b/66b使用了带有非扰码同步字符和控制字符的扰码方式。图3-12示意了64b/66b编码机制。

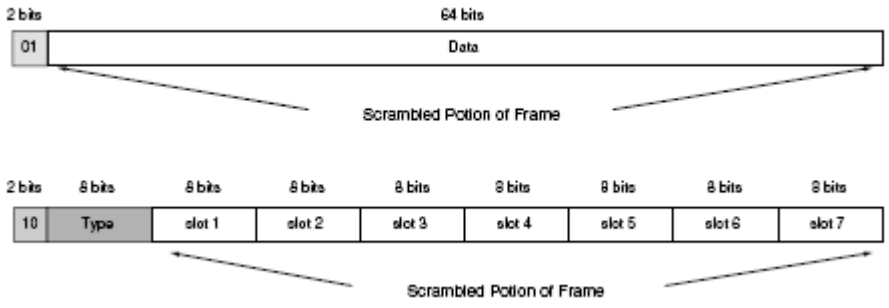


图 3-12 64b/66b 框图

主帧的类型主要有两种。简单的主帧包括两位同步比特01以及 64位的数据。数据经过扰码处理，但是同步比特则不进行扰码处理。另一种主帧即可以是数据也可以是控制信息。控制帧的前两位是同步比特10。类型域的8比特定义其余56位有效载荷的格式。举例说明，如果类型是十六进制0xcc，则该帧包含4个字节的数据和3个字节的控制信息（图3-13）。

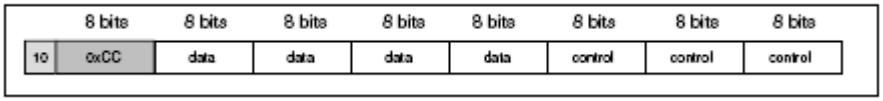


图 3-13 **0xcc 类型示例**

此处还存在一个和帧相关的0位宽符号（图3-14）。

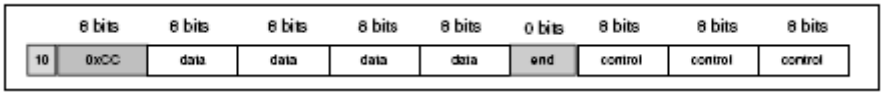


图3-14 **0xcc Type及符号示例**

怎样产生一个0位宽的符号？这个符号实际上并不是真的0比特，而是映射在有效载荷中的类型字节的一部分。类型域中有8个比特，允许存在256种不同类型的有效载荷。大多数64b/66b系统定义了约15种不同的类型。这15种类型的简单定义是：x位数据伴随y位控制比特。也可以反过来是x位控制比特伴随y位数据。通常可以使用某些0位宽符号来定义这些数据/控制符号的位置。

常用的0位宽符号是t（结尾）和s（开始）。图3-15给出了一种64b/66b机制的全部控制块格式列表。

Input Data	Sync	Block Payload							
Bit Position / Data Block Format	01	2							65
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7	01	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
Control Block Formats		Block Type Field							
C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7	10	0x1e	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
C_0 C_1 C_2 C_3 O_4 D_5 D_6 D_7	10	0x2d	C_0	C_1	C_2	C_3	O_4	D_5	D_6	D_7
C_0 C_1 C_2 C_3 S_4 D_5 D_6 D_7	10	0x33	C_0	C_1	C_2	C_3		D_5	D_6	D_7
O_0 D_1 D_2 D_3 S_4 D_5 D_6 D_7	10	0x66	D_1	D_2	D_3	O_0		D_5	D_6	D_7
O_0 D_1 D_2 D_3 O_4 D_5 D_6 D_7	10	0x55	D_1	D_2	D_3	O_0	O_4	D_5	D_6	D_7
S_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7	10	0x78	D_1	D_2	D_3	D_4		D_5	D_6	D_7
O_0 D_1 D_2 D_3 C_4 C_5 C_6 C_7	10	0x4b	D_1	D_2	D_3	O_0	C_4	C_5	C_6	C_7
T_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7	10	0x87		C_1	C_2	C_3	C_4	C_5	C_6	C_7
D_0 T_1 C_2 C_3 C_4 C_5 C_6 C_7	10	0x99	D_0		C_2	C_3	C_4	C_5	C_6	C_7
D_0 D_1 T_2 C_3 C_4 C_5 C_6 C_7	10	0xaa	D_0	D_1		C_3	C_4	C_5	C_6	C_7
D_0 D_1 D_2 T_3 C_4 C_5 C_6 C_7	10	0xb4	D_0	D_1	D_2		C_4	C_5	C_6	C_7
D_0 D_1 D_2 D_3 T_4 C_5 C_6 C_7	10	0xcc	D_0	D_1	D_2	D_3		C_5	C_6	C_7
D_0 D_1 D_2 D_3 D_4 T_5 C_6 C_7	10	0xd2	D_0	D_1	D_2	D_3	D_4		C_6	C_7
D_0 D_1 D_2 D_3 D_4 D_5 T_6 C_7	10	0xe1	D_0	D_1	D_2	D_3	D_4	D_5		C_7
D_0 D_1 D_2 D_3 D_4 D_5 D_6 T_7	10	0xff	D_0	D_1	D_2	D_3	D_4	D_5	D_6	

图3-15 64b/66b的控制模块格式列表

仔细观察此表就会发现0符号。这个符号用于定义有序集。这样就可以允许使用8b/10b编码机制的协议转而使用64b/66b编码机制。

现在，让我们来实际讨论一种线路编码机制。我们将仔细研究编码机制的各种主要功能，并了解它们是如何实现的。

充足的跳变

有效载荷段的扰码可以提供足够的转变用于时钟恢复。细心选择的扰码器同时还可以解决直流偏置问题。64b/66b中使用的扰码器是$X_{58} + X_{19} + 1$。

对齐

64b/66b的对齐程序和其他方法不同。图3-16中显示了它是如何工作的。

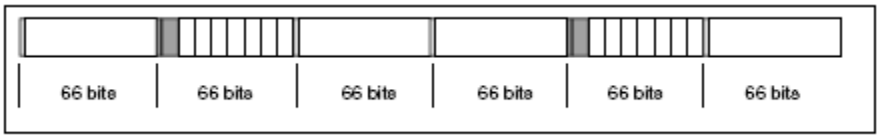


图3-16 64b/66b对齐程序

每66位中都会有01或10的同步比特。在比特流的其他地方也会出现这样的比特组合。对齐程序首先随机选择一个起点。它首先搜寻有效的同步（01或者10组合），如果没有找到，则移动一位然后重新检测。一旦找到01或者10组合，则检查后续的66个比特。如果后续比特中包含一个有效同步符号，则计数器增1，然后继续检测后面的66个比特。如果在一行中能够连续检测到足够多的同步符号，而且没有发生错误，则确定对齐。如果检测过程中出现任何错误，则计数器清零。

一旦对齐位置锁定，错误的同步符号将被认为是错误。如果在一段时间内发生了足够多的错误，则对齐位置将重新估计。乍一看，这种方法可以在最大有效同步尝试内（+66或者更少）完成锁定。但是因为数据窗中出现类似的01或10序列概率很大，所以排除这些错误路径会花费很多时间。

为了加快锁定时间，建议使用其他的可选协议。这些协议中用特殊的训练序列或锁定序列来替换数据，从而可以实现简单的定位。

时钟修正

时钟修正可以在字节、多字节边界或66位码字内进行。时钟修正符号可以是一种特殊的码字。码字的有效载荷内不包含任何有用信息，在必要的时候可以重复或者删除。除此之外，一个字节宽度的时钟修正符号可以被定义作为任意的未用值。当然，如果我们使用的SERDES只支持上述方法中的一种，我们就使用这种特定方法。单字节宽度的方法是最常用的方法，因为它可以有较小的接收FIFO缓冲器而且能够更好地兼容现存的协议。

通道对齐

通道对齐的操作很类似于时钟对齐，既可以是一个特殊符号，也可以是包含在控制信息中的特定序列。

子通道

子通道的处理也和时钟对齐类似，既可以是一个特殊符号，也可以是包含在控制信息中的特定序列。

4b/5b 64b/66b 的权衡折衷

通过开销编码方法等功能的使用，10G以太网可以采用现存的SONET类型的激光二极管。除了使用相同的激光二极管，这种方法和SONET还有很多相似之处；SONET的定位原理中有很多和这种方法是相同的，但是要比64b/66b复杂的多。

低开销的代价是更长的对齐时间、出现轻微直流偏置的可能性和更加复杂的编码器和解码器。很多复杂处理使得64b/66b的电路比它们的近亲8b/10b要复杂的多，例如启用或者关闭有效载荷的扰码器就是很复杂的过程。解码器启用和使用也更加复杂了。

包简介

部分设计师觉得在局域网上用包来传输所有数据完全是一种浪费。我们首先从包的定义来开始研究这个问题。

包：一种确切定义的字节集合，包括头部、数据和尾部。

注意，定义中没有包括源地址和目的地址、CRC校验码、最小长度、或者开放系统互连协议层。包只不过是一个定义了起点和终点的数据结构。局域网的包通常有很多特性，但是其他用途的包则通常要简单得多。

包用于各种场合下的信息传递－例如汽车动力布线、手机以及家庭娱乐中心等等。但是，包和千兆位串行链路有什么关系呢？

通过千兆位串行链路传输的数据多数都是嵌入在某种类型的包中的。SERDES自然需要一种将输入的数据流对齐成字的方法。如果系统需要时钟修正，还必须发送特殊的比特序列或者comma字符。comma字符是指示帧的开始和结束的天然标识。如果需要时钟修正，时钟修正序列常常是比较理想的字符。加入有序集合用于指示包的开始、结束以及包的特殊类型之后，我们就有了一个简单而又强大的传输通道。

idle符号或序列是包的概念的另一要点。如果没有信息需要发送，则发送idle符号。数据的连续传输保证链路能够维持对齐，

同时PLL可以保持恢复时钟锁定。图 3-17给出了一些不同标准的包格式。

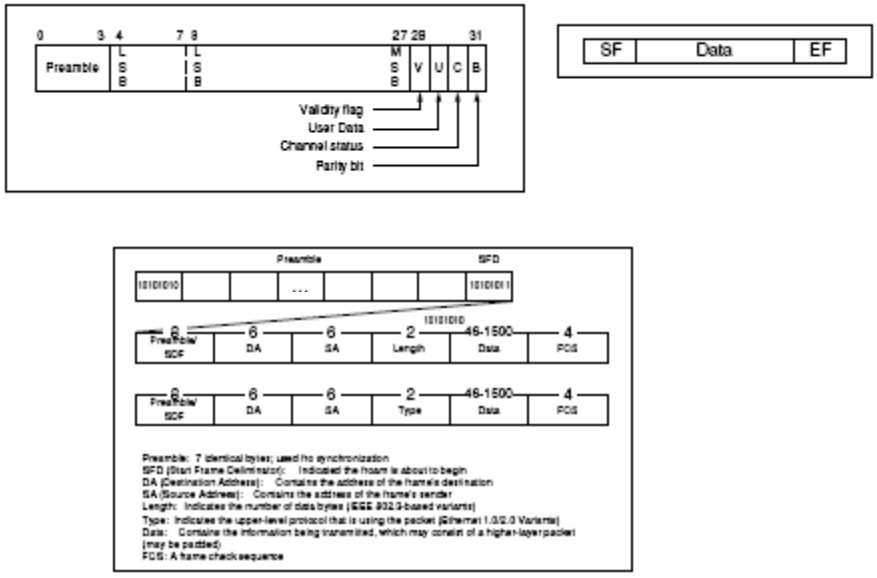


图 3-17 包格式框图

参考时钟的要求

千兆位级收发器的输入时钟、或是参考时钟的规格定义是非常严格的。其中包含非常严格的频率要求，通常用每百万次容许频率错误的单位PPM来定义。抖动要求也是十分严格的，通常用时间（皮秒）或者时间间隔（UI）定义。

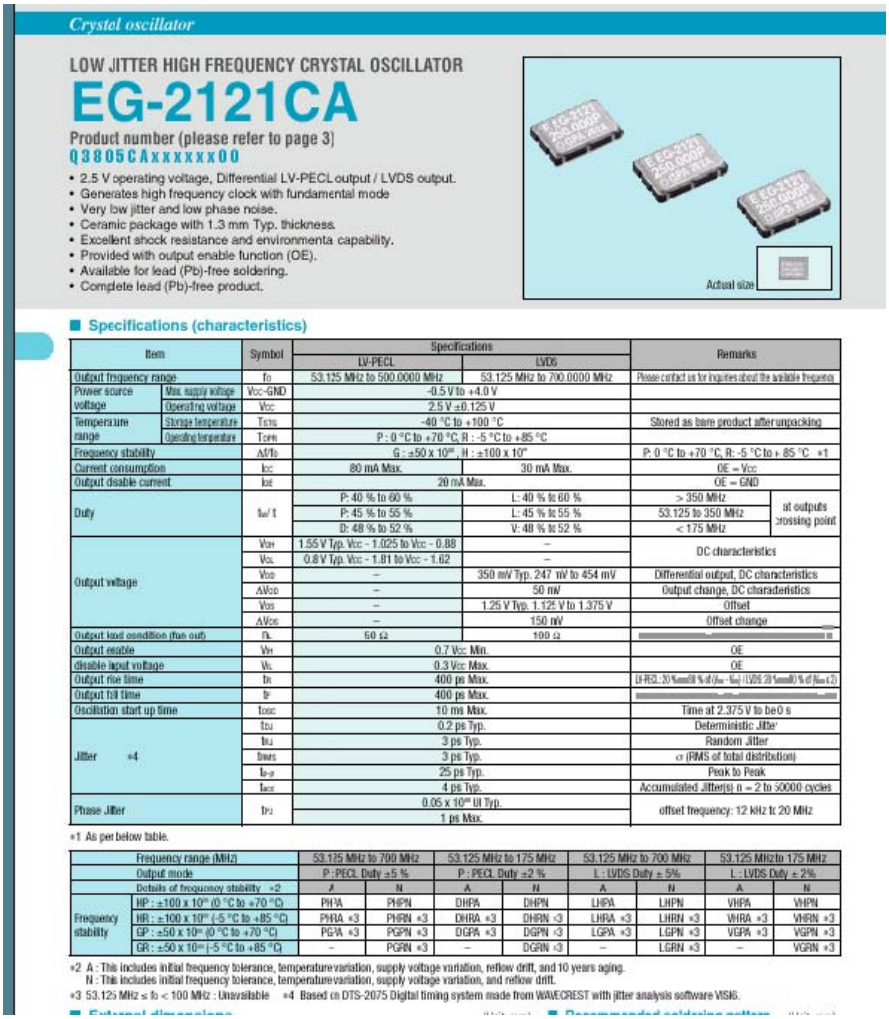
PPM：百万分之一；用来描述非常小的比率。

UI：时间间隔；等价于一个符号的时间长度，例如：0.2UI = 20%的符号时间。

抖动：理想传输位置的偏差。

如此严格的规定才使得PLL和时钟提取电路能够正常工作。通常系统的每一个印刷电路板都需要有一个精确石英晶体振荡器供MGT使用。这些晶体振荡器的精确度比大多数用在数字系统中的晶体振荡器要高一个级别，而且价格也要高出一截。很多情况下，一般的时钟发生芯片和PLL因为带有很大的抖动，而不能用于MGT。

图3-18给出了一些最新发布的时钟生成单元，其性能十分优越，可以用在千兆位级SERDES中。



Crystal oscillator

LOW JITTER HIGH FREQUENCY CRYSTAL OSCILLATOR

EG-2121CA

Product number (please refer to page 3)

Q3805CAxxxxxx00

- 2.5 V operating voltage, Differential LV-PECL output / LVDS output.
- Generates high frequency clock with fundamental mode
- Very low jitter and low phase noise.
- Ceramic package with 1.3 mm Typ. thickness
- Excellent shock resistance and environmenta capability.
- Provided with output enable function (OE).
- Available for lead (Pb)-free soldering.
- Complete lead (Pb)-free product.

Actual size

■ Specifications (characteristics)

Item		Symbol	Specifications LV-PECL	Specifications LVDS	Remarks
Output frequency range		fo	53.125 MHz to 500.0000 MHz	53.125 MHz to 700.0000 MHz	Please contact us for inquiries about the available frequency
Power source voltage	Max. supply voltage	Vcc-GND	-0.5 V to +4.0 V		
	Operating voltage	Vcc	2.5 V ±0.125 V		
Temperature range	Storage temperature	Tstg	-40 °C to +100 °C		Stored as bare product after unpacking
	Operating temperature	Topr	P : 0 °C to +70 °C, R : -5 °C to +85 °C		
Frequency stability		Δf/fo	G : ±50 x 10⁻⁶, H : ±100 x 10⁻⁶		P: 0 °C to +70 °C, R: -5 °C to + 85 °C *1
Current consumption		Icc	80 mA Max.	30 mA Max.	OE = Vcc
Output disable current		Ioe	20 mA Max.		OE = GND
Duty		tw / t	P: 40 % to 60 %	L: 40 % to 60 %	> 350 MHz
			P: 45 % to 55 %	L: 45 % to 55 %	53.125 to 350 MHz
			D: 48 % to 52 %	V: 48 % to 52 %	< 175 MHz
Output voltage		VOH	1.55 V Typ. Vcc - 1.025 to Vcc - 0.88	–	DC characteristics
		VOL	0.8 V Typ. Vcc - 1.81 to Vcc - 1.62	–	
		VOD	–	350 mV Typ. 247 mV to 454 mV	Differential output, DC characteristics
		ΔVOD	–	50 mV	Output change, DC characteristics
		VOS	–	1.25 V Typ. 1.125 V to 1.375 V	Offset
		ΔVOS	–	150 mV	Offset change
Output load condition (fan out)		RL	50 Ω	100 Ω	
Output enable		VIH	0.7 Vcc Min.		OE
disable input voltage		VIL	0.3 Vcc Max.		OE
Output rise time		tr	400 ps Max.		(P-PECL: 20 %vmod% of (VOH - VOL) / (LVDS: 20 %vmod%) % of (VOH x 2))
Output fall time		tf	400 ps Max.		
Oscillation start up time		tosc	10 ms Max.		Time at 2.375 V to be 0 s
Jitter *4		tDJ	0.2 ps Typ.		Deterministic Jitter
		tRJ	3 ps Typ.		Random Jitter
		trms	3 ps Typ.		σ (RMS of total distribution)
		tp-p	25 ps Typ.		Peak to Peak
		tacc	4 ps Typ.		Accumulated Jitter(s) n = 2 to 50000 cycles
Phase Jitter		tPJ	0.05 x 10⁻⁶ UI Typ.		offset frequency: 12 kHz to 20 MHz
			1 ps Max.		

*1 As per below table.

Frequency range (MHz)		53.125 MHz to 700 MHz		53.125 MHz to 175 MHz		53.125 MHz to 700 MHz		53.125 MHz to 175 MHz	
Output mode		P : PECL Duty ±5 %		P : PECL Duty ±2 %		L : LVDS Duty ± 5%		L : LVDS Duty ± 2%	
Details of frequency stability *2		A	N	A	N	A	N	A	N
Frequency stability	HP : ±100 x 10⁻⁶ (0 °C to +70 °C)	PHPA	PHPN	DHPA	DHPN	LHPA	LHPN	VHPA	VHPN
	HR : ±100 x 10⁻⁶ (-5 °C to +85 °C)	PHRA *3	PHRN *3	DHRA *3	DHRN *3	LHRA *3	LHRN *3	VHRA *3	VHRN *3
	GP : ±50 x 10⁻⁶ (0 °C to +70 °C)	PGPA *3	PGPN *3	DGPA *3	DGPN *3	LGPA *3	LGPN *3	VGPA *3	VGPN *3
	GR : ±50 x 10⁻⁶ (-5 °C to +85 °C)	–	PGRN *3	–	DGRN *3	–	LGRN *3	–	VGRN *3

*2 A : This includes initial frequency tolerance, temperature variation, supply voltage variation, reflow drift, and 10 years aging.
N : This includes initial frequency tolerance, temperature variation, supply voltage variation, and reflow drift.
*3 53.125 MHz ≤ fo < 100 MHz : Unavailable *4 Based on DTS-2075 Digital timing system made from WAVECREST with jitter analysis software VISI6.

■ External dimensions ■ Recommended soldering pattern

图3-18 低抖动晶体振荡器规格

时钟修正

传输时钟有非常严格的抖动要求，所以千兆位SERDES通常不能将恢复时钟作为传输时钟。每一个PCB集合都有唯一的振荡器和唯一的频率。如果两个1GHz的振荡器仅仅有1PPM的频差，同时我们提供1/20th的参考时钟，则数据流的时钟每秒钟可能会增加或者缺失20,000个周期。因此，在8b/10b编码的系统中，每秒将会额外增加或者损失2万个符号。

大多数的SERDES都有时钟修正选项。时钟修正需要使用唯一的符号或者符号序列，它们在数据流中是不会出现的。因为时钟修正是对齐的后续处理，所以可以比较容易地通过保留一个K字符、或者一组有序的K字符、或者一个时钟修正数据序列来实现。

某些情况需要使用4个符号的时钟修正序列。时钟修正通过检测接收FIFO来完成其工作。如果FIFO接近于满，则查找下一个时钟修正序列而不将数据序列写入FIFO。这种操作称作丢弃。相反地，如果FIFO接近于空，则查找下一个时钟修正序列，同时它会被两次写入FIFO。这种操作通常也称作重复。

时钟修正进行的频数必须足够多，从而可以通过丢弃或者重复来补偿时钟的差异。时钟修正序列和idle序列通常也是一样的。

有些系统并不需要时钟修正。例如，在很多芯片到芯片（chip-to-chip）的应用中，同一个振荡器为所有收发器提供参考时钟。相同的参考时钟和相同的速率意味着不需要进行时钟修正。同样，如果所有接收电路的时钟都来自恢复时钟，那么时钟修正也是不需要的。如果FIFO的写入速率和读出速率相等，也没有必要进行时钟修正。

如果所有的传输参考时钟都是通过一个外部的PLL锁定在一个公共的参考频率上，那么也不需要时钟修正。对于高精确度的串行数字视频链路来说，这是常用的一种结构。所有的传输时钟都是从一个公共的视频参考中获取的。无法锁定到这个信号往往将导致自由滑动的视频流，相对于其他的锁定信号。在1G或2G速率上实现是比较简单的，但是设计一个能够用于10Gb链路且有足够精度的参考时钟是非常有挑战性的。

表 3-4给出了在不同晶振频率和晶振精度下时钟修正序列之间的最大时钟周期数。

表3-4 时钟修正表

						修正前的最大周期数			
振荡器频率 (MHz)	**晶振精度 (PPM)**	**线路速率 (GB/s)**	**Fmax (MHz)**	**Fmin (MHz)**	**差异/周期 (ps)**	**Remove1 序列**	**Remove2 序列**	**Remove3 序列**	**Remove 4 序列**
156.25	100	3.125	156.2656	156.2344	1.2800	4,999	9,999	14,998	19,998
156.25	50	3.125	156.2578	156.2422	0.6400	9,999	19,998	29,998	‘39,997
156.25	20	3.125	156.2531	156.2469	0.2560	24,999	49,999	74,998	99,998
125	**100**	**2.500**	**125.0125**	**124.9875**	**1.6000**	**4,999**	**9,998**	**14,998**	**19,997**
125	50	2.500	125.0063	124.9938	0.8000	9,999	19,998	29,998	‘39,997
125	20	2.500	125.0025	124.9975	0.3200	24,999	49,998	74,998	99,997
62.5	100	1.250	62.5063	62.4938	3.2000	4,999	9,998	14,998	19,997
62.5	50	1.250	62.5031	62.4969	1.6000	9,999	19,998	29,998	‘39,997
62.5	20	1.250	62.5013	62.4988	0.6400	24,999	49,998	74,998	99,997

接收和发送缓冲器

接收和发送缓冲器，即FIFO，是千兆位级收发器的主要数字接口。FIFO通常是数据写入和读出的地方。发送端通常有一个小型的FIFO，它要求读取和写入的时钟是等时同步（isochronous）的（频率匹配但相位不一定匹配）。

等时同步（isochronous）：频率匹配但是相位不一定匹配。

如果tx_write和tx_read选通信号不是工作在精确相同的频率，则通常采用另外的方法。此时，需要使用一个较大的FIFO，同时要求持续不断地检测FIFO的当前状态。如果FIFO被不断地填充，将最终导致溢出。在这种情况下，必须在输入数据流中检测idle符号。如果检测到idle符号，则不把idle符号写入FIFO。

反过来，如果FIFO运行较慢则在输出数据流会出现idle符号，数据被传送给用户。此时写指针保持不动，不断重复idle符号。使用idle符号而不使用字节对齐、comma字符、时钟修正序列或者通道绑定序列，这一点是非常重要的。为了保证一定的发送速率，所有这些序列都是必需的。

相对于发送缓冲器而言，MGT内建的接收FIFO通常需要有更深层次的考虑。它的主要目的是为了实现时钟修正和通道绑定。

通道绑定

有时候我们需要传送的数据会超过一条串行链路的承载能力。在这种情况下，可以同时使用多条链路来并行传输数据。如果使用这种方式，则输入的数据流必须是对齐的。这个过程通常称作通道绑定。通道绑定可以吸收两个或多个MGT之间的偏差，将数据提交给用户，就像只使用一条链路进行传送一样。

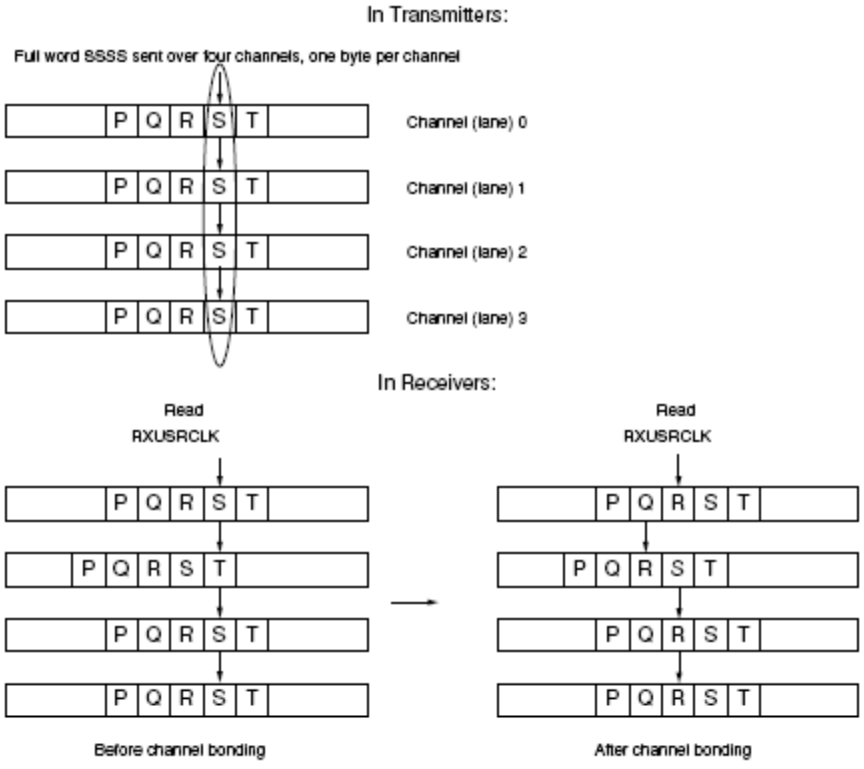


图 3-19 通道绑定框图

通道绑定：吸收两个或多个MGT之间的偏差，将数据提交给用户，就像只使用一条链路进行传送一样。

不同MGT间数据偏差的一些主要原因：

- 传输通道长度的偏差
- 传输通道的有源中继器
- 时钟修正引起的偏差
- 锁定和字节对齐引起的时间偏差

因为通道绑定需要涉及到收发器之间的通信，所以具体的细节因厂家、器件而异。但是它们有一些共同的特性，例如：指定一个通道作为主通道，指定从通道，还可能需要指定前向从通道。三级通道绑定包括一个主通道和前向从通道，所以通常也称为两-跳通道绑定。

通道绑定序列必须是唯一的而且是可扩展的，因为可能会添加或丢弃通道绑定序列，所以下行链路中必须忽略。通常时钟修正序列和通道绑定序列之间会有最小间隔符号数。很多基于8b/10b的标准协议规定时钟修正序列和通道绑定序列之间至少需要间隔四个符号。因此，四个符号或字节是比较常用的间隔距离。

物理信号

千兆位级SERDES的物理实现普遍采用基于差分的电气接口。常用的差分信号方法有三种－低电压差分信号（LVDS）、低电压伪射级耦合逻辑(LVPECL)和电流模式逻辑(CML)。千兆位链路通常使用CML。CML采用最常用的接口类型，并且通常都会提供AC或DC端接以及可选的输出驱动。部分输入还提供了内建的线路均衡和/或是内部端接。通常端接的阻抗也是可选的。

前端和后端共同组成了物理接口

CML：电流模式逻辑；一种基于差分的电气接口，很适合于千兆位链路。

图3-20给出了一个CML型的驱动电路。这些高速驱动电路的原理非常简单。两个电阻中的一个始终都有电流通过，并且此电流和通过另一个电阻的电流不同。图 3-21 给出了一个MGT接收器的示意图。

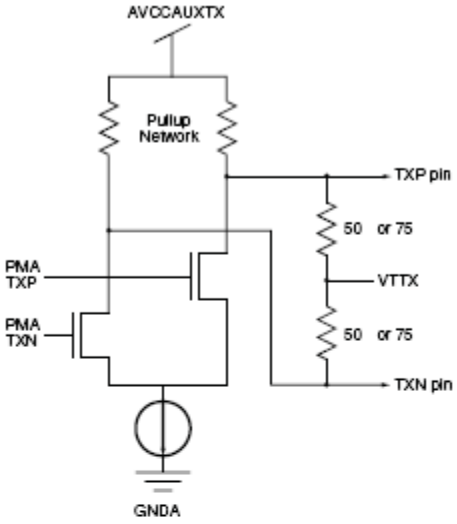


图 3-20 CML 驱动电路

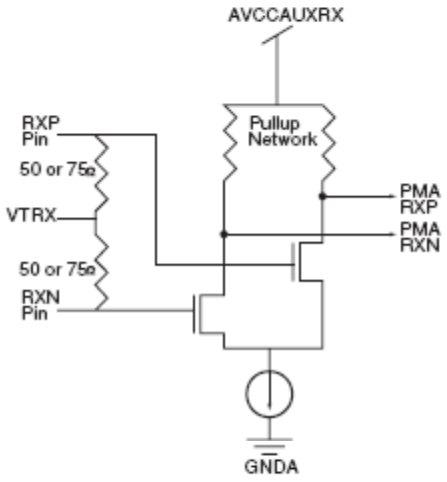


图3-21 MGT接收器

图3-22和图3-23给出了模拟前端和后端的数据资料。

Parameter		Min	Typ	Max	Units	Conditions
V_{IN}	Serial input differential peak to peak (RXP/RXN)	175		2000	mV	
V_{ICM}	Common mode input voltage range	500		2500	mV	
T_{ISKEW}	Differential input skew			75	ps	
T_{JTOL}	Receive data total jitter tolerance (peak to peak)			0.65	UI(1)	
T_{DJTOL}	Receive data deterministic jitter tolerance (peak to peak)			0.41	UI	

Notes:

1. UI = Unit Interval

图3-22 差分接收器参数

Parameter		Min	Typ	Max	Units	Conditions
V_{OUT}	Serial output differential peak to peak (TXP/TXN)	800		1600	mV	Output differential voltage is programmable
V_{TTX}	Output termination voltage supply	1.8		2.625	V	
V_{TCM}	Common mode output voltage range (no transmission line connected)	1.1		1.5	V	
V_{TCM}	Common mode output voltage range (transmission line connected)	1.1		2.0	V	The common mode deperds on coupling (DC or AC), VTTX, VTRX, and differential swing. Spice simulation gives the exact common mode voltage for any given system.
V_{ISKEW}	Differential output skew			15	ps	

图3-23 差分发送器的参数

预加重

千兆位级驱动器最重要的特性可能就是预加重的能力。预加重是在转变开始前的有意过量驱动。如果没有相关的经验，这看起来会像是一个缺陷；看起来就象是一个不好的设计可能发生的上冲和下冲。为了弄懂这么做的意图，我们需要理解符号间干扰。

预加重：转变起始的有意过量驱动。

如果串行流包含多个比特位时间的相同数值数据，而其后跟着短比特位（1或2）时间的相反数据数值时，会发生符号间干扰。介质（传输通道电容）在短位时间过程中没有足够的充电时间，因此产生了较低的幅度。

码间干扰（ISI）：符号间的干扰－如果串行流包含有多个比特位时间的相同数值数据，而其后跟着短比特位（1或2）时间的相反数据数值时，会发生符号间干扰。

在符号间干扰的情况下，长时间的恒定值将通道中的等效电容完全的充电，在紧接着的相反数据数值位时间内无法反相补偿。所以，相反数据的电压值有可能不会被检测到。图3-24、图3-25、图3-26给出了这种现象的示意。这个问题的解决方法是：转变开始时加入过量驱动，而在任意的连续相同数值时间内减少驱动量，这种过程有时也称作去加重。

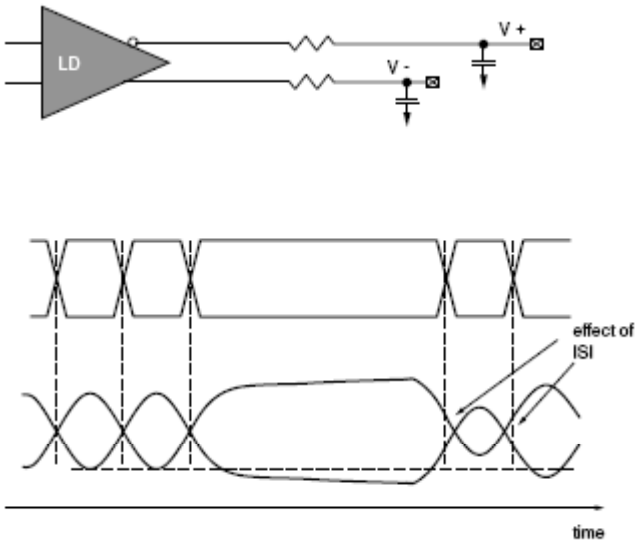


图 3-24 符号间干扰

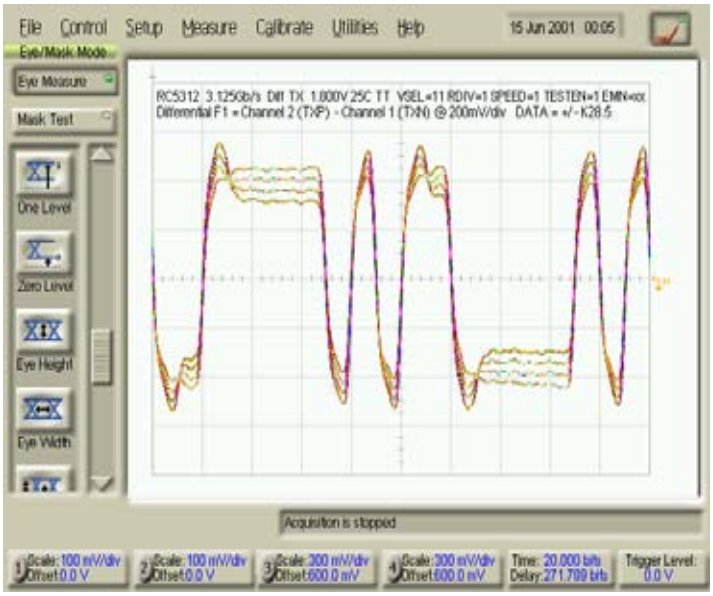


图3-25 DCA屏幕拷贝

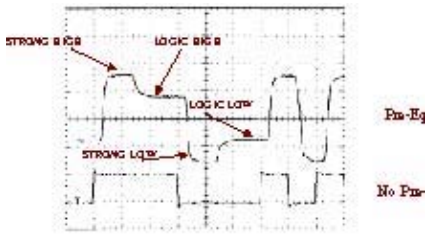
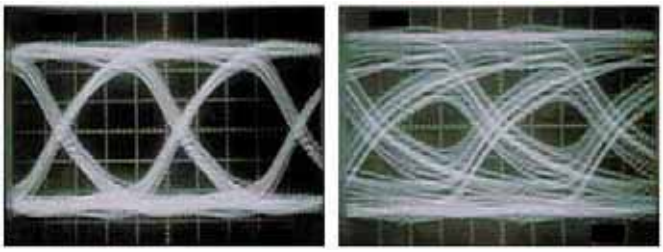


图 3-26 预加重

从图3-27中两个眼图可以看出，使用预加重减少码间干扰后，眼图的睁开度得到了改善。



Pre-Emphasis No Pre-Emphasis

图3-27 预加重和没有预加重的眼图

眼图：数字采样示波器上通常显示的波形。眼图用于指示信号质量。抖动、阻抗匹配以及幅度都可以用眼图来刻画。

预加重可以通过两个平行的CML驱动来实现，其中的一个CML要落后一个位时间。图3-28 和图3-29给出了一个示例电路及其波形，该电路通过驱动晶体管获得输出信号。

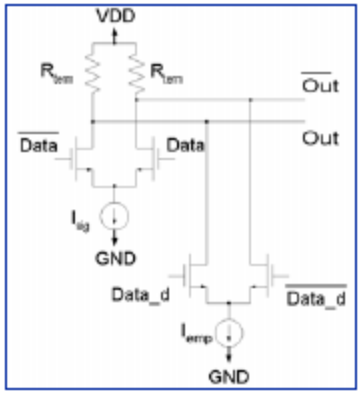


图 3-28 预加重 CML电路

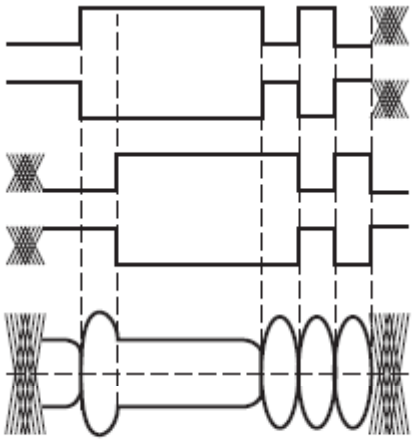


图3-29 图3-28中所示CML电路的时序图

差分传输线路

数字设计工程师和PCB设计师们曾经一度认为布线只不过是简单的互连或连线。实际上，原型建造时通常采用一种叫做蛇行布线的技术。实际中并不是一定要运用传输线和传输线理论。如果线路的传输延时只是信号上升时间中的很小一部分，我们可以不使用传输线理论。

但是随着信号的频率增大，PCB设计过程中就必须使用传输线理论。

对于千兆位级操作而言，不仅包括传输线和阻抗控制，还包括差分线路对的阻抗控制。

差分线路对阻抗匹配的两条线路是相邻的。两条线路之间的间隔使得两线路相互耦合。如果两线路相隔较远，则称作弱耦合（图 3-30）。如果相隔较近，则为强耦合（图 3-31）。

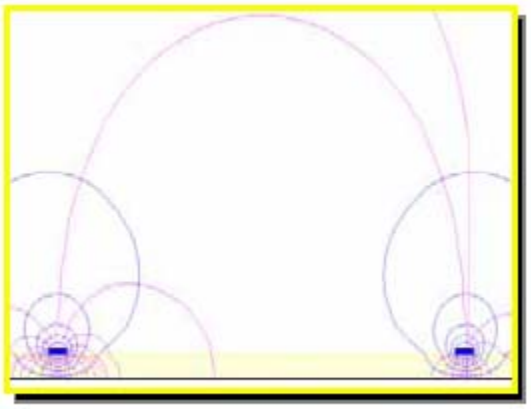


图3-30　弱耦合

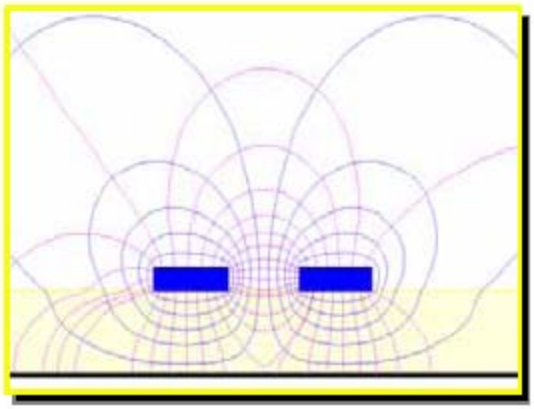


图3-31 强耦合

如果给定线路的长度以及叠层结构（会带来给定的阻抗），耦合还会影响线路的阻抗。相同几何形状的差分线路对也会有不同的阻抗。阻抗的精确大小因材料而异，但是板制作厂商通常会提供精确的数据。

有一种称作*field solver*的工具/数学模型也可以用来计算阻抗的大小。图3-32给出了几种主要类型的受控阻抗差分线路－微带线、带状线、偏置带状线以及层间耦合线路。表 3-5 中给出了受控阻抗差分线路的各种类型。

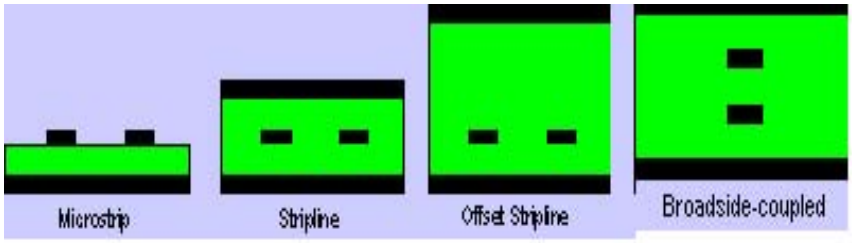


图3-32 差分线路的受控阻抗

表 3-5不同受控阻抗类型的差分线路

类型	优点	缺点
微带线	较小的走线内损耗	只有两层（顶层和底层）对干扰敏感
带状线	较好的屏蔽，更多的层	高频信号的单位英寸振幅损耗高于微带线
偏置带状线	用于非对称的叠层结构。可以用于限制电源层/底层的数量。	如果用于减少层数，则走线上方的偏置区域必须没有其他走线，而且不可以有平行走线。
层间耦合线路	十分紧密的耦合。	层间耦合线路是很难制造的，因为它的容差是很严格的。在千兆位级设计中。不建议使用。

线路均衡

均衡主要用于补偿由频率不同而引起的阻抗/衰减差异。均衡器有很多种形式，但总体上可以分为有源和无源两种。

有源均衡器：依赖频率的放大器/衰减器。

无源均衡器：无源电路，其频率响应可以补偿传输衰减，和滤波器类似。

无源均衡器是无源电路，其频率响应可以补偿传输衰减。无源均衡器可以认为是一个滤波器。如果我们的滤波器可以使传输线所使用的各频率通过，而将传输线没有使用的其他频

率滤除，那么整体的频率响应就会变得平坦许多，如图 3-33。

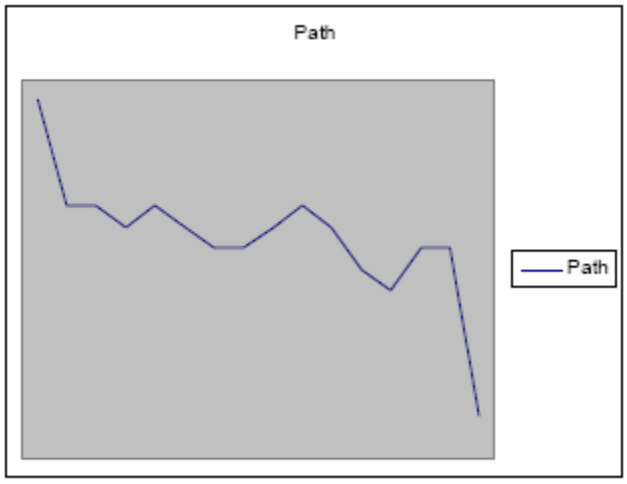


图 3-33 未均衡的系统频率响应示例

有源均衡器可以认为是依赖频率的放大器/衰减器。有源均衡器主要有两种：固定形式有源均衡器和自适应有源均衡器。对于任意的输入数据流，固定形式有源均衡器的频率响应都是一样的（图 3-34 和图 3-35）。

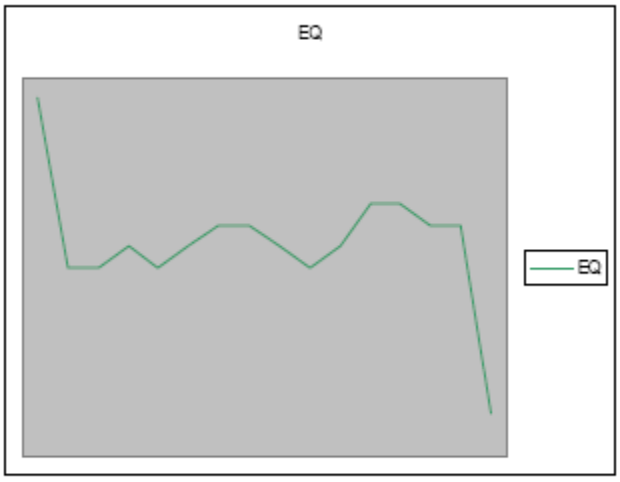


图 3-34 均衡器的频率响应示例

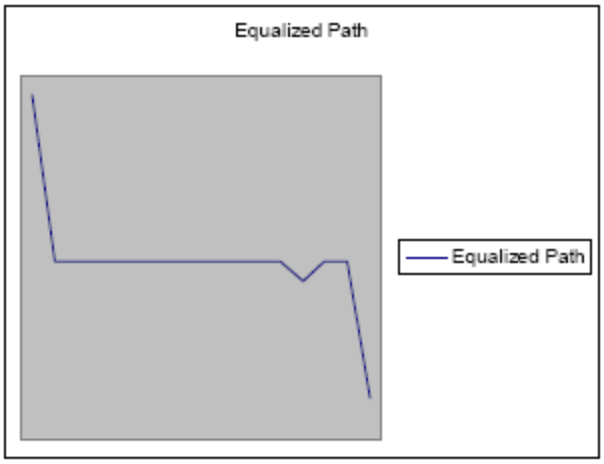


图 3-35 均衡系统频率响应示例

固定形式均衡器的增益/衰减量通常是用户可选择的，或者可编程的。部分均衡器有一个简单的控制参数—n 用于设置高增益或低增益，类似于简单音响系统中的低音设置。还有一些均衡器可以独立设置不同频率的增益/衰减量，这和复杂音响系统中的均衡设置是类似的。图 3-36 给出固定形式均衡器的一种可能的频率响应。

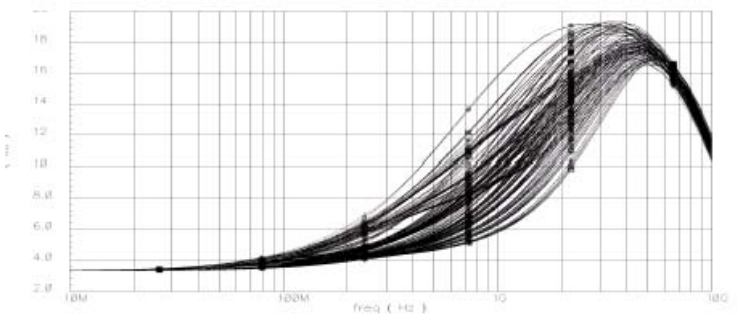


Figure 4-14: **Magnitude (dB) vs. Frequency (Hz) Plot for all 1024 states of RXFER[9:0]**

图 3-36 均衡器频率响应示例

自适应均衡器（也称学习型均衡器）要复杂的多，自适应均衡器需要分析输入信号并检测哪些频率在传输通道中被削弱。测量和调节是以闭环形式实现的。自适应均衡器的频率响应取决于输入的比特流。

固定形式均衡器和自适应均衡器是有源均衡器的两种不同类型。

自适应均衡器通常和特殊形式的线路编码机制协同工作。自适应均衡器对于可变通道的链路来说是最合适的，可变通道可以是可变的电缆长度，或是显著的位置依赖的背板系统。

固定形式均衡器比较适合于不变系统中，例如：芯片到芯片，平衡化的背板系统以及固定长度电缆的系统。均衡器通常包含在 SERDES 的模拟前端，或者作为系统的一个独立部分（图 3-37）。

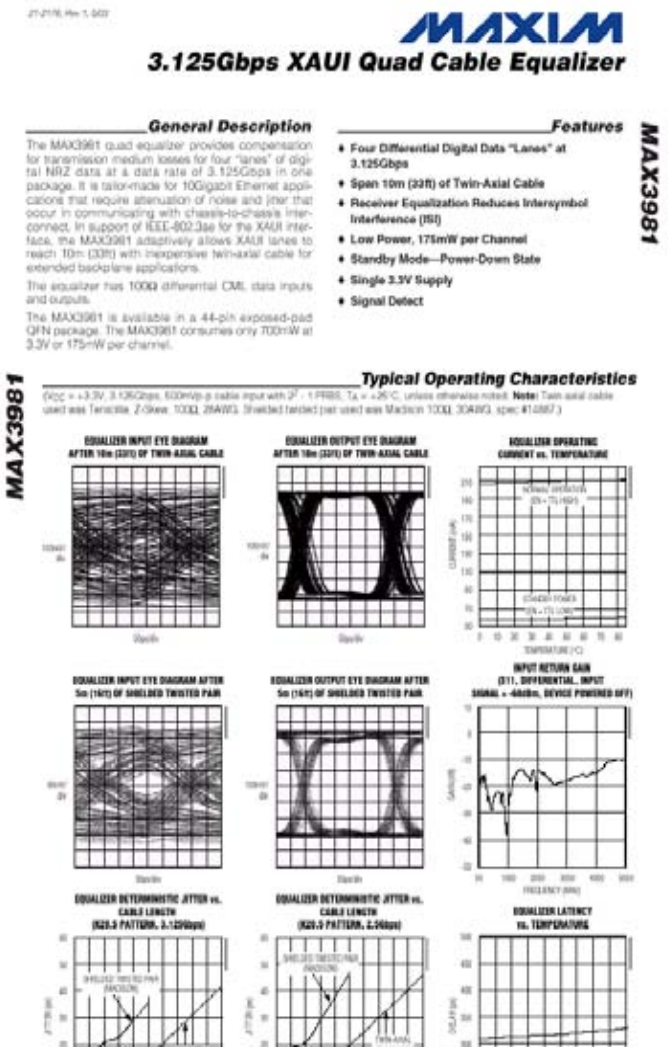


图 3-37 外部 3G 均衡器的说明示例

电缆也可以被均衡化。最常用的电缆均衡技术是在电缆集合中添加有源均衡电路，通常添加在连接器中。

一些高端电缆有均衡化的特性，它们采用了先进的电缆制造技术，例如镀银铜芯电缆（图 3 -38）。

图 3-38 均衡电缆内部组件

光解决方案

如果系统中电缆要传输的距离很远（远大于相邻底板的距离），那么通常采用光解决方案。使用光纤可以实现多种长距离传输，例如：楼下到楼上，楼与楼之间，街区之间或者城镇之间。

最基本的光纤系统包括：发送器或信号源、光纤以及接收器。

光纤系统使用光信号取代电信号来传输信息。最基本的光纤系统包括发送器或信号源、光纤以及接收器，接收器将光脉冲重新转变为电信号。信号源通常是注入型激光二极管（ILD）或者发光二极管（LED），如图 3-39 所示。

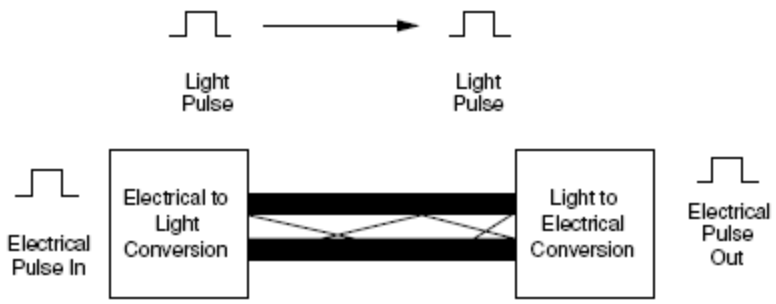


图 3-39 基本的光纤传输系统

光纤中的光脉冲传输是基于全反射定理的。全反射定理：如果入射角大于临界值，则光线不会透射而会全部反射回来。简单的说，光纤可以看作是一个内部全是镜子的弹性管线。光线在管道中不停反射前进，就算管道发生了弯曲，光线也能一直前进到达末端。

全反射：如果入射角大于临界值，则光线不会透射而会全部反射回来。

光线有两种类型—单模光纤和多模光纤（图 3-40 和图 3-41）。单模的价格较高，可以传输的距离也较长。多模光纤的价格较低，只能用于短距离传输。基本的光连接器如图 3-42 所示。

图 3-40　SMF 单通道

图 3-41 NMF 多通道

图 3-42 基本类型的光纤连接器

误码率

误码率（BER）是千兆位链路设计师很关心的问题，特别当链路是由并行设备到串行背板系统时。没有一个链路是零误码率的，因为总是会有发生错误的潜在可能性。在多数低速率系统中，错误的起因最可能是宇宙射线的干扰。但是这种可能性实在太小了，所以本质上可以认为是零。那么为什么串行链路是不同的呢？

主要有三个原因：

1. 宇宙射线可以引起错误，特别是信号转变的时候。信号的速度越快，转变越多，则转变时越可能受到宇宙射线的干扰。
2. 对于任何给定的 BER，信号的速度越快，发生错误的可能性就越大。
3. 高速时钟数据恢复不够精确。抖动、ISI、以及其他的实际干扰都可能造成不好的数据判决，从而引起错误。例如，PLL 不断尝试调整持续变化的输入信号。同时，由于振荡器会随着温度漂移，所以也可能会引起错误。

测试的真实性

上述几个原因的影响确实存在，仔细分析并行背板、源端同步链路或者任何通信通道，就会发现类似的问题。但是大部分情况下，还是假定 BER 很接近于零并且忽略不计。

为什么在千兆位 SERDES 中不能有同样的结论呢？因为它们的原始背景是使用各种光学设备的通信产业。这个产业通常很关注 BER，通过测试、设计并最后具体指定 BER。这一点极大地影响了千兆位 SERDES 链路的 BER。

部分标准，例如 XAUI 和部分版本的 SONET，会给出其可以接受的最大 BER。不幸的是，BER 的测试十分困难、令人厌烦而且需要花费很多时间。同时改善单位 BER 的难度会呈指数增长。BER 通常以 10-x 的符号出现，所以要把 BER 从 10-8 改善到 10-9 需要 10-x 的时间。而且从某些角度来看，测试是不切实际的。因此，多数的厂商测试只会持续到标准中给出的最严重的 BER，而不会继续下去。

CRC

设计师还是需要设计一个稳健的系统。首先，他需要检查系统的要求，看是否能够使用常用的方法来解决问题。

一种方法是错误检测数据重传。检查输入数据中是否有错。如果发现错误，则发送信息给发送者要求重传数据。错误检测的首选方法是 CRC。因为 CRC 十分常用，所以许多 SERDES 内部都有 CRC 发生器和检测逻辑。通常重传请求是由上层协议定义的。如果协议支持 CRC 和重传，或者数据要求正是其所能满足的，那么这种方法将会是最好的选择。

如果情况不是这样的话，设计师还可以有其他的选择。设计师可以建造并测试所设计的系统，以观察其能否正常工作。SERDES 发布的 BER 可以用来确定测试需要进行到什么程度，所以设计师还是有一些机动空间的。设计师不可能设计出一个远优于发布数据的系统。除了持续测试直到达到发布的 BER，设计师还需要在各种极限情况下进行测试（例如，输入抖动十分靠近容限）。如果给系统设计提供更好的输入流，那么结果会更好。

数据还提供了另一种值得考虑的选择。多数的数据流都是有模式的，和用于 BER 测试的伪随机比特流相比，数据流更容易预测。这一点可能是好处也可能是坏处，这取决于传输通道和均衡器适配数据流的情况。所以必须进行测试和调整。

所以建造一个系统并观察其能否正常工作，这种方法并不是十分牵强的。尽管如此，如果这种方法会出现操作上的问题，那么前向纠错（FEC）可能会有所帮助。

某些应用中的 FEC

由于设计师知道可能会发生错误，所以可以通过提供冗余数据位来恢复这些错误。

FEC：前向纠错—添加额外的位，用于帮助恢复错误数据

让我们来看看 FEC 是怎样工作的。考虑一个待传输的数据块，其大小为 NxR 字节，分为 R 行，每行 N 字节。现在给矩阵的每一行附加额外的一个字节，并给矩阵附加额外的一行。这些地方就是额外的位置。

数据块的附加信息就保存在这些额外的位置中。此例中，额外的信息是奇偶位。附加字节的每一位代表此行中各字节对应位的奇偶性。也就是说，P[1][0]是 D[1.1][0] D[1.2][0] D[1.3][0] …. D[1.N][0]的奇偶性。而对于额外的行而言，其中的每一位就是对应列中各位的奇偶性。也就是说，P[R+1.0][0]是 D[0.0][0], D[1.0][0] D[2.0][0] ….D[N.0][0]的奇偶性。矩阵的示意图如图 3-43 所示。

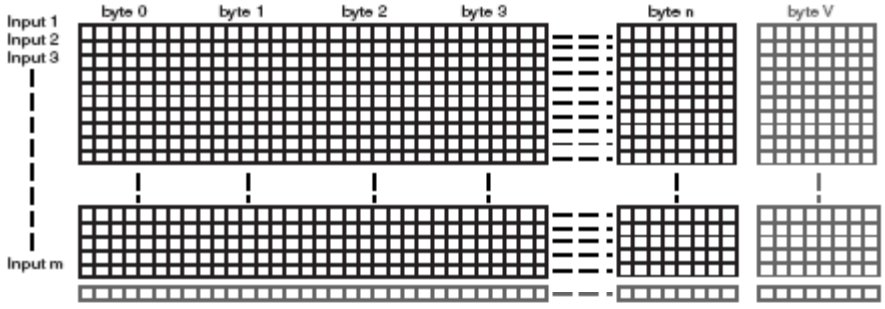


图 3-43 FEC 框图

数据和附加位同时通过链路传输。在另一侧，接收器会检查矩阵的奇偶性。如果数据的任一位是错误的，那么它会标记出来，并通过行值和列值来确定位置。只需要简单的取反，即可纠正该位的错误。多位的错误即可能被纠正，也可能会导致混乱而且会阻止其他错误的纠正，这取决于错误发生的位置。

这种方法通常称作简单矩阵奇偶性法，而且也是 FEC 的最初类型。这也是多数 FEC 方法基本模块。这个例子是简单易懂的，但是它有局限性。针对恶劣环境和性能不好的传输通道，已经开发出多种 FEC 方法，例如 Viterbi，Reed-Soloman 和 Turbo 编码。所有这些方法都有强大的纠错能力，但是纠错也是有代价的：

- **运行速度不够快：**与大多数可以在常规结构下发挥作用的方法相比，千兆位级 SERDES 的速度较快。
- **编解码器太过庞大：**编码器和解码器的电路逻辑数量可能会是 MGT 及其剩余部分的十倍。
- **编码开销过大：**编码开销就是那些附加的位。编码开销过大常常可以使一种 FEC 方法不可行。

SERDES 技术促进 I/O 设计的发展

SERDES 技术中的各种可用功能极大地促进了 I/O 设计的发展。SERDES 的各种功能例如 RX 定位、时钟管理器、发送/接收 FIFO、线路编码器/解码器等被广泛用于提高速度和精确度。SERDES 在未来 I/O 设计中扮演着重要角色，它提供的各种功能也将是高效 I/O 器件设计的重要工具。

第四章

千兆位串行 IO 设计

理解所面临的挑战和权衡折衷

千兆位级收发器设计面临的挑战

理解所面临的挑战是解决工程问题的关键。在设计千兆位级收发器（Multi-Gigabit Transceiver，MGT）时，面临的挑战包括：理解收发器协议、信号完整性、阻抗和功率要求、屏蔽性要求、印刷电路板（PCB）设计要求以及连接器和电缆的选择要求。原型的仿真和测试对于一个成功的 MGT 设计而言也是相当重要的。

设计时应考虑的事项及可提供的选择

本章从板极角度讨论了 MGT 设计者将面临的挑战和抉择，同时针对以 SERDES 为核心的设计给出了处理这些问题的一般方法。本章介绍了各种可用的传输协议以及各自的优缺点。本章还讨论了信号和电源的设计注意事项，以及高速设计中屏蔽的重要性。同时，本章还讨论了印刷电路板设计的要求和如何选择合适的连接器、电缆。最后，而且是最重要的，本章给出了 MGT 设计的仿真方法（模拟和数字）。最后本章深入探讨了原型的测试和测量，同时给出了一些重要的调试提示和 MGT 设计的建议。

协议

串行器 / 解串器（Serializer/Deserializers，SERDES）本身就是相当灵活的设备。为了启动 SERDES，首先需要定义对齐序列，时钟修正序列，线路编码方法和物理链路，之后数据可以在两个收发器间相互传送。但是所传送数据的含义还需要有更详细的定义，这也是协议存在的意义。什么数据传送到何处，数据的含义是什么，数据中需要插入什么特殊位，什么样的数据可以被丢弃，这些都是由协议定义的。

MGT 相关协议的范围很广，从简单的数据定义到支持上层协议的复杂接口。千兆位级协议中的具体内容包括：

- **数据格式：**视频和音频协议的值定义；如何通过 0 和 1 来代表特定的值或特定含义。
- **子通道：**通常在一个链路中需要有多个不同的通道，子通道的主要用途包括控制、状态和辅助数据通道。
- **数据提取：**协议的一个通常功能是定义如何将数据和开销分离。这个功能通常称作数据提取或者反嵌入。
- **嵌入：**协议通常还定义怎样将数据嵌入到协议流或包中。对于遵从协议栈模型的协议而言，这个功能是很必要的。
- **错误检测和处理：**协议通常都会定义如何检测错误以及错误发生时的应对操作。
- **流量控制：**协议中往往还定义流量控制。流量控制的内容很多，包括动态缩放子通道的带宽分配，以及调整空闲时隙的插入速率以满足时钟修正的需要。
- **寻址/交换/转发：**如果串行协议是针对点对点的应用，则不需要寻址时序。而更复杂的协议通常含有寻址时序，在寻址的基础上可以实现转发和交换。
- **物理接口：**协议具体定义驱动电平、预加重等等，以确保各器件间的兼容性。

通常协议的选择是比较简单的。如果要设计一个 PCI Express 卡，则选用 PCI Express 协议。但是如果要设计一个专有系统，那么系统架构师就需要选择，使用预定义的协议，还是设计一个定制协议。

标准协议

下面的几页列出了部分工业标准协议的摘要（这些标准协议的具体定义可能有很多页的内容，所以这里不详细列出）：

XAUI：4 通道接口（2.5 Gb/s 有效载荷，3.125 Gb/s 传输线速度），用于 10G 以太网。

PCI Express：由旧的并行 PCI 结构改进得到的快速串行结构。上层的协议依旧是兼容的，可以很容易适配到旧的 PCI 系统。

Serial RapidIO：旧的并行协议的串行版本，RapidIO 相当灵活，可以用于多协议间的接口（例如 PCI 和 Infiniband）。

FiberChannel：FiberChannel 一直以来都是串行协议，不过其速度在不断增长。随着铜线链路的改进，FiberChannel 即可用于光纤通道也可用于铜线通道。

Infiniband：一个盒到盒（box-to-box）的协议，可以运行在铜线或者光纤上。Infiniband 类型的电缆广泛应用于短距离（几米）的千兆位级链路。此协议适用于多种设备，并支持很大的复杂性。协议内还定义了中继器、交换机和集线器的规范，以增加可连接设备的数目。Infiniband 还可用于使用 Infiniband 交换和控制台的复杂系统配置（如图 4-1）。

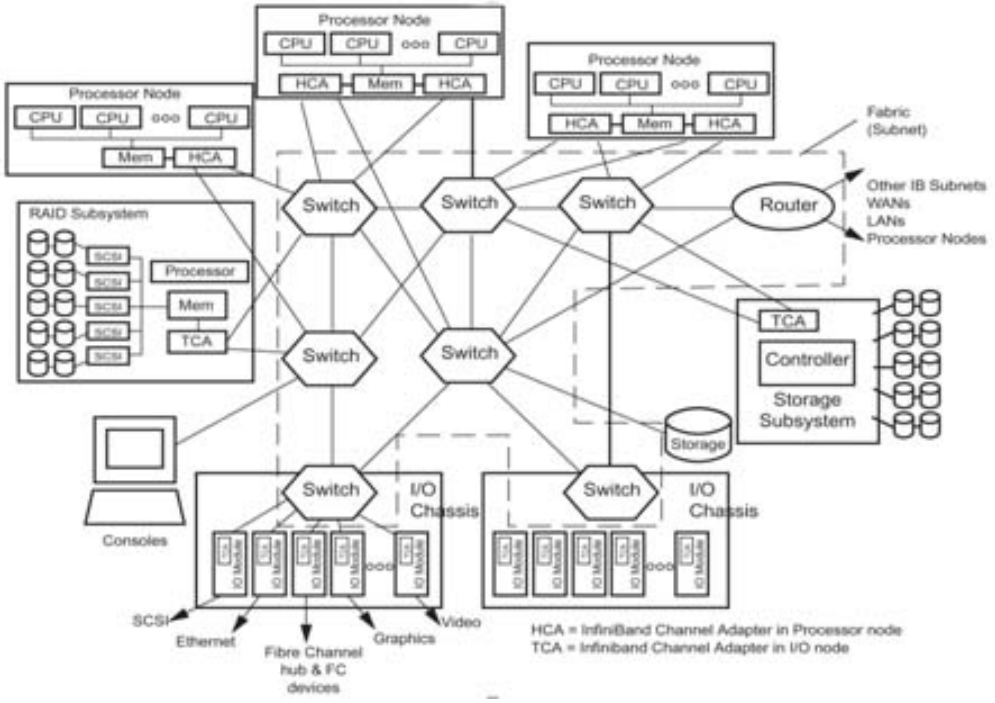


图 4-1: **Infiniband 交换和控制台**

Advanced Switching：交换结构协议和 PCI Express 是基于相同的物理和数据级协议。针对交换结构领域设计特定的协议是很有意义的。

PICMG：PICMG 是一个超过 600 家公司的联盟，该联盟致力于制定高性能标准化背板结构的开放标准。PICMG 的很多标准使用其它的工业标准，例如 PCI 和 Infiniband。

图 4-2: **ACTA 卡示例**

ATCA：ATCA 的全称是 Advanced Telecom Computing Architecture，即高级电信计算架构，又称作 AdvancedTCA。这个 PICMG 标准是为下一代电信设备制定的。它的目标是在提供一个十分灵活、可扩展的系统的同时，尽量简化不同制造商生产的设备之间的互操作。针对某个课题，此标准可以提供多种实现方案，包括星形连接结构的背板、冗余星形连接结构的背板和全连接的网状结构。

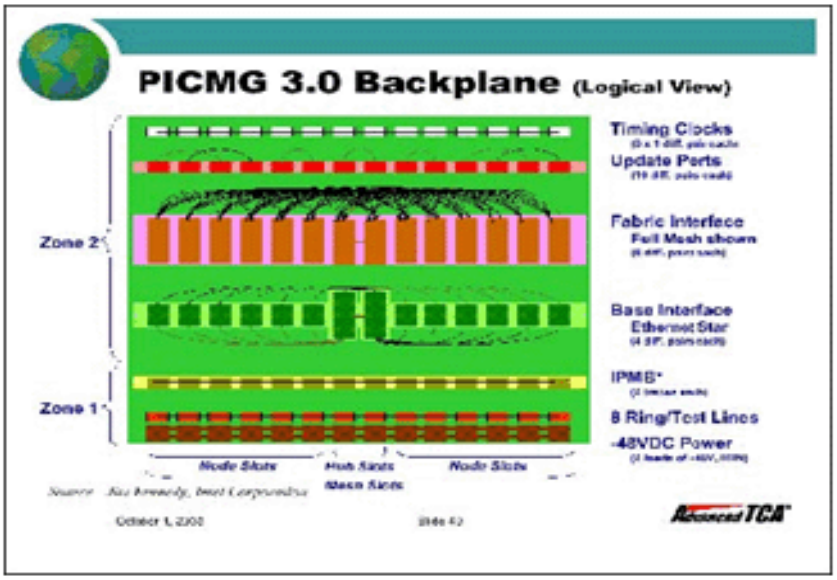


图 4-3: **PICMG 3.0 背板**

Aurora：Aurora 是一个相对简单的协议，只控制链路层和物理层。Aurora 的设计理念是使其它高层协议，例如 TCP/IP 和以太网，可以很容易的运行在 Aurora 之上。Aurora 协议使用 1 个或多个高速的串行通道（如图 4-4 所示）。

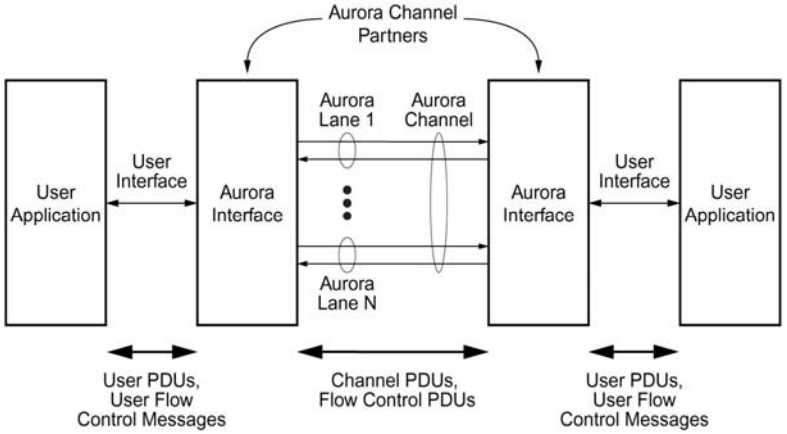


图 4-4: **Aurora 通道框图**

Aurora 不仅定义了物理接口，而且定义了包结构、嵌入其它协议包的推荐程序、数据提取和流量控制。协议中定义了有效链路的初始化程序，同时还描述了禁止使用发生过量错误的链路的相关程序。由于协议中没有寻址时序，所以不支持交换。协议中也没有定义错误检测、重传或有效载荷的纠错。此协议是由 Xilinx 开发的，并且无限制地开放给公众自由使用。

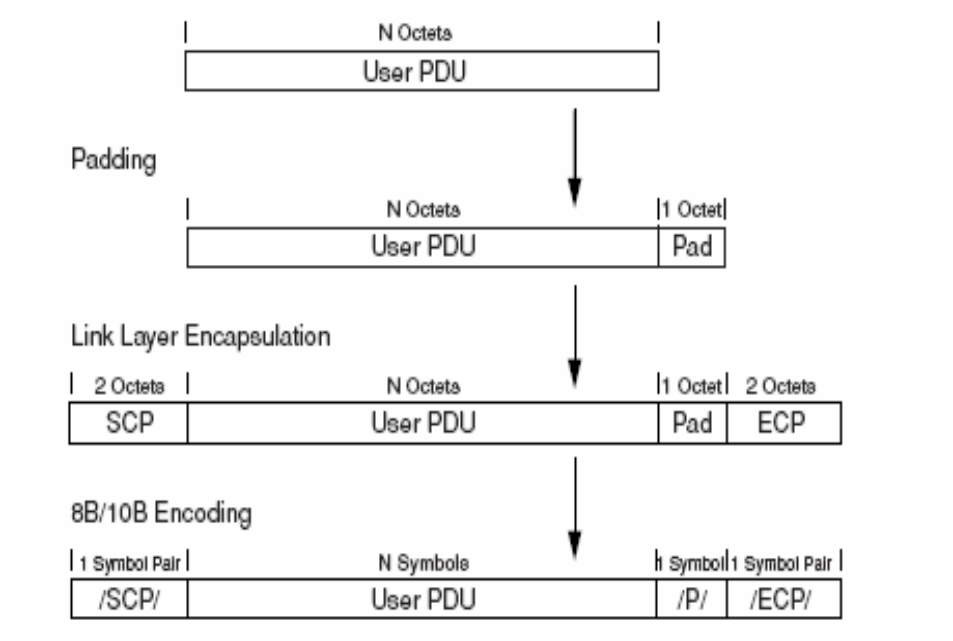


图 4-5 :**Aurora 加载其它协议到数据流中**

定制协议

在某些情况下，用户可能希望制定自己的协议。特别是当标准协议不能满足要求，或者标准协议对于用户的应用来说太过宽泛时，制订用户自己的协议是个很好的选择。当然，有时用户可能也需要一个新的复杂协议，但是这种情况通常留给制定标准的专业协会。

通过一个简单的例子，我们很容易了解到在制定自己的协议时应当考虑的各种事项。在这个简单的应用中，我们需要将恒定的 1.8 GHz 信号流从一块板传送到另一块板。系统的输入输出使用 12 位的总线，工作在 150MHz。针对这个简单的应用需求，协议中需要定义的内容包括：数据帧结构、对齐和 idle（空闲）字符。此例中，我们使用 8b/10b 作为线路编码机制，并从其它 8b/10b 标准中借用标记及 comma 字符的定义。链路的基本结构如图 4-6 所示：

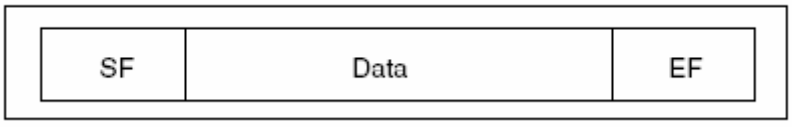


图 4-6: **基本链路结构**

首先我们需要为SF（帧开始）、EF（帧结束）和idle（空闲）指定字符或者有序的字符集，之后需要确定线路速率和数据帧大小。适当设置数据帧的大小，以保证对齐时有充足的SF符号，并且进行时钟修正能够有足够的idle符号。在这个例子中，因为我们要传送1.8 GHz的载荷，所以我们选定传输线速率为2.5Gb/s。其有效载荷速率为2Gb/s，可以满足1.8GHz的数据需求，其额外的容量还可用于所需开销。

sf: 帧开始; **ef:** 帧结束

因为我们要实现板与板之间的传送，肯定需要使用不同的振荡器来驱动各自收发器的传送时钟，所以我们需要考虑时钟修正问题。在决定数据帧的大小时，我们需要折衷两个相互矛盾的要求。数据帧越大，开销就越小，而且数据可以拥有更多的带宽。数据帧越小，就可以有更多的对齐符号和时钟修正符号。显然，我们需要平衡这两种要求。我们应当计算两者各自的界限，并选取一个中间状态。

数据传输容量很容易计算。我们共有 2Gb/s 的可用带宽，其中 1.8Gb/s 用于传送数据，所以只要开销能承载在剩余的带宽中就可以了。下面我们选用一个合适尺寸来看看其能否工作。

如果数据帧中承载 2048 字节且开销为 10 字节（SF-2 字节，EF-2 字节，idle-6 字节），则开销率为 10/2058，约为 5%。我们可以承受的开销率为 0.2/2 或 10%。所以就开销预算而言，我们可以把数据帧缩小 20 倍。

如果我们需要进行时钟修正，那么我们希望数据帧小点，所以需要考虑下面所述的事项。我们的MGT要求参考时钟的精度达到 20 ppm。如果我们把MGT设置为 2 符号宽的idle序列，则时钟修正最多每 49,999 个符号进行一次。所以，idle字符之间的距离应当小于这个数。多数情况下，我们希望其值小于最大idle字符距离的 1/3，我们选择的值大约为 1/24th。至此我们基本上已经定义好我们的简单协议，唯一剩下的事情就是处理流量控制问题。

记住我们的开销只占用了 0.5%的时间，但是实际上有 10%的可用时间。所以我们需要定义在剩余时间中填充什么内容以及由谁来控制填充。我们在多余的时间中填充 idle 符号，所以实际的线路如图 4-7 所示。

图 4-7 :**SF 之间的距离**

需要注意的是，帧与帧之间的 idle 符号数量可能有轻微的变化，取决于使用的参考振荡器，对于这种情况如何发生，在协议中应该有详细定义，同时协议也要定义如何去除 idle 字符。我们可以在协议中简单的定义如下：发送器的职责是在传输线路中插入足够多的 idle 字符；而接收器的职责是从接收数据流中去除所有 idle 字符、SF 字符和 EF 字符。

以上是定义一个简单协议所要涉及的内容。但是如何实现此协议呢？大部分的工作由启用 8b/10b 的 MGT 来完成。我们需要在接口处添加一些自定义的逻辑，但是不需要有处理器和其它的软件，甚至不需要有复杂的状态机。

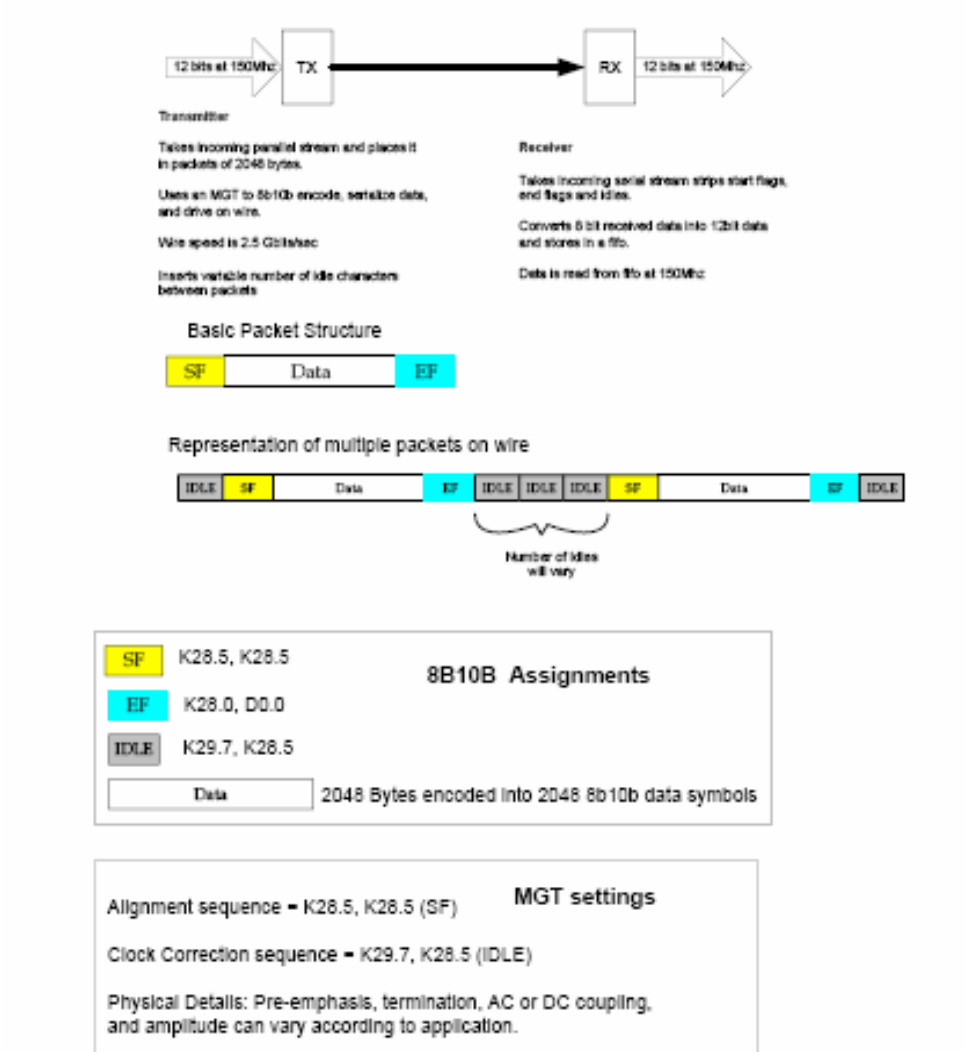


图 4-8 :**两板间传送 1.8Gb/s 数据的简单协议**

实现此协议的 MGT 和自定义硬件结构框图如图 4-9 所示：

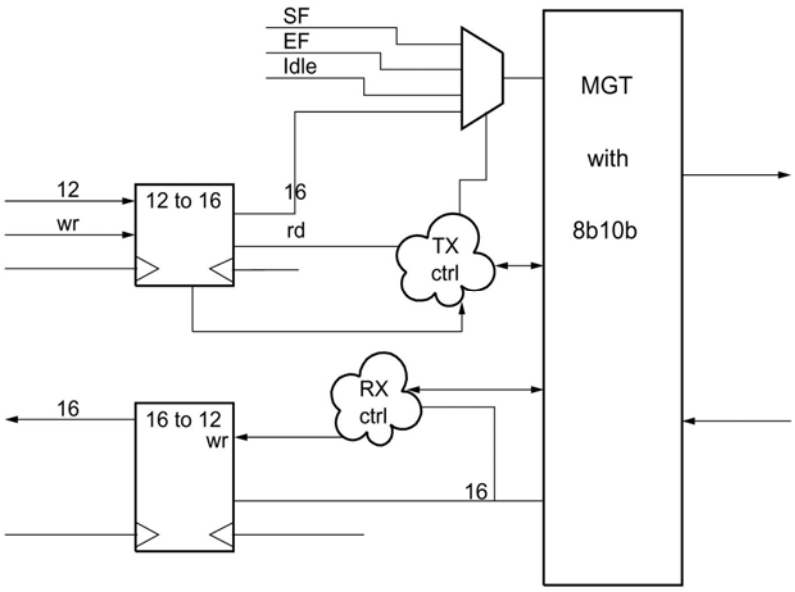


图 4-9：**使用 MGT 的自定义硬件**

有时候，一个很简单的自定义协议可能正是你所需要的。

信号完整性

为了满足完整性，信号必须是可靠的（即可重复和可预测的）。我们需要知道要达到的目标。信号必须是真实的或纯净的且未受干扰的。信号必须保持其纯净形式，而不能受其它信号干扰（串扰）或由于环境反射信号的叠加而引起衰减。所以我们现在需要考虑三个因素以保证信号的完整性：阻抗、功率和屏蔽性。

为确保良好的信号完整性，信号必须是可靠的，即可重复和可预测的。

阻抗

实现信号完整性的第一步就是在差分传输线上传送这些信号。根据通常的定义，传输线都有一定的固定阻抗。实际上，阻抗值并不是恒定的，而是变化的。这个问题在下面的几种情况下尤其突出：信号由一层转移到另一层时，信号遇到元件的焊盘时，或信号通过连接器或电缆时。当运行在千兆位级速率范围内，些许的阻抗增加都会是潜在的问题。千兆位级链路需要使用阻抗无限制通道，否则其无法工作。

TDR: 时域反射测量法

我们需要对传输通道进行模拟，并在布线之前使用 CAD 的信号完整性工具来最终确定连接器和电缆。我们在获取初始原型时，需要用时域反射测量法（TDR）来检验通道的阻抗。100 欧姆和 50 欧姆的传输线是最常用的传输线。部分收发器可以适配两种传输线，而部分收发器可能只能支持其中的一种。对于 10Gb/s 范围的应用，50 欧姆显然是最通常的选择。如果收发器同时支持 100 欧姆和 50 欧姆，那么连接器和电缆的选择问题应当慎重考虑。

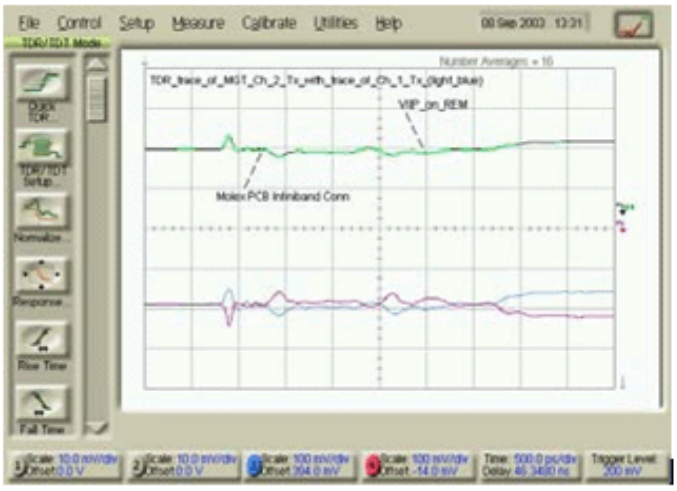


图 4-10 : **DCA屏幕拷贝**

电源

电源传送也是使用千兆位级收发器时需要考虑的重要因素。多数的 MGT 都需要由多个电源供电。

典型的电源包括：

- RX 模拟电源
- TX 模拟电源
- 模拟地
- RX 终端电压
- TX 终端电压
- 数字电源
- 数字地

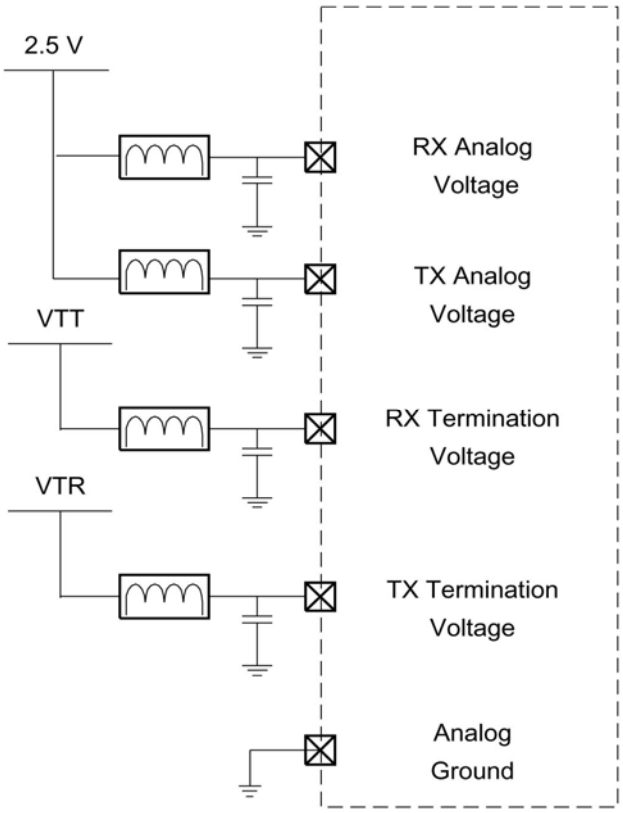


图 4-11： **MGT 电源的滤波电路**

所有的模拟发送和接收电源以及相关的模拟地必须是极其干净的，这一点是十分重要的。所以，MGT 制造商通常都会使用特定电路。所以，至少要求每个电压值都有各自的模拟电压校准器（如果不是每个 MGT 都有各自独立的校准器），并且要求使用无源的电源滤波器（由一个电容和一个铁氧体磁珠组成）。

铁氧体磁珠在低频时阻抗很小，而在高频时有很高的阻抗（如图 4-12 所示）。

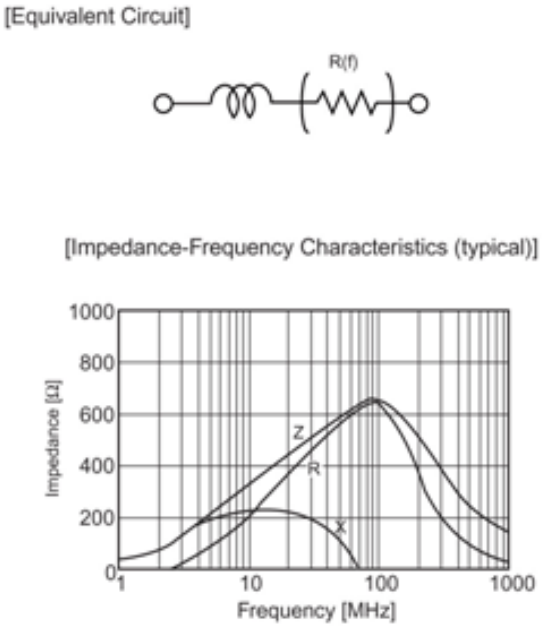


图 4-12 : **等价电路和频率特性**

铁氧体磁珠和电容的特性十分重要，通常制造商都会给出具体的推荐值。图 4-13 和图 4-14 给出了从芯片文档中截取的部分资料。

GHz Range	For Standard	BLM15HG601SN1	600±25%	1000±40%	200
		BLM15HG102SN1	1000±25%	1400±40%	100
		BLM18HG471SN1	470±25%	600 (Typ.)	200
		BLM18HG601SN1	600±25%	700 (Typ.)	
		BLM18HG102SN1	1000±25%	1000 (Typ.)	100
	For High Speed Signal	BLM18HB121SN1	120±25%	500±40%	200
		BLM18HB221SN1	220±25%	1100±40%	100
		BLM18HB331SN1	330±25%	1600±40%	50
		BLM15HD601SN1	600±25%	1400±40%	100
		BLM15HD102SN1	1000±25%	2000±40%	50
		BLM18HD471SN1	470±25%	1000 (Typ.)	100
		BLM18HD601SN1	600±25%	1200 (Typ.)	
		BLM18HD102SN1	1000±25%	1700 (Typ.)	50
	For Digital Interface	BLM18HK331SN1	330±25%	400±40%	200
		BLM18HK471SN1	470±25%	600±40%	
		BLM18HK601SN1	600±25%	700±40%	100
		BLM18HK102SN1	1000±25%	1200±40%	50
	For Standard (Low DC Resistance Type)	BLM15EG121SN1	120±25%	145 (Typ.)	1500*
		BLM15EG221SN1	220±25%	270 (Typ.)	700*
		BLM18EG101TN1	100±25%	140 (Typ.)	2000*
		BLM18EG121SN1	120±25%	145 (Typ.)	2000*
		BLM18EG221TN1	220±25%	300 (Typ.)	1000
		BLM18EG331TN1	330±25%	450 (Typ.)	500
		BLM18EG391TN1	390±25%	520 (Typ.)	500
		BLM18EG471SN1	470±25%	550 (Typ.)	500
		BLM18EG601SN1	600±25%	700 (Typ.)	500
		BLM18GG471SN1	470±25%	1800±30%	100

图 4-13:**部分芯片资料示例**

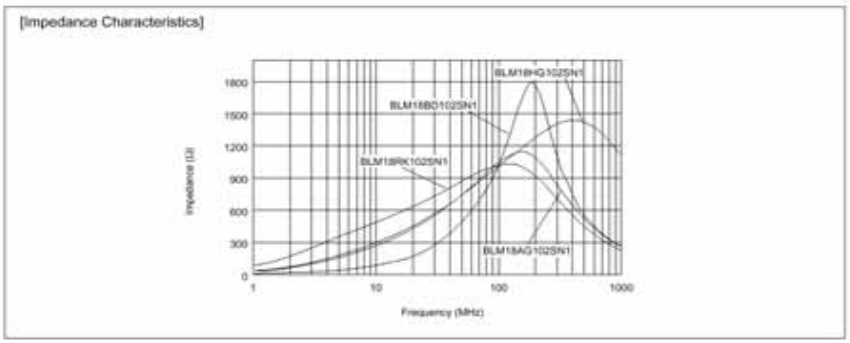


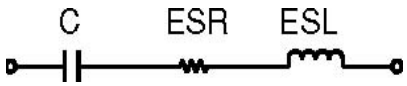
图4-14: **阻抗特性图示例**

某些 MGT（尤其是使用倒装封装的芯片）将电容包含在封装的内部，此时通常只需要铁氧体磁珠。如果制造商建议使用特定电路，则通常最好遵从其建议。原因之一是，在公共部分配置了多个 MGT 的情况下，通常只需要一个单独的线性调整器即可。滤波器电路可以防止电源噪声进入 MGT，同时它还可以防止来自某个 MGT 的噪声滤进其它 MGT。此滤波器既是输入滤波器也是输出滤波器。有时制造商会基于自己所需的输出滤波性能，在输入滤波器和输出滤波器性能间做出折衷。

ESL：等效电感

ESR：等效电阻

在设计时，除了模拟电源，数字电源也有需要特别考虑的事项。MGT 的数字电源通常是器件中所有数字逻辑的共同电源。由于使用了很多的开关电路，所以旁路的要求是很严格。但是，由于工作速度很高，我们不能仅仅插入一些电容就宣告完成旁路。这种方法在多年前被采用，那么为什么现在不采用呢？如果我们可以找到完美的电容（没有电感值或电阻值），板上的布线和过孔都是完美的（没有电感值或电阻值），并且封装等等也都是完美的，那么这种方法还是可以工作的。但是随着开关切换频率的提高以及所需电流的增大，以前可以忽略的 ESR 和 ESL 现在必须考虑。



实际的电容

图 4-15 开关电路

电源分配和旁路网络的目的是保证在变化电流的情况下能够提供正确的电压。旁路电路必须根据不同应用的具体要求进行设计。一种分析的方法是查看电源系统的阻抗和相关的频率。图 4-16 给出了标准应用中推荐使用的针对三个电容的频率响应。

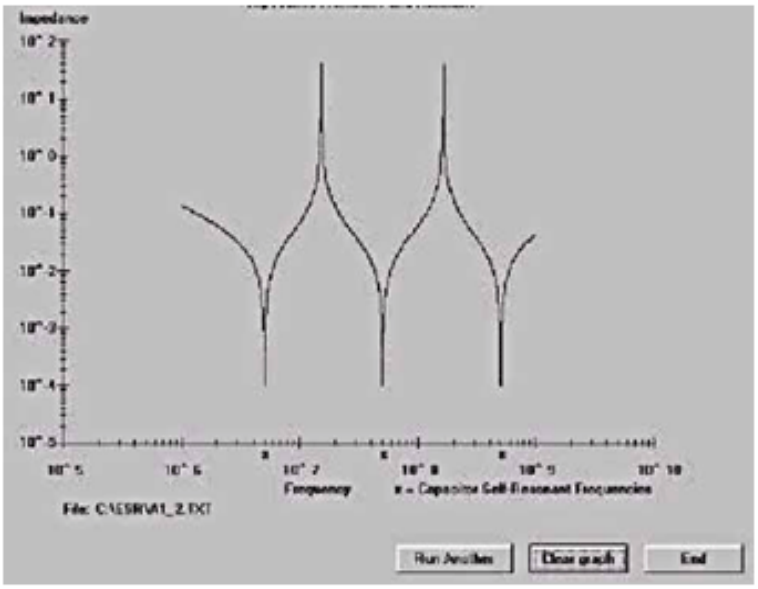


图 4-16 :**合理选择的电容的阻抗-频率图**

有两个问题需要注意，一个问题是某些频率范围会出现大阻抗峰值。如果我们的系统正好需要工作在这些频率范围内，就会有问题。所以旁路电路设计时应当确保这些峰值不会出现在我们的特定设计中，可以通过使用不同的电容来满足这个要求（图 4-17）。

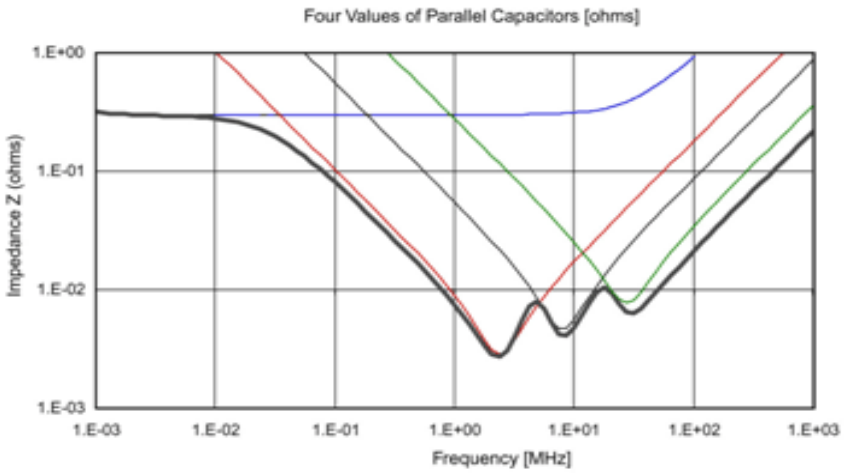


图 4-17:**合理选择的电容的阻抗-频率图**

另一个问题出现在高频范围，随着频率的增大，越来越难找到电容可以覆盖此频率范围，直至不可能找到这样的电容。随着电容值的减小，相关的杂散电感和封装的电阻值并不相应改变，所以频率响应也不会发生太大变化。为了在高速情况下实现较好的电源分配，我们需要利用电源层和地层来建构我们自己的电容。为了更有效的达到我们的目的，通常需要使用相邻的电源层和地层。典型的叠层结构如图 4-18 所示。

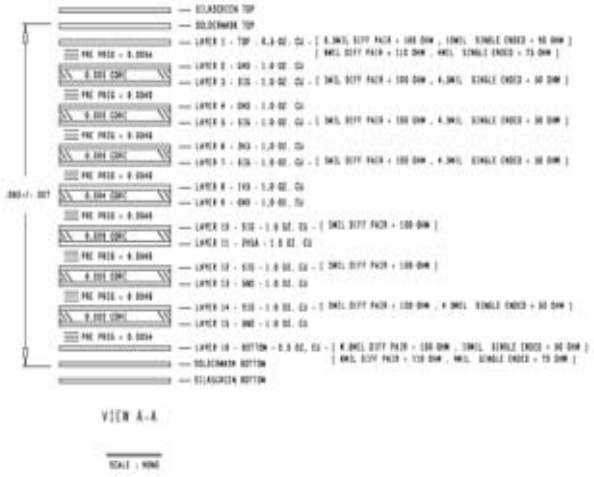


图 4-18 : **典型的叠层结构**

旁路需要考虑的另一重要方面是电容的放置。一般的规则是，电容值越大则其放置要求越不严格。若电容值较小，则电容应该尽可能靠近电源和地的引脚。可以采用的一种方法是将不用的通用 IO 的走线和过孔移除，从而为旁路电容腾出空间。如果在 FPGA 内使用 MGT，则通常采用上述的方法。如图 4-19 所示。

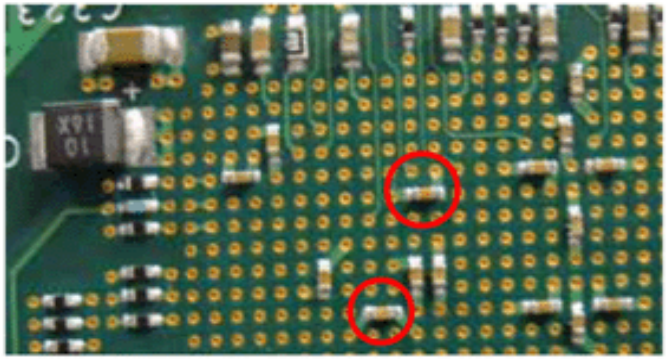


图 4-19: **移除走线和过孔**

屏蔽

任何千兆位级信号都必须是孤立的，以防止发生干扰或者被其它信号所干扰，无论信号是在板上，还是电缆中或是通过连接器。一方面，可以通过隔离和屏蔽连接器及电缆来实现。此外在 PCB 板上，千兆位级信号必须和其他信号分隔开，而且不同层的并行走线必须用电源层或者地层分隔开。

板、连接器和电缆

在设计使用千兆位级串行数据流系统时，元件的选择和板的设计都是很严格的。不合适的连接器、勉强合格的叠层结构或者错误的 PCB 材料都可能完全毁掉 MGT 工程。

印刷电路板设计

设计用于千兆位级操作的 PCB 对于最好的 PCB 设计者来说也会是一个挑战。差分线路必须匹配，阻抗受限的差分线路的几何形状必须随着层数的增加而相应变化，电源分配也必须严格分析。因为可能存在成千上万的独立的设计折衷和决定，所以全面列出所面临的问题可能会有所帮助。主要包括如下的几个方面：

- 材料选择
- 叠层结构/板厚度
- 电源层和地层
- 差分线路对
- 差分线路的宽度和间隔
- 过孔
- 线路对之间的间隔
- 电源布局

材料选择

FR-4 成为标准的板材料已经有很多年，同时其他不少可供选择的低耗材料也变得更容易获得了。大体的原则是：若线路长小于 20inch 且速度小于等于 3.125 Gb/s，那么 FR-4 是可以接受的。如果我们需要更长的线路或者更高的速度，则我们应该认真考虑选用高速材料，例如 ROGERS 3450。

叠层结构/板厚度

一旦我们选定了材料，下一步就是制定一个总体的叠层结构设计。信号层数是固定的，叠层结构是可以改变的，但是我们在设计过程中应谨记所使用的叠层结构。同时不要忘了尽量让电源层和地层靠近以改善旁路的效果。

电源层和地层

我们需要考虑如何分配那些特殊的模拟电压值。所以我们可能会考虑为每个模拟电压值各自独立分配一层。隔离和滤波作为千兆位级信号参考层的地层可能会是个不错的主意。同时，在速度低于吉比特的信号区域，我们可以考虑省去数字电源层。

差分线路对

为了达到最好的效果，差分线路对应当紧密耦合并且接近匹配。线路长度匹配是极为必要的。在 FR-4 中，100mil（inch 的十分之一）的线路距离差会导致差分信号间有大约 18 皮秒的差异。这个偏差量足以引起问题。而且，就算我们使用普通的布线方法从一个 BGA 连接到另一个 BGA 时，十分之一英寸听起来都是很多的，但是往往差分线路对之间的长度差值可以达到 300-400mil。所以如果 PCB 工具中有自动线路匹配，我们最好使用它进行匹配。总体来说，我们希望差分线路对之间的长度差为 50mil 或者更小。

差分线路宽度和间隔

每种特定叠层结构都需要独自设计差分线路宽度和间隔。PCB 板制作厂家可以是一个非常有用的资源，但是我们要确保他们知道正在做什么。在一些发表的指导资料中，不建议让 PCB 制作厂家来进行这些计算。我们需要确保厂家使用 *filed solver* 工具来表示紧密耦合的线路对的宽度和间隔。注意有一种技术我们绝对不能使用，即只选定几何结构而让 PCB 板制作厂家通过欠蚀刻和过蚀刻来调整阻抗。如果我们自己有自己的域解决软件并且可以熟练使用，那样会更好些。

几何结构示例如图 4-20 和图 4-21 所示：

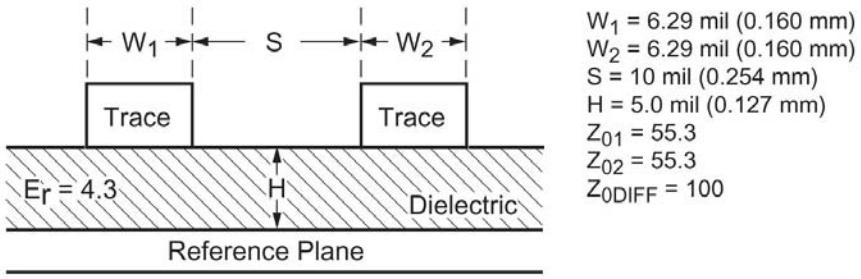


图4-20：**几何结构示例:微带线边沿耦合差分线路对**

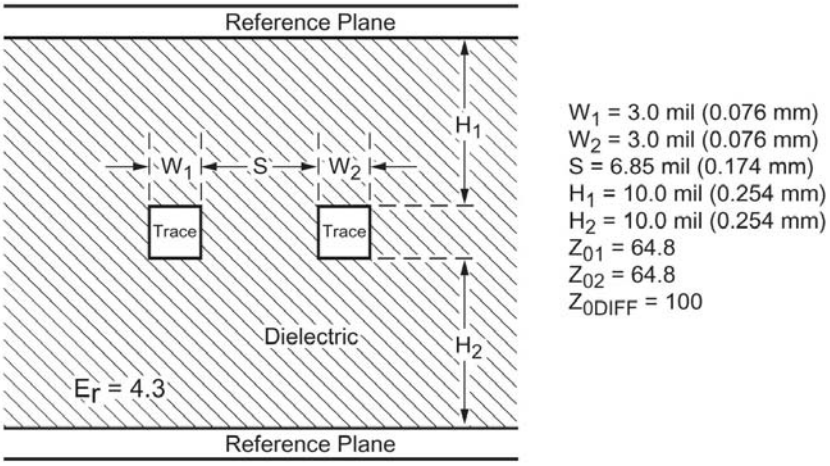


图4-21：**几何结构示例:带状线边沿耦合差分线路对**

过孔

千兆位级信号差分线路应当尽可能避免改变走线层。如果跨层传输是必须的，那么需要特别小心。首先，必须提供一个完整的返回路径。所以我们必须把层 A 的参考层和层 B 的参考层耦合在一起。最理想的情况是两个参考层都是地层。在这种情况下，返回路径可以通过在转层过孔附近放置另一个连接两个参考层的过孔来实现。图 4-22 给出这种技术的示意图。

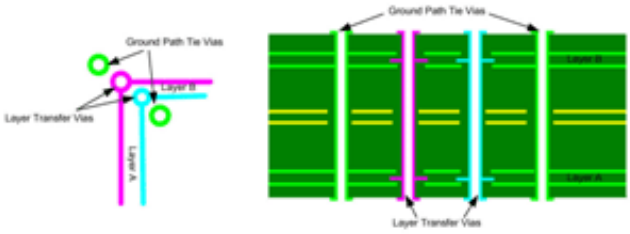


图 4-22 ：**参考层均为地层时的过孔**

如果参考层是不同的（一个是地层，另一个是电源层），则需要在离过孔尽可能近的地方放置 0.01µF 的电容来连接两个参考层。如图 4-23 所示。

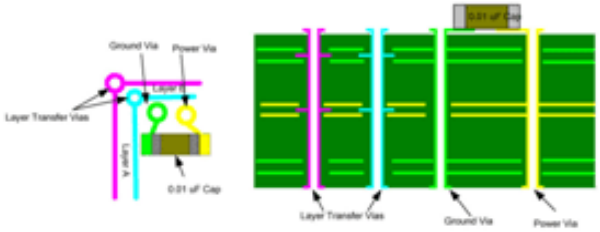


图 4-23 ： **参考层不同时的过孔**

过孔的另一个问题是过孔意味着短截线。显然，我们知道在传输线中引入短截线并不是个好主意（如图 4-24 所示）。

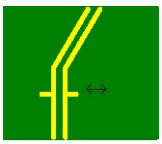


图 4-24 ： **传输线**

如果一个过孔将信号从内部发送到顶层，或者将信号发送到底层，则过孔中未使用的部分将会成为短截线。避免短截线问题的一种技术称作后钻孔技术，完成金属镀层之后，未使用的过孔部分通过钻孔除去（如图 4-25 中所示，其钻孔位处在过孔的低端）。

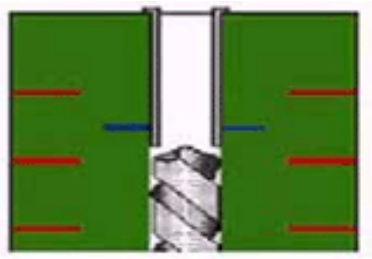


图 4-25 ： **后钻孔过程**

所有速率超过5Gb/s的设计都需要认真考虑进行了后钻孔的过孔（见图4-26）。

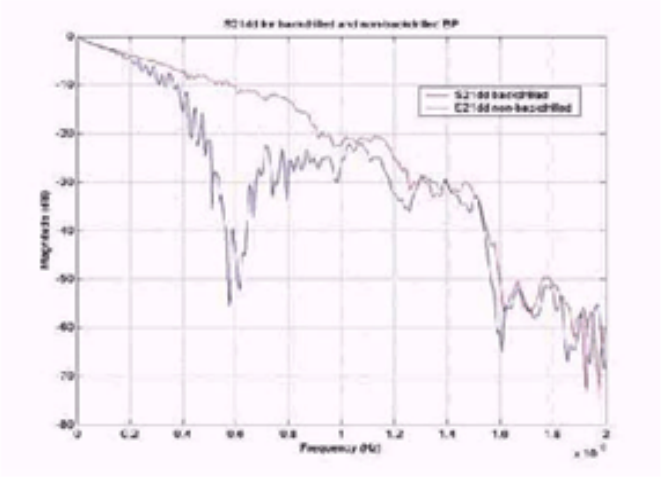


图 4-26： **后钻孔的过孔和没有后钻孔的过孔**

线路对间的间隔

差分线路对之间以及差分线路和其他线路之间都要保持一定的距离，这一点是很重要的。通常的规则是：相邻线路对间的距离至少要 5 倍于线路对中两线的距离（图 4-27）。

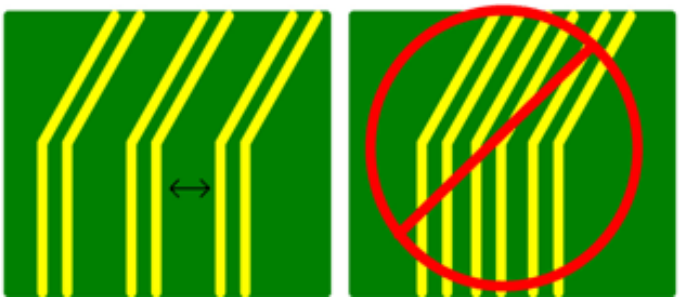


图 4-27 ： **线路对之间的间隔**

线路对之间的地层保护

另一种技术是通过同差分线路并行布置保护地线来屏蔽信号。使用过孔把保护地层连接回参考层，并且和线路保持平行，可以改善这种屏蔽方法的效果（图 4-28）。

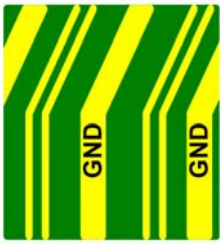


图4-28： **线路对之间的地线保护**

电源布局

许多讨论 MGT 电源供应的问题中都会涉及到板极布局。用于模拟电源滤波的铁氧体磁珠和电容的放置必须仔细考虑，应该参照电源引脚以及信号线来放置（如图 4-30）。

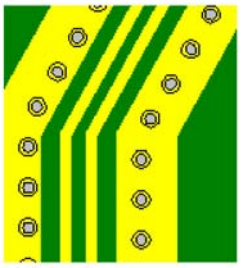


图 4-29： **铁氧体磁珠的放置**

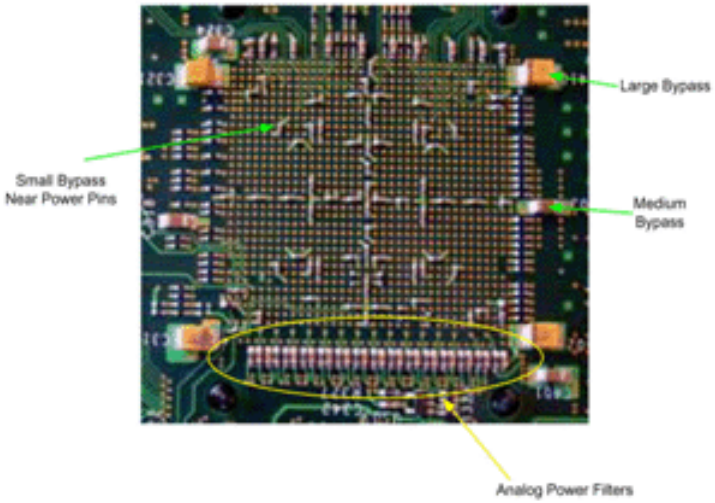


图 4-30 ： **电源布局**

连接器的选择

只有高速连接器才能用于千兆位级信号。和线路上的其他部分一样，高速连接器也有受限的阻抗。由于连接器的阻抗不可能像 PCB 线一样连续，所以使用高速连接器会比一般的连接器要好得多（图 4-31）。早期的高速连接器的设计同时适用于差分信号和单端信号。近来的高速连接器是都为差分信号专门设计的。下面给出了几个高速连接器的例子：

- Gbx
- VHDM-HSD
- VHDM
- HDM
- High Density Plus
- Z-PACK HM-Zd
- Z-PACK HS3

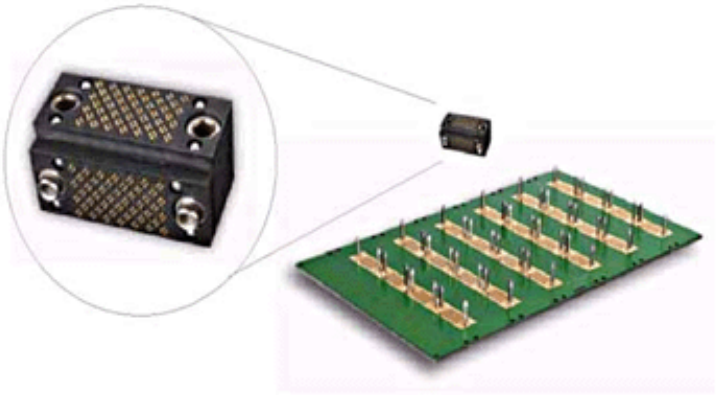


图 4-31： **高密度 10Gb/s 铜制互连系统**

如果设计中采用预设的协议或总线，则连接器的选择可以参考相应的标准。否则，除了选择一般连接器时需考虑的信号数、密度和尺寸等问题，还要考虑下面几个问题：

- 带宽
- 屏蔽
- 差分线路对
- 最高边沿速度

带宽

选择时应考虑连接器的速度规格以及其曾经成功应用的速度。许多早期的千兆位连接器最初设计用在 1 或 2Gb/s，但是最终都可以广泛应用在 3Gb/s（如图 4-32）。

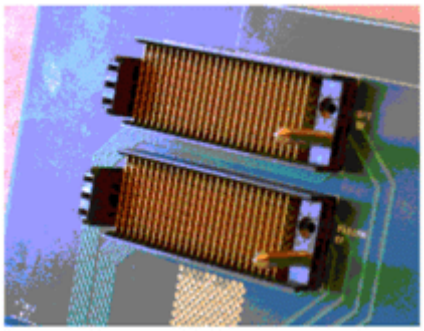


图 4-32： **高速 2G 连接器**

屏蔽性

选择时应考虑信号如何屏蔽其他信号以及外部的影响。在连接器的侧面，有许多屏蔽问题需要考虑。

差分线路对

选择时应考虑连接器是为差分信号专门设计的，还是只能兼容差分信号。

最高边沿速度

我们应当如何考虑最高的边沿速度呢？如果进入连接器的信号边沿速度太快，则连接器内部很可能会出现串扰。所以，我们应该明确知道连接器所能支持的最高速度以及我们需要的工作速度。

电缆的选择

如果在传统应用中使用 box-to-box 链路，那么我们需要选择电缆/连接器时序。首先我们需要考虑信号要传输的距离，以确定铜线是否能够支持这么长的传输距离，还是只能使用光纤。如果距离小于 20 米而且速度低于 6Gb/s，则铜线是可以使用的。

在千兆位级应用中常用的一种电缆是 Infiniband 电缆（如图 4-33）。此电缆最初是为 2.5Gb/s 的 Infiniband 应用而设计的，经过简单修改后就可用于光纤通道、CX4（10G 以太网）和其他一些用途。Infiniband 电缆可以有 1、4 或者 12 对线。

图 4-33： **Infiniband 电缆**

另一种有趣的电缆选择方案是使用电缆组合，它被设计成用于插入背板型的连接器（如图 4-34 和图 4-35）。这些电缆组合可以用在机箱内部，而且许多电缆组合都包含 EMI 屏蔽，所以可以用于 box-to-box 链路。

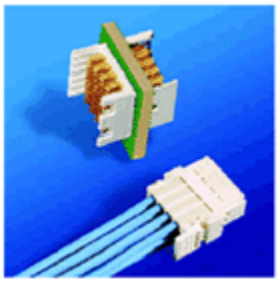


图 4-34 ：**背板电缆组合**

图 4-35： **更多的背板电缆组合**

许多其他电缆都正在被研究以适用于千兆位级链路，包括同轴电缆和我们熟悉的 5 类双绞线。

仿真

仿真在任何一个成功的 MGT 工程中都是很重要的部分。设计的模拟部分和数字部分都需要进行仿真。

模拟部分

多数的数字设计人员自大学以来从没考虑过进行模拟仿真。为何在我们的千兆位链路中需要进行模拟仿真呢？模拟仿真并不简单而且显然不便宜。但是，有人说过：“模拟仿真是保证链路正常工作，而所需重新设计次数最少的唯一方法。”。

不进行模拟仿真而设计出高速链路是可能的，但是在设计过程中往往会有数块失败的设计板。现代的模拟 EDA 工具可以对板上的差分传输线以及由过孔引起的不连续性进行建模。如果我们加入一个连接器模型，则我们可以通过仿真在制板之前察看其 TDR 情况。加入一个 MGT 收发器模型，则我们可以通过仿真来察看收发器的眼图。如果板不符合要求，则改变其布局后再重新尝试。

- 常用的模拟仿真工具包括：
- 信号完整性分析器
- SPICE 仿真器
- 电源完整性分析器
- 设计工具包

信号完整性（SI）分析器

信号完整性分析器常常是 PCB 布局工具的一个可选组件。信号完整性分析器可以分析 PCB 布局的信号完整性。通常分析器还允许通过添加连接器和电缆模型来进行多 PCB 系统的分析。另一个很有用的功能是分析器支持 IC 模型，特别是千兆位级的收发器。在 SI 分析器中使用有源电路模型时通常还需要模拟电路仿真工具（analog circuit simulator，SPICE）。除了 SPICE 模型外，SI 工具通常还支持 IBIS 模型和 s 参数。Allegro PCB SI 和 Mentor HyperLynx_GHZ 是两种全功能的 SI 工具。许多低端的 PCB 板工具也有 SI 分析选项。但是一般来说，低端的工具对于 Gb/s 的速度是不能胜任的，不过将来可能会有所改善。

SPICE 仿真器

在 MGT 模拟仿真和分析中，SPICE 仿真器的主要作用是作为 SI 分析工具的行为模型引擎。通常 MGT 供应商会以 SPICE 模型的形式提供行为模型，但是由于 SPICE 模型可以从本质上很好地描述电路，所以大部分人都采用加密的 SPICE 模型。这些 SPICE 模型通常需要使用为 IC 开发专门设计的高端 SPICE 工具。

SPICE 模型：基于文字描述的电路行为描述。十分精确，而且能够展示电路结构的详细资料。

S-参数：基于文字的描述，可以描述频率很高的电路、板上走线或者连接器。最初用在微波设计中，现在 s-参数常用于对高速板和连接器组合进行更有效的建模。s-参数描述在传输线中微波传输的散射和反射。

这样的工具包括 Synopsys 的 H-SPICE 和 Mentor 的 ICX/Edo。这些高端的 SPICE 往往是很昂贵的。将来低端的 SPICE 工具可能也会有加密能力。虽然 SPICE 是个强大的工具，但是还是有许多它不能解决的问题，不过若把信号完整性分析工具和 SPICE 结合起来使用会有更好的效果。注意旧版本的 P-SPICE 不能进行这样的分析。

电源完整性工具

电源完整性工具用于帮助设计电源传送（旁路、滤波等等）系统。许多信号完整性分析工具中都附带电源完整性工具。对应于 SI 工具为信号提供的功能，电源完整性工具可以为电源系统提供同样类型的功能。某些工具可以帮助解决电源系统中的部分问题，它们的功能不是很强大但是更能满足要求，例如 UltraCAD 的 ESR 和 Bypass Capacitor Calculator（帮助选择电容以满足具体旁路要求）。

IBIS 模型：基于文字的电路行为描述。在 1GHz 以下足够精确。不提供电路结构的具体资料。

设计工具包

通常 SERDES 制造商和信号完整性工具开发商会共同合作，并提供相关资料以加快模拟仿真过程。工具包中通常包括一个建立好的并且可以立即使用的常用项目。我们可以在 SI 工具中打开这个项目，从中学习这些工具的使用并了解 SERDES 的功能。一旦熟悉了一般概念，我们就可以用自己的设计代替这些预设计板和连接器。部分工具会有陡峭的学习曲线，这一点是十分有用的。

只有选定 SERDES 厂商之后，我们才能选取相应的模拟仿真工具。通常仿真模型或设计工具包只适用于一种特定的工具集。随着时间的推移，这个问题可能会有所改善，但现在我们应该记住这个问题。

数字部分

千兆位级链路的模拟仿真需求让我们进入了一个崭新的 EDA 工具世界，而其数字部分的影响则要小得多。尽管如此，数字仿真时还是有几个事项需要考虑的。

首先，许多 MGT 行为模型都是以特殊的加密形式出现的。这些模型都是复杂的内核，而且都是非常有价值的知识产权（intellectual property，IP）核。所以，厂商为了保护他们的知识产权，通常只会以 IP-safe 的格式发布这些模型。最流行的格式称作 smart 模型或者 swift。通常，模型经过加密后，仿真器可以读取模型但是用户则不能。模型内部的节点和层次对于用户来说是不可见的；用户只能看见模型的输入和输出。

smart 模型和 swift 会出现 2 个问题。第一，仿真器必须支持这种模型（一些通常用于 FPGA 仿真的低端工具并不支持此模型）。第二，我们需要启用仿真工具才能使用这些模型，这通常也要花费一些精力。

MGT 数字仿真的另一个问题是仿真速度。通常的数字电路大多数工作在 100-300MHz 的范围。我们需要把仿真的时标调整到几个纳秒。但是如果我们添加一个 MGT 的线速度模型，我们的信号速度就会比之前最快的信号还要快 20 倍以上（图 4-36）。

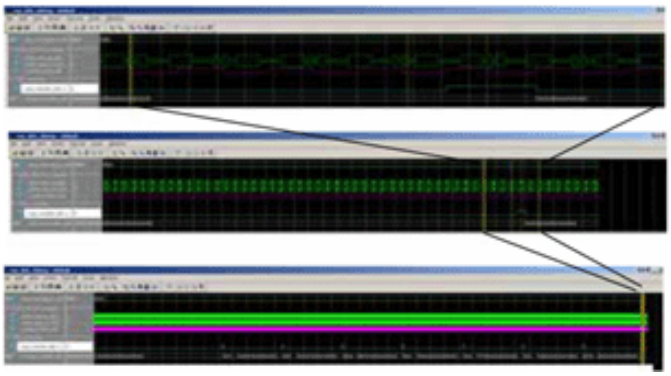
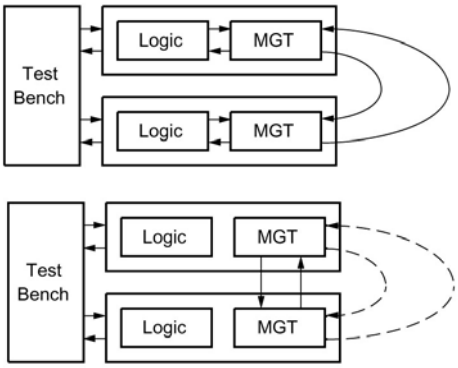


图 4-36： **MGT 仿真**

仿真的时标（time scale）必须不断调整，这会使事情变得更慢。甚至在屏幕上重绘这些信号也会显著变慢。我们可以通过下面几项措施来减小影响。首先我们必须清楚地认识到速度必将会是一个问题，并且围绕着“这些高速信号是设计的一部分”这个事实来不断优化我们的仿真。另一件必须要记住的事情是许多 MGT 模型都有一个并行的输入和输出端口。如果我们在大多数的测试台中使用这些措施，并创建一个实际运行在全数据速率的小型测试组件，则 MGT 对全局验证时间的影响将会大大减小。这种方法的框图如图 4-37 所示。



含有千兆位信号的全功能仿真，用于检测和 MGT 相关的功能。例如模拟参考时钟的细微差别以检查时钟修正功能。

并行接口代替了千兆位串行链路。用于更高速的仿真，但是功能较少。此模型用于多数的接口逻辑检验。

图 4-37： **两种不同测试台的框图**

数字仿真的最后一条建议

我们应当确保我们的测试台可以处理由于时钟不同而引起的问题，例如检查时钟修正和通道绑定（如果有使用的话）的变化。当工程进入原型实验室后，时钟修正问题会最先导致工程的停滞。

测试和测量

在工程的原型阶段，基于 MGT 测试和测量的考虑，需要使用一些专门的测试设备以确保我们的工程象所预期的那样运转。

采样示波器和数字通信分析器

在千兆位应用调试中所需要的最重要的设备可能就是采样数字示波器（或者就是个示波器）。采样示波器和普通的模拟示波器、数字存储型示波器稍微有点不同。模拟示波器的工作原理是：直接将待研究的信号提供给电子束的 Y 方向，电子束不停地横跨 CRT 显示器进行扫描，并在显示器上留下定义实际信号形状的路径。数字存储型示波器对输入的信号进行数字采样，保存各采样点的数值，最后使用这些采样点数据在显示器上重建信号。采样示波器同样将输入的信息进行数字化处理后保存。

采样示波器：将信息数字化并保存。如果信号的速度超出了 AD 转换器所能支持的范围，则示波器在每个周期内只采样很少的样点。通过逐次移动采样，示波器可以获取足够的信息以代表重复性的信号。

数字存储型示波器：将输入信号转化为数字采样并保存，最后使用这些采样数据在显示器上重建信号。

采样示波器和模拟示波器、数字存储型示波器不同的地方在于：采样示波器可以用于分析速度极高的信号。为了采样速度高于 AD 转换器工作速度的信号，采样示波器每个周期只采样很少的样点。逐次移动采样使其可以获取足够的信息以代表重复性的信号。如果信号重复性很强（例如 0 和 1 交替信号），则实际的波形就可以呈现出来。大多数情况下，采样示波器显示的是信号的眼图。除了信号输入，采样示波器还有一个时钟输入。这个时钟是示波器采样时的参考信号。输入的时钟的频率通常是线路速率或者线路速率的 1/n。通常，线路速率的 1/20 是可以接受的。图 4-38 给出了一种数字采样示波器（digital sampling oscilloscope DSO），以及在测试单元（unit under test，UUT）的数据和时钟输入。

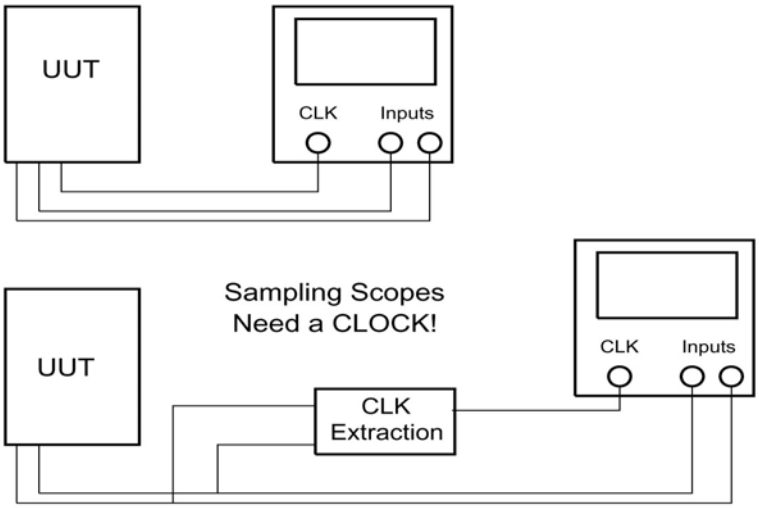


图 4-38 ：**UUT 数据和时钟输入的数字采样示波器**

为了应付极高速度的信号，采样示波器在 ADC 之前没有使用衰减器或放大器。这意味着相对于其他示波器而言，采样示波器输入的电压范围相当有限。采样示波器中也没有使用保护二极管，因为二极管会引起信号失真。DSO 的输入还对过压和静电放电相当敏感。

DCA：数字通信分析器；包含有采样示波器，同时还提供很多其他功能。

采样示波器通常是数字通信分析器（DCA）系统中的一个功能模块。DCA 中配置了采样示波器，同时还提供其他的一些功能。DCA 中通常集成了其他功能，多数是通过添加模块来实现的。常用的可选模块包括时钟恢复模块，该模块从输入的比特流中提取出低抖动的时钟，可以作为采样时钟。时域反射（TDR）模块可以用于测量阻抗，它也是个常用的可选模块。

时域反射

当我们得到原型时，需要做的第一件事就是到实验室用 TDR 来检验我们的传输通道。TDR 可以帮助我们检查传输通道的阻抗变化。通道越平缓，则问题就越少。TDR 的工作原理是：传送一个脉冲到传输通道中并测量反射回来的信号。通过测量，可以确定阻抗不连续的位置和激烈程度。TDR 模块和 DSO 模块可以联合工作，即 TDT 方法。TDT 查看的是在另一端接收到的脉冲而不是反射回来的脉冲。TDT 可以用于检查线路长度失配（图 4-39 和图 4-40）。

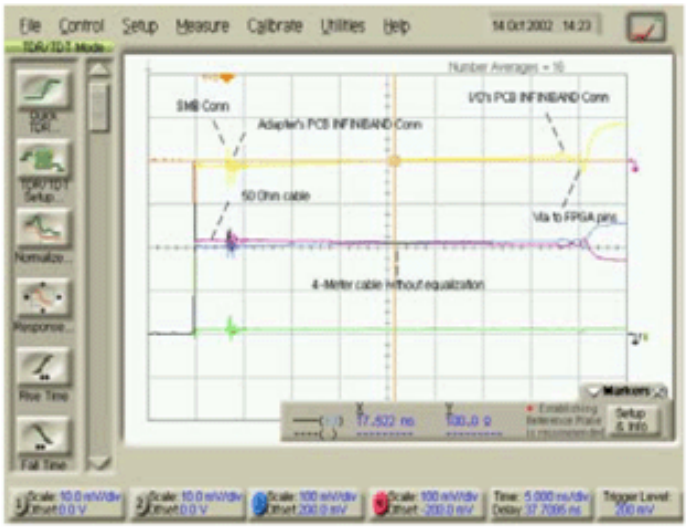


图 4-39 ： **例 1：TDR/TDT 屏幕拷贝**

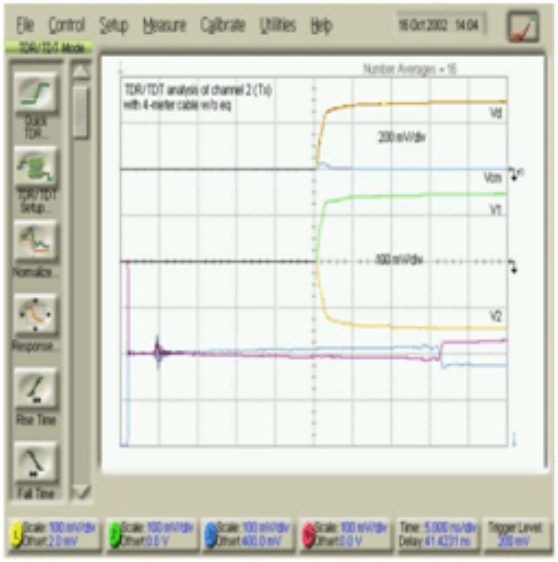


图 4-40：**例 2：TDR/TDT 屏幕拷贝**

眼图

上面我们已经多次提及眼图。在谈及高速串行信号流时不可能不涉及到眼图。眼图是采样示波器工作方式的自然结果。对随机信号进行相同时长的快速取样，采样后的图形不断叠加在一起，即得到眼图（图 4-41）。

眼图：数字采样示波器上常见的波形。眼图用于指示信号的质量。眼图可以刻画信号的抖动、阻抗匹配和振幅。

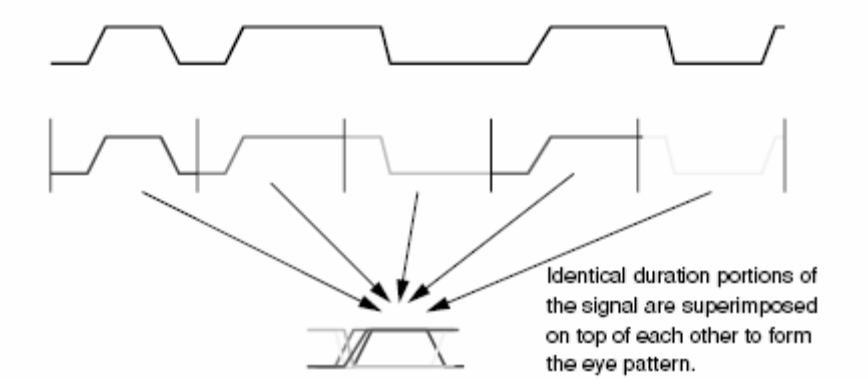


图 4-41：**眼图结构**

如果我们开启针对随机比特流的持续性选项，则使用数字存储型示波器同样也可以观察到眼图。随着信号质量的下降，图形会变得像凝视的眼睛一样，所以称之为眼图。

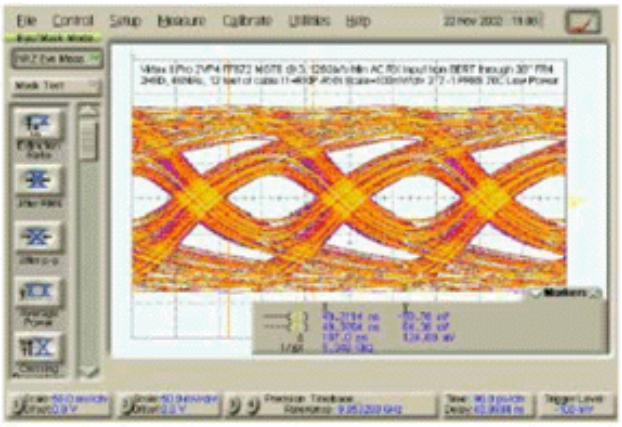


图 4-42 ：**FR-R 线路传输的眼图**

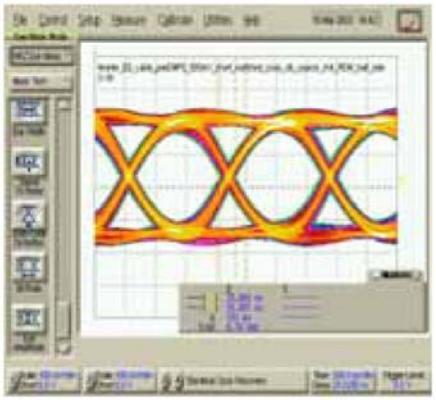


图 4-43 ：**眼图的颜色**

通过分析眼图，我们可以得到信号和传输通道的很多信息。眼图的高度和宽度同接收器接收该信号的能力相关。通常接收器会发布其相应的眼图掩板。如果信号的眼图可以包含在掩板内，则接收器可以检测到该信号。

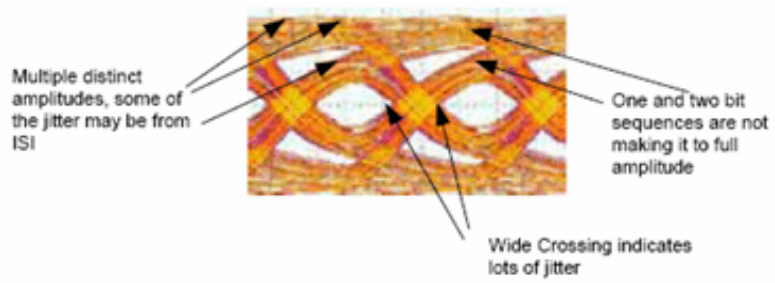


图 4-44： **眼图描绘**

眼图中的交叉宽度代表着系统的抖动。其他的一些细节，例如太多或太少的预加重以及阻抗失配，即差分线路对两侧的阻抗不对称，都可以由眼图的不规则性得到（图 4-45）。

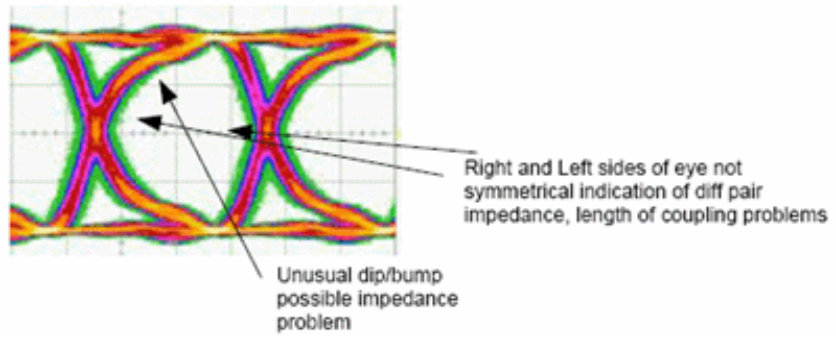


图 4-45： **眼图不规则性**

眼图的另一个重要方面是颜色。许多现代的设备使用颜色来标记强度。颜色越暗或者越“热”，则此处的数据样点就越多。在图 4-46 的眼图中，橙色代表较多的数据样点，绿色则代表较少的数据样点。

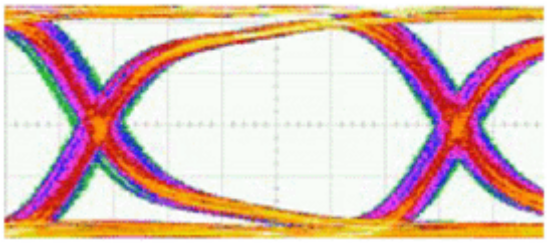
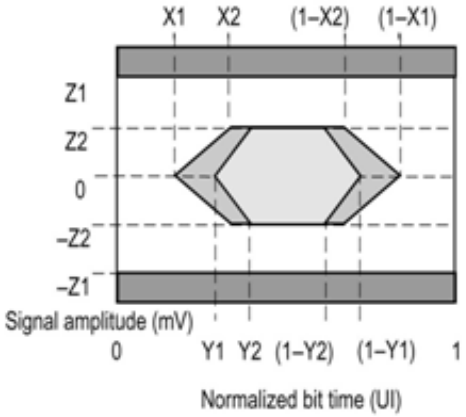


图 4-46 ： **眼图的颜色**

和眼图相关的另一个术语是眼图掩板。它用于描述保证接收器正确接收信号所需要的眼图的睁开度。眼图掩板可能和下图（图 4-47）是类似的。



Data rate (Gbps)	X1 (UI)	X2 (UI)	Y1 (UI)	Y2 (UI)	Z1 (mV)	Z2 (mV)
3.125 (XAUI)	0.180	0.305	0.300	0.425	800	100
2.5 (Infiniband)	0.205	0.380	0.325	0.500	800	87.5
1.25 (802.3z)	0.231	0.356	0.375	0.500	800	100

图 4-47 ：**眼图掩板图形**

抖动

上面我们已经用到了术语“抖动”，并且我们知道它和眼图中不同眼睛相交叉的宽度是相关的。当我们在调试一个千兆位级链路时，我们需要很好的理解什么是抖动，抖动来自何处，以及抖动会造成什么影响。

抖动：完美的过零交叉和实际的过零交叉之间的区别。

从数学上来讲，我们可以将抖动看作信号持续时间内的一个变量。例如：如果我们有一个正弦时钟，则我们可以定义一个完美的零抖动时钟如下：

$$\cos(w(t))$$

则抖动信号可以描述如下：

$$\cos(w(t) + j(t))$$

其中 j(t)用于描述抖动，抖动通常分为两类：确定性抖动和随机抖动。

- 随机抖动：随机抖动是由差分模式和普通模式的随机噪声引起的，例如电源噪声和热噪声。随机抖动也称作 rj、RJ 或者不确定性抖动。
- 确定性抖动：由具体眼图或者事件确定的抖动。引起确定性抖动的源包括：不对称的上升/下降时间、字符交叉干扰、电源馈电、振荡器、来自其他信号的串扰。通常也简称为 DJ 或 dj。

如果千兆位级链路不能工作，最可能的原因就是过量的抖动。我们需要了解接收器输入信号的抖动程度，并和接收器的规格相比较。同时，我们最好还要检查一下振幅和眼图的高度。如果这几项都是可以接收的，则很可能不是抖动的问题。

发生器和误码检测器

实验室中其他的有用工具包括样本、时钟发生器和误码检测器。低抖动的时钟发生器对于开发测试样本和检测误码率是十分有用的。下面是一种可能采用的方法（图 4-48）。

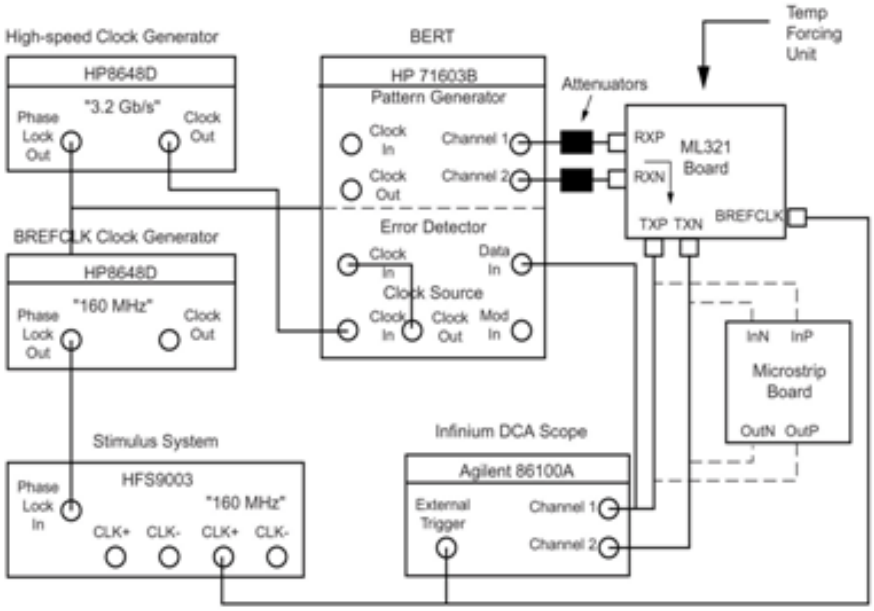


图 4-48 ： **发生器和误码检测器**

和串行链路相关的另一类图表是浴缸形曲线（图 4-49）。

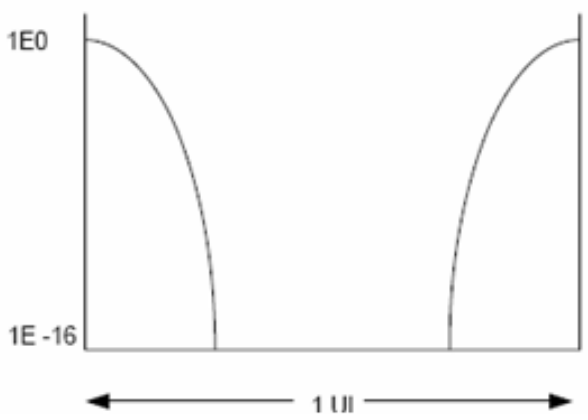
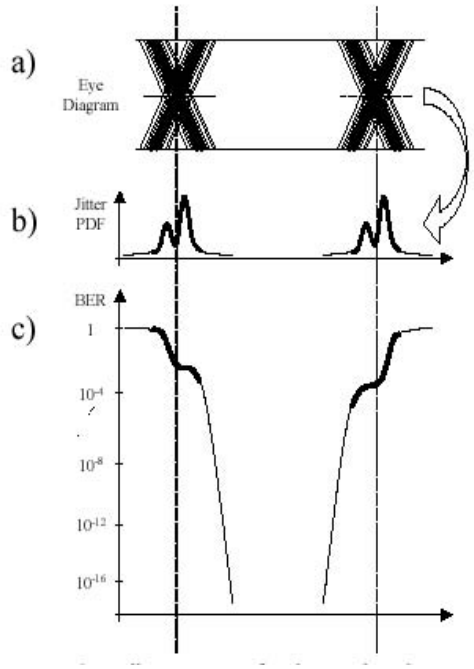


图 4-49 ： **浴缸形曲线**

浴缸形曲线用于显示单元间隔内和采样位置相关的误码率。曲线的底部并不是零，但是相当接近于零，通常在 10^{-12} 到 10^{-16} 的范围内。曲线的上限是 100%误码率或者 1。误码检测器通常用来产生浴缸形曲线。在某些情况下，眼图和浴缸形曲线之间存在一定的关系，下图是由一家叫做Wavecrest 的BER测试商提供的（图 4-50）。



眼图、抖动概率密度和浴缸形曲线间的关系

a） 眼图指示数据传输的极限

b） 抖动概率密度（粗线）和 TailFit™的推断（细线）

c） 从抖动概率密度得到的浴缸形曲线（粗线）和 TailFit™的推断（细线）

图 4-50 ： **波峰**

在一定的条件下，上述的关系才成立。如果传输通道从外部连接到含有有源均衡器的 MGT，在这种情况下试图通过眼图来推导浴缸形曲线或者 BER 都是不正确的。

配置所需的设备

当我们的 PC 板已从 PCB 制作厂家拿回来，并且我们的原型实验室已配好装备，我们可能会发现我们自己在疑惑下一步该做些什么。每个工程都有不同的要求，下面是一些建议：

1. 我们需要在传输通道上运行 TDR。TDR 可以告诉我们许多事情，包括：PCB 板满足阻抗要求的程度，由过孔引起的阻抗增大量，连接器连接线等等。在第一遍检查过程中，我们必须沿着通道仔细检查每一部分。同时，我们要特别注意那些干扰源。检查过程中，连接器和过孔等都可以看作是干扰源。核实观测眼图所处的位置，这可以通过把手指按在过孔或者焊盘上来实现。由于手指的电容会稍微改变总阻抗值，所以我们通常可以在显示器上看出变化。比较实际 TDR 和使用信号完整性分析工具得到的仿真值，可能也会给我们提供有价值的参考信息。
2. 接下来要做的事情是察看尽可能靠近接收器的引脚处的眼图。将得到的眼图和接收器的规格相比较。抖动是否在容限内？眼图的高度/振幅是否正确？如果一切正常，即可以开始测试链路。如果有问题，则我们需要检查如下的调试提示。

千兆位级调试提示

调试千兆位级设计有时可能也会是一个挑战。一些可用的调试提示涵盖了如下几个方面：

- 低的信号幅度
- 低的眼图高度
- 过量的抖动
- 使用 SI 工具
- 最终的调试提示

接收器引脚处的低信号幅度

如果信号幅度太低，我们可能需要加大输出驱动器的电压。如果问题不是出现在输出驱动器，则应该是因为信号在板和连接器上损耗过大；如果是这种情况，我们往往会意识到一开始就应该进行模拟仿真，因为现在我们将面临板的重设计。在确定需要重设计之前，首先我们要确保不是由于测试启动问题或者制造缺陷而导致的信号幅度过低。检查所有的连线、器件号、器件值等等。我们还需要沿着传输通道检查不同点处的振幅，从而大概了解损耗发生的位置。

低的眼图高度

如果整体幅度是足够高的而眼图高度却很低，则有些位置的幅度是足够高的，而其他位置则是不够高的。这通常是由于通道或者发送器在部分频率下的增益/衰减不同而造成的。通常我们要首先检查预加重设置。错误的预加重设置可以导致单比特传输都无法达到足够的高度。如果我们有均衡器或者在通道中使用了平衡电缆，则应当检查并确保均衡器或平衡电缆处于正确状态。如果均衡是可调的，则我们可以尝试调节一下均衡。

接收的过量抖动

链路不工作最可能的问题就是过量抖动。低的眼图高度常常伴随着抖动问题，所以解决低眼图高度的各种建议此处同样适用。如果过量抖动不是预加重问题引起的，也不是由内部信号完整性的相关抖动或者均衡器的设置引起的，此时我们需要查找抖动的其他来源。部分可能的来源如下：

- 电源问题、馈电和噪声
- 串扰
- 不对称的上升/下降时间
- 不匹配差分线路引起的通常问题
- 振荡器漂移或者抖动

确定抖动来源也是很困难的。我们可能需要使用最先进的技术，很好的记录来系统地拆分问题并解决。而且我们需要熟悉测试设备的每项功能。一些新的高端 DCA 有极其强大的抖动诊断功能可以帮助我们找到问题的根源。

把问题分解并解决是一个很好的起点。首先，沿着通道获取不同点的眼图。从何处开始抖动开始增加？一旦我们知道问题出现在通道的什么位置，我们就可以集中精力来研究这部分通道及其可能的抖动来源。

例如，如果抖动在发送器处是可以接受的，但是经过第一个连接器之后就变得不可接受，则可能的抖动来源包括：

- 位于连接器和发送器之间的其他板上信号所引起的串扰。
- 板上电源层引起的串扰
- 与连接器或者过孔相关的阻抗颠簸引起的反射
- 连接器内部其他信号引起的串扰

讨论可能的抖动来源是简单的，但是证实某种抖动来源确实是问题（或者部分问题）的起因往往很困难。针对串扰问题，要注意禁用那些可能的串扰源。

使用 EDA SI 工具

EDA SI 工具是非常有用的资源。仿真中的一个小问题在实际情况下是否会变得更大？我们现在是否已经更了解模型？如果是的话，则修改后再次进行仿真可能会在同样的位置重现问题。

有一件事我们没有太多讨论过，即如何将板上的信号输入到 DCA。DCA 连接器通常是 SNA 型同轴电缆，如果市场上有适配的电缆和连接器，我们可以购买这些适配器件。如果市场上没有，那么我们可能需要自己设计适配器件（图 4-51 和图 4-52）。

图 4-51： **1x Infiniband 到 SNA 的连接器**

图 4-52： **4x Infiniband 到 SNA 的连接器**

观察 PC 板上的信号是相当困难的。如果我们使用了 AC 耦合，我们通常可以把耦合电容焊接到一小段线上。之后再把电容的另一端焊接到焊盘上。

这种方法可能会有些凌乱，而且会毁掉一些原型。所以，在我们的计划中应该考虑因测试和检查而损坏的原型。甚至我们应当特别制作一些特定部分缺失的原型。

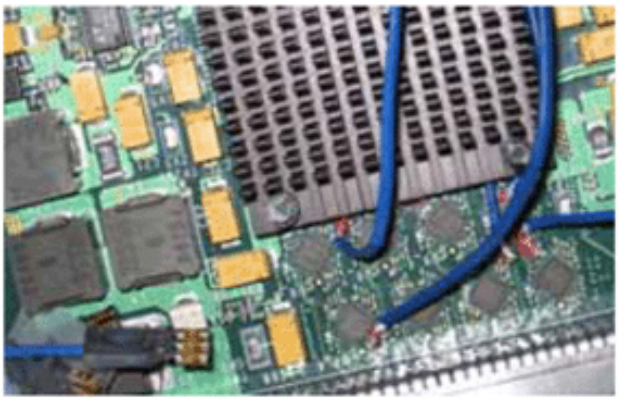


图 4-53： **接线的原型**

最后的调试提示

调试千兆位链路时还有最后一件事情必须谨记。链路有两个主要部分，物理部分和协议。上述的所有建议都是针对物理链路而言的，而且是基于协议层一切正常的假定，或是通过使用样本和时钟发生器将协议层从测试中忽略掉。数字仿真应当在制作原型之前已经解决掉大部分的协议缺陷，但是两者之间通常还是会有交迭。例如，如果在比特流中出现了错误，但是接收器的输入抖动看起来很严重，那么实际上可能是时钟修正问题。

互操作性

当我们制定一个特定标准时，我们应当注意要和其他产品相兼容。我们可能需要和此特定应用的先前版本以及使用其他 SERDES 厂家的旧版本相接口就可以。实际设计互操作能力时应当注意如下的几个事项。

协议层

当和使用“相同”协议的其他系统接口时，也可能出现问题。这些协议太过复杂，所以肯定会出现不同版本的协议解释。我们可以将设备交给独立的检验实验室，以确保我们对协议的解释是正确的。购买协议引擎设计或软件对互操作性设计都是有利的。如果我们要和一个定制协议相结合，则尽可能使用相同的源代码。

电气层

在物理层上，事情可能会简单一点。如果我们使用的是标准协议，多数的事情例如连接器、层次等都是定义好的。若是一个定制应用，则物理接口的详细信息可能还没有定义。我们必须记住三条：电平、耦合和终端匹配。

我们希望发送和接收发生在所接口设备的相同层次上。我们还需要决定如何进行耦合。如果 SERDES 是类似的，或者有相同的终端电压，则我们可以进行 DC 耦合。这样的话我们就可以省去 AC 耦合电容，也就不会遇到由 AC 耦合电容引起的一些问题。如果通常的 DC 耦合不能胜任，则 AC 耦合可能可以解决问题。

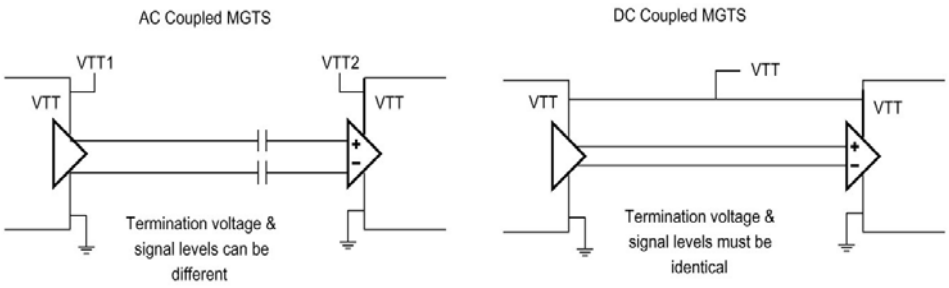


图 4-54 ： **AC 和 DC 耦合框图**

其他资源

至此，您可能已经准备好进行自己的设计，并准备好使用这种新技术。那么，您一定是一位在职设计工程师。您可能会很激动，但同时会有点担心。您也可能会问“我们的产量增加的够快吗？我们通常的板制作厂家能胜任吗？我们有足够的钱购买这些新设备吗？”。那么显然您是一个工程项目经理。如果您的位置处于两者之间，那么您可能是一个经验丰富的设计工程师并且关注此技术已经很长时间。别太过虑，这里有许多可用的帮助。

设计服务

和工程世界的多数事情一样，我们可以雇佣有经验的人为我们工作。如果我们考虑这个方式，那么我们一定要考虑开发工具和流程。设计公司是否有信号完整性工具？有哪几个工具？他们是否会使用我们的板极设计包？通常，一个优秀的设计顾问不仅能使我们快速启动第一个项目，而且在项目过程中还能够培训我们的队伍，这样他们就可以自己运作下一个项目。

测试中心

如果测试和测量设备的预算是一个大问题，那么可以考虑使用厂商或者第三方测试中心。由于设备十分昂贵，所以许多 SERDES 厂商都建立了相应的实验室，用户可以使用实验室进行检查或者调试硬件。这些实验室通常都有很好的设备，并且可能会有经验丰富的千兆位级工程师或技术人员为您提供帮助。某些第三方团体经营类似的实验室。还有第三种选择，不少引领技术潮流的大公司近期都开始向外短期出租其研究设备。这些实验室中有许多用于先进技术研究的设备，通常也配置有千兆位级实验所需要的所有设备。

开发平台

另一个开发资源是预设计的原型板。它们可以从许多 SERDES 厂商或者第三方团体获取。这些原型板可用于进行一些前期工作，可当作一个信号发生器，或用于让我们的队伍学习新的 DCA，通常其定价合理而且十分有用。

第5章

Xilinx—您的设计合作伙伴

Xilinx提供的其他设计资源

串行I/O设计的考虑事项

我们已经看到了一个成功的千兆位级收发器(MGT)项目的历史、技术与实际设计的考虑因素。这些信息提供了开展高效串行输入/输出(I/O)设计所必需的技术背景。此外，Xilinx还提供了帮助您在I/O设计中采用千兆位级串行技术的其他信息。

一站式串行I/0门户网

Xilinx提供了功能强大的技术网站，通过单击链接即可获得关于知识产权核、参考设计、白皮书、信号完整性(SI)、PCB设计工具、电路板、教育培训服务等各方面信息。

保存到书签中的URL地址为：

http://www.xilinx.com/cn/serialsolution/.

图5-1 ： 高速串行解决方案网站

信号完整性中心

我们已经了解到对于高速串行设计，信号完整性考虑因素在设计过程中扮演着重要角色。Xilinx还提供了另一处出色的在线资源，称为“信号完整性中心”(http://www.xilinx.com/cn/signalintegrity) 。在这里，您可以轻松浏览到信号完整性的基本原理、PCB设计工具、计算器和估计器、电源、PDS设计考虑因素、散热设计指导方面的论文及文章，以及其他信号完整性资料。

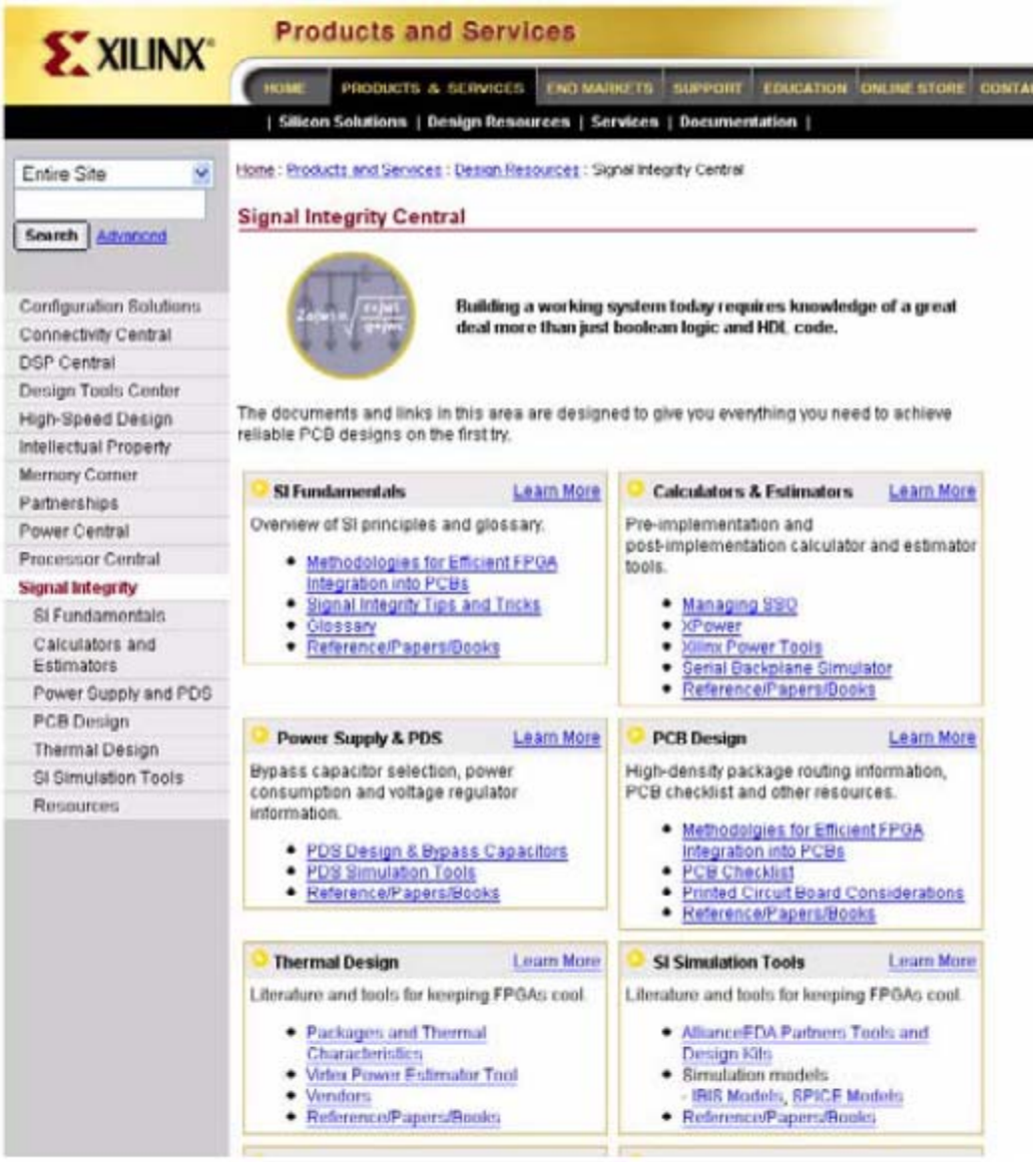


图5-2 ： 信号完整性网站

另一组珍贵的资料是Xilinx与全球信号处理领域最杰出的权威专家-Howard Johnson博士共同制作的一套DVD。这套DVD包括“信号完整性技术”与“RocketIO

收发器的损耗预算”两张DVD。每张DVD长度约为45分钟，包含Dr. Johnson提供的介绍性评论、高速串行设计相关专题的详细技术讨论及生动的“实验室”产品演示。这些内容使观众能够对高速系统性能的关键问题有一个清晰的了解。这套DVD可以在Xilinx网上商店http://www.xilinx.com/cn/store/dvd订购，售价19.95美元。

图5-3 ： Xilinx网上商店—DVD 订购

其他参考资料

为了支持您的设计任务，我们提供了附录，里面包含了其他一些设计信息：

附录A包含了基于FPGA的串行器/解串器(SERDES)的部分数据手册。它可以帮助您将所学的知识用于实际的串行器/解串器(SERDES)设计中。它包括结构框图与寄存器定义。请注意这项技术正处于快速发展中，因此请随时查阅网站上的最新信息。

附录B是完整的8b/10b列表

附录C是一份白皮书，将MGT链路与源同步链路进行了对比

附录D是技术词汇列表

Xilinx—强大的设计合作伙伴

Xilinx是千兆位级技术的领导厂商。Xilinx在1984年发明了现场可编程门阵列(FPGA), 1998年推出了Virtex™ FPGA，这是全球首款提供了100万个系统门电路的FPGA产品，从根本上重新定义了可编程逻辑技术。2000年，Xilinx推出了旗舰产品Virtex-II FPGA，树立了FPGA平台高密度、高性能逻辑的新标准。

为了满足对更大I/O带宽的需求， Xilinx在2002年推出其第三代Virtex产品：Virtex-II Pro。在这款产品中，Xilinx添加了3.125 Gb/s串行收发器和嵌入式IBM PowerPC™ 405处理器，作为标准的FPGA功能，从而再次对FPGA进行了重新定义。接下来，Xilinx收购了千兆位级串行技术的主要研发厂商Rocket-Chip公司，一举成为了千兆位级技术的先锋领导。

目前，Xilinx正在发售Virtex-II Pro FPGA（其收发器支持622 Mb/s -10.3125 Gb/s的传输速度）和Virtex-II Pro X FPGA（其收发器支持2.488 Gb/s -10.3125 Gb/s的传输速度），同时还准备发布Virtex-4 FX FPGA — 这款产品的收发器将达到前所未有的、超过目前所有产品的最宽速度范围（622 Mb/s-10.3125 Mb/s），并具有高级信道均衡性能（如决策回馈等化，DFE）、与主要串行I/O标准兼容等先进特性。在为Virtex-II Pro与Virtex-II Pro X FPGA提供了EasyPath™解决方案（表5-1）后，Xilinx还将针对Virtex-4 FPGA提供EasyPath™解决方案（表5-2），从而将FPGA的优势进一步扩展到大批量生产中。

表5−1：**Virtex-II Pro EasyPath解决方案**

特性/产品	**XC 2VP2**	**XC 2VP4**	**XC 2VP7**	**XC 2VP20**	**XC 2VPX20**	**XC 2VP30**	**XC 2VP40**	**XC 2VP50**	**XC 2VP70**	**XC 2VPX70**	**XC 2VP100**
EasyPath成本降低解决方案	-	-	-	-	-	**XCE 2VP30**	**XCE 2VP40**	**XCE 2VP50**	**XCE 2VP70**	**XCE2 VPEX70**	**XCE 2VP100**
逻辑单元	3,168	6,768	11,088	20,880	22,032	30,816	46,632	53,136	74,448	74,448	99,216
BRAM总数 (Kbits)	216	504	792	1,584	1,584	2,448	3,456	4,176	5,904	5,544	7,992
18x18 乘法器	12	28	44	88	88	136	192	232	328	308	444
数字时钟管理模块	4	4	4	8	8	8	8	8	8	8	12
配置(Mbits)	1.31	3.01	4.49	8.21	8.21	11.36	15.56	19.02	26.1	26.1	33.65
PowerPC处理器	0	1	1	2	1	2	2	2	2	2	2
最多可提供的3.125 Gb/s RocketIO收发器	4	4	8	8	0	8	12	16	20	0	
最多可提供的10.3125 Gb/s RocketIO收发器	0	0	0	0	8	0	0	0	0	20	
最多可提供的用户I/O	204	348	396	564	552	644	804	852	996	992	1164

表 5-2 ： Virtex-4 EasyPath解决方案

特性/产品	XC 4VFX12	XC 4VFX20	XC 4VFX40	XC 4VFX60	XC 4VFX100	XC 4VFX140
EasyPath成本降低解决方案	-	XCE 4VFX20	XCE 4VFX40	XCE 4VFX60	XCE 4VFX100	XCE 4VFX140
逻辑单元	12,312	19,224	41,904	56,880	94,896	142,128
BRAM总数(Kbits)	648	1,224	2,592	4,176	6,768	9,936
数字时钟管理器(DCM)	4	4	8	12	12	20
相位匹配时钟驱动器 (PMCD)	0	0	4	8	8	8
差分I/O对最大数	160	160	224	288	384	448
XtremeDSP Slice	32	32	48	128	160	192
PowerPC 处理器模块	1	1	2	2	2	2
10/100/1000以太网MAC块	2	2	4	4	4	4
RocketIO 串行收发器	0	8	12	16	20	24
配置存储器位	5,017,088	7,641,088	15,838,464	22,262,016	35,122,240	50,900,352
最大SelectIO数	320	320	448	576	768	896

Xilinx世界级的技术支持

Xilinx通过高品质的技术支持材料来支持其先进的芯片产品，这些材料包括广泛的知识产权核、参考设计、模拟电路模块、信号完整性（SI）设计套件、数字仿真的质量行为模型等。此外，Xilinx还提供了众多设计服务、开发平台以及最佳的FPGA实现工具。选择Xilinx作为您的合作伙伴，您可以确保您所有的设计需求都将获得可能的最佳产品和与技术支持。

Xilinx希望您能拥有愉快的串行I/O设计经历。我们撰写此书的重要原因之一就是想改变这样一种看法，即：串行I/O设计总是一件很困难的事。虽然串行I/O技术对于多数设计者而言是一项新技术，但它被广泛应用在高端电信与数据通信中已有相当长的时间，这充分证明了这种技术自身的高可靠性和低成本。

我们很想了解您是否喜欢本书，以及它是否能帮助您快速展开串行设计。请将您的反馈发送给作者：serialio@xilinx.com

附录 A

SERDES 示例资料
——RocketIO X 收发器概述

本章的内容摘自 RocketIO™ X 收发器的用户指南，相关的更新信息请参见 http://www.xilinx.com/cn/bvdocs/userguides/ug035.pdf

基本体系结构和性能

本指南中的定义，描述，以及建议适用于“Step 1”级器件。对于“Step 0” 级器件，请参见订正表再作特殊考虑。

RocketIO X 的结构框图如图 1-1 所示。Virtex™-II Pro X FPGA 内部配置了 8 到 20 个收发器模块，具体数目由器件的规模决定，如图表 1-1 所示。

表 1-1 各种器件的 RocketIO X 内核数量

器件	RocketIO X 内核数量
XC2VPX20	8
XC2VPX70	20

定义：

- **属性（Attribute）**：属性是配置 RocketIO X 收发器的控制参数。RocketIO X 属性包括原型端口（传统的控制和状态 I/O 端口），和收发器属性。属性用来控制收发器的数据带宽和编码规则，这些控制是以“软”参数的形式通过设计原型的调用来实现的。

- **GT10 设计原型（Primitive）**：设计原型是一组预定义的属性集合，包括数据率、编码类型、数据宽度等等。一个简单的设计原型调用，例如：OC-192 模式，只需要一步设

置即可将所有的相关属性定义为适配参数。

收发器模块每通道支持的比特率可以从 2.488Gb/s 到 10.3125Gb/s，涵盖了表 A-1-2（第 111 页）中列出各种通信标准所采用的具体比特率。在 **GT10** 设计原型中需要对具体的数据速率属性进行适当设置。

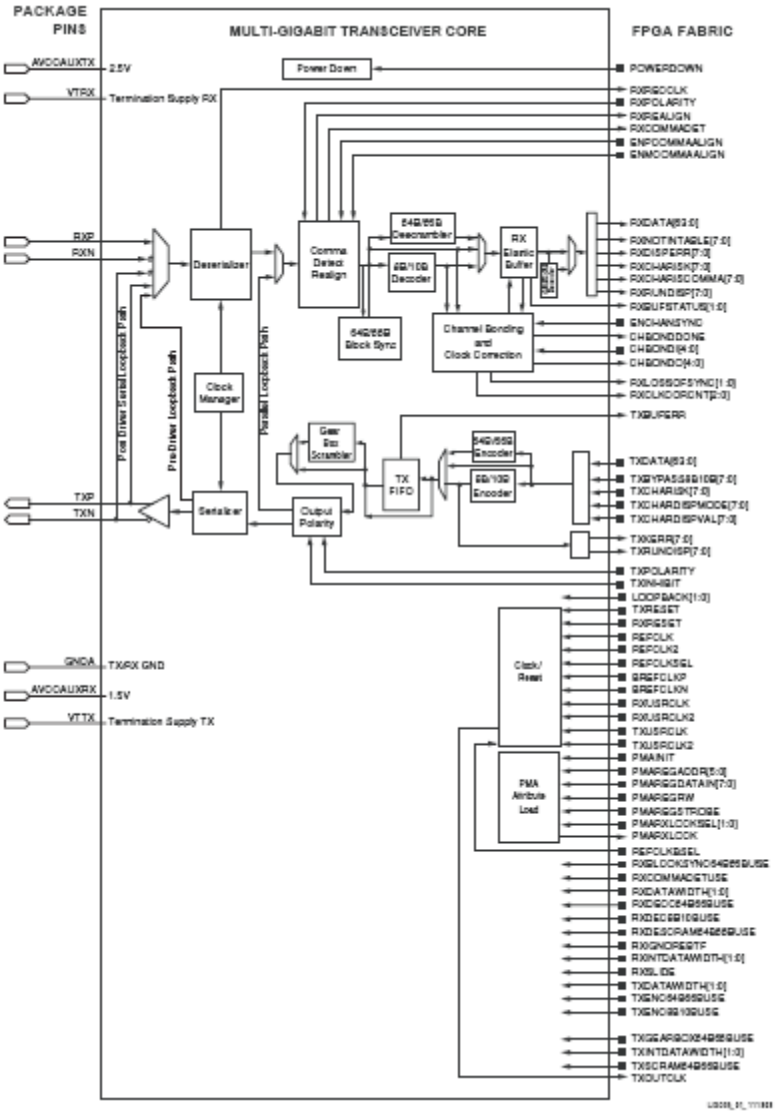


图1-1 ：RocketIO X 收发器结构框图

表1-2: RocketIO X 收发器支持的通信标准

模式	通道数(1)	I/O 比特率(Gb/s)
SONET OC-48	1	2.488
PCI Express	1, 2, 4, 8, 16	2.5
Infiniband	1, 4, 12	2.5
XAUI (10-Gigabit Ethernet)	4	3.125
XAUI (10-Gigabit Fibre Channel)	4	3.1875
SONET OC-192(2)	1	9.95328
Aurora (Xilinx protocol)	1, 2, 3, 4,...	2.488 – 10.3125
Custom Mode	1, 2, 3, 4,...	2.488 – 10.3125

注：

1. 一个通道意味着一个收发器。

2. 具体实现中的建议参见解决方案中的 19020 号记录.

表 1-3 列出了 RocketIO X 提供的收发器设计原型，这些设计原型可以直接将收发器的各属性设置为各通信协议（如表 1-2 中所列出）所采用的具体数值。各种通信协议可以选用不同的数据宽度，例如 1、2、4 bytes（低速率）和 4、8 bytes（高速率）。

表1-3 RocketIO X 收发器支持的设计原型

设计原型	描述	设计原型	描述
GT10_CUSTOM	用户自定义	GT10_XAUI_4	10GE XAUI 4-byte 数据通道
GT10_OC48_1	SONET OC-48, 1-byte 数据通道	GT10_AURORA_1	Xilinx protocol, 1-byte 数据通道
GT10_OC48_2	SONET OC-48, 2-byte 数据通道	GT10_AURORA_2	Xilinx protocol, 2-byte 数据通道
GT10_OC48_4	SONET OC-48, 4-byte 数据通道	GT10_AURORA_4	Xilinx protocol, 4-byte 数据通道
GT10_PCI_EXPRESS_1	PCI Express, 1-byte 数据通道	GT10_OC192_4	SONET OC-192, 4-byte 数据通道
GT10_PCI_EXPRESS_2	PCI Express, 2-byte 数据通道	GT10_OC192_8	SONET OC-192, 8-byte 数据通道
GT10_PCI_EXPRESS_4	PCI Express, 4-byte 数据通道	GT10_10GE_4	10-Gbit Ethernet, 4-byte 数据通道

GT10_INFINIBAND_1	Infiniband, 1-byte 数据通道	GT10_10GE_8	10-Gbit Ethernet, 8-byte data path
GT10_INFINIBAND_2	Infiniband, 2-byte 数据通道	GT10_10GFC_4	10-Gbit Fibre Channel, 4-byte 数据通道
GT10_INFINIBAND_4	Infiniband, 4-byte 数据通道	GT10_10GFC_8	10-Gbit Fibre Channel, 8-byte 数据通道
GT10_XAUI_1	10GE XAUI, 1-byte 数据通道	GT10_AURORAX_4	Xilinx 10G protocol, 4-byte 数据通道
GT10_XAUI_2	10GE XAUI, 2-byte 数据通道	GT10_AURORAX_8	Xilinx 10G protocol, 8-byte 数据通道

有三种方法可以配置 RocketIO X 收发器：

- 静态属性可以通过 HDL 代码中的属性进行设置。属性的使用具体参见“设计原型属性”，第 120 页。
- 属性的动态改变可以通过属性编程总线实现。
- 属性的动态改变也可以通过设计原型的端口实现。

RocketIO X 收发器由 PMA (Physical Media Attachment ,物理介质接入层)和 PCS(Physical Coding Sublayer，物理编码子层)组成。PMA 包括串行器/解串器(serializer/deserializer ，SERDES)，TX 和 RX 缓冲器，时钟发生器和时钟恢复电路。PCS 包括 8b/10b 编码器/解码器，64b/66b 编码器/解码器/扰码器/解扰器，以及用于支持通道绑定和时钟修正的弹性缓冲器。此时，我们可以回顾一下图 1-1，第 110 页，图中给出了 RocketIO X 收发器的总体框图和 FPGA 接口信号。

RocketIO X 收发器实例

针对不同的时钟机制，需要对参数进行相应修改，包括 USRCLK 和 USRCLK2 的时钟频率。GT10_CUSTOM 的数据和控制端口通常会使用最大的数据宽度。如果设计中不需要如此大的数据宽度，在 Verilog 设计中可以将没有使用的输入和输出线接地；在 VHDL 设计中则应将没有使用的输入线接地，输出线悬空。

HDL 代码范例

设计过程中可以使用结构化设计向导（Architecture Wizard）来创建实例模板。针对特定的应用，向导会自动生成代码和模板，并对模板属性进行相应设置。

可用端口

表 1-4 给出了所有设计原型的端口描述，RocketIO X 收发器设计原型包含有 72 个端口。差分串行数据端口（RXN, RXP, TXN, 和 TXP）直接和外部引脚相连接；其余的 68 个端口可以通过 FPGA 内部逻辑进行访问和控制。

表1-4 设计原型端口

端口	输入/输出（I/0）	端口大小	定义
BREFCLKNIN	I	1	来自BREFCLK引脚的差分BREFCLK 反向输入 。
BREFCLKPIN	I	1	来自BREFCLK引脚的差分BREFCLK 正向输入 。
CHBONDDONE	O	1	当信号为高时，表示接收器成功完成通道绑定。
CHBONDI[4:0]	I	5	通道绑定的级联控制信号，仅适用于“从”收发器。
CHBONDO[4:0]	O	5	通道绑定的级联控制信号，控制其他收发器进行通道绑定和时钟修正。
ENCHANSYNC	I	1	送给收发器的控制信号，使能收发器完成通道绑定功能。
ENMCOMMAALIGN	I	1	选择基于“－”comma字符的串行输入bit流对齐方式。当检测到“－”comma字符时，在重对齐的比特流中标记字节边界。
ENPCOMMAALIGN	I	1	选择基于“＋”comma字符的串行输入bit流对齐方式。当检测到“＋”comma字符时，在重对齐的比特流中重新标记字节边界。
LOOPBACK[1:0]	I	2	选择反馈测试模式，包括三种模式：内部并行环回、预驱动串行和后驱动串行环回测试。
PMAINIT	I	1	当信号从高变低时，从设计原型PMA_SPEED 中重载PMA系数，并复位PCS。.
PMAREGADDR[5:0]	I	6	PMA 设计原型总线地址，输入为异步信号。

表1-4 设计原型端口（续）

端口	输入/输出（I/O）	端口大小	定义
PMAREGDATAIN[7:0]	I	8	PMA 设计原型的数据输入，输入为异步信号。
PMAREGRW	I	1	PMA 设计原型的读/写控制信号，输入为异步信号。
PMAREGSTROBE	I	1	PMA 设计原型的总线选通信号，输入为异步信号。
PMARXLOCK	O	1	接收PLL锁定指示信号. 当 RX PLL 设置为“锁定数据”，此信号总为逻辑“1”
PMARXLOCKSEL[1:0]	I	2	选择接收PLL的锁定方式
POWERDOWN	I	1	当信号为高时，关闭收发器模块的接收和发送，注：输入为异步信号。
REFCLK	I	1	嵌入在模块内部的参考时钟输入。
REFCLK2	I	1	REFCLK的备选信号，可以通过REFCLKSEL来选择。
REFCLKBSEL	I	1	参考时钟的选择控制输入。如果信号为1，则选取BREFCLK作为参考时钟；如果信号为0，则选取REFCLK 或 REFCLK2作为参考时钟，取决于REFCLKSEL信号。
REFCLKSEL	I	1	REFCLK/REFCLK2的选择控制信号。如果信号为0，则选择REFCLK作为参考时钟；信号为1，则选择REFCLK2作为参考时钟
RXBUFSTATUS[1:0]	O	2	接收弹性缓冲器状态输出。用于指示接收FIFO指针，通道绑定偏移值和时钟修正状态。
RXBLOCKSYNC64B66BUSE	I	1	如果信号为1，则使用块同步；如果信号为0，则块同步逻辑被旁路。
RXCHARISCOMMA [7:0]	O	1, 2, 4, 8(1)	用于指示8b/10b解码器接收到的K28.0,K28.5,K28.7字符,和部分绑定comma的字符（取决于DEC_VALID_COMMA_ONLY的设定）

表1-4 设计原型端口（续）

端口	输入/输出（I/O）	端口大小	定义
RXCHARISK[7:0]	O	1, 2, 4, 8(1)	当8b/10b解码器处于使能状态时，若信号为1表示接收到的数据为“K”字符，包含在字节映射中。如果8b/10b解码器被设置成旁路，则表示10字节编码数据中的首位（比特“a”）。
RXCLKCORCNT[2:0]	O	3	用于指示时钟修正事件，通道绑定事件和FIFO指针的状态。状态与输入的RXDATA是同步的。
RXCOMMADET	O	1	用于指示接收到PCOMMA_10B_VALUE（如果ENPCOMMAALIGN信号为1）和/或MCOMMA_10B_VALUE（如果ENMCOMMAALIGN信号为1）所定义的符号。
RXCOMMADETUSE	I	1	如果信号为1，则启用comma字符检测功能；如果信号为 0，则旁路comma字符检测功能。
RXDATA[63:0]	O	8, 16, 32, 64(2)	接收到的1,2,4或8字节解码数据（8b/10b使能时）或编码数据（8b/10b被旁路）。
RXDATAWIDTH[1:0]	I	2	(00, 01, 10, 11)指示FPGA并行总线宽度。
RXDEC64B66BUSE	I	1	如果信号为1，则启用64b/66b解码器；如果信号为0，则旁路64b/66b解码器。
RXDEC8B10BUSE	I	1	如果信号为1，则启用8b/10b解码器；如果信号为0，则旁路8b/10b解码器。CLK_COR_8B10B_DE = RXDEC8B10BUSE
RXDESCRAM64B66BUSE	I	1	如果信号为1，则启用扰码器。如果信号为0，则禁用扰码器。
RXDISPERR[7:0]	O	1, 2, 4, 8(1)	如果8b/10b解码处于启用状态，则用于指示串行线上发生的不一致错误，包含在字节-映射序列中。
RXIGNOREBTF	I	1	如果信号为 1，则 64b/66b 解码器忽略块类型域（BTF），不进行错误检测和报告，而依原样直接发送块。如果信号为 0，则进行错误检测，未知的 BTF 会被标记为错误块。

表1-4 设计原型端口（续）

端口	输入/输出（I/O）	端口大小	定义
RXINTDATAWIDTH [1:0]	I	2	(00, 01, 10, 11)设置接收PCS的内部模式，即16-, 20-, 32-，或40-位。
RXLOSSOFSYNC [1:0]	O	2	第0位保持为0。第1位为0时表示发生了64b/66b块锁。
RXN	I	1	串行差分端口 (FPGA 外部)
RXNOTINTABLE [7:0]	O	1, 2, 4, 8(1)	用于指示解码数据的状态，当信号为0时表示数据为非法字符。应用于字节-映射序列。
RXP	I	1	串行差分端口 (FPGA 外部) 。
RXPOLARITY	I	1	用于设置RXN 和RXP的极性，和和TXPOLARITY类似。信号为0时表示正常极性，信号为1表示反极性。
RXREALIGN	O	1	从PMA输出的信号，用于指示基于comma字符检测的数据流的字节对齐。信号为1表示了字节对齐。
RXRECCLK	O	1	从数据流中提取，并经过分频的时钟信号。分频比取决于PMA_SPEED和/或PMA属性。
RXRESET	I	1	RX系统的同步复位，实现将接收弹性缓冲器归位，同时复位8b/10b解码器、comma字符检测、通道绑定、时钟修正逻辑以及其他内部接收寄存器。但是此操作并不复位接收PLL。
RXRUNDISP[7:0]	O	1, 2, 4, 8(1)	用于指示接收串行数据的运行不一致(0=负，1=正)。如果8b/10b 被旁路，则此端口仍为接收到的10位编码数据中的第2位（位“b”）。
RXSLIDE	I	1	用于使能检测块的1位移位，即实现了从低位到高位的递增。此端口信号必须首先为1之后转为0，并和RXUSRCKLK2保持同步。RXSLIDE信号在重新变为高电平之前，必须保持低电平至少2个时钟周期。

表1-4 设计原型端口（续）

端口	输入/输出（I/0）	端口大小	定义
RXUSRCLK	I	1	从DCM或BUFG引入的时钟信号，用于读取RX弹性缓冲器。此时钟信号还用于发送器和CHBONDI 及CHBONDO间的输入输出。RXUSRCLK和TXUSRCLK是类似的。RXUSRCLK和RXUSRCLK2必须保持180度的相位差。
RXUSRCLK2	I	1	从DCM输出的时钟信号，为接收器的数据提供时钟，同时也是收发器和FPGA结构之间状态的时钟。RXUSRCLK2和TXUSRCLK2是类似的。RXUSRCLK和RXUSRCLK2必须保持180度的相位差。
TXBUFERR	O	1	提供发送FIFO的状态。如果信号为1，则表示发生了上溢或下溢。当此信号为1时，只能通过设置TXRESET来使其复位。
TXBYPASS8B10B[7:0]	I	8	如果TXENC8B10BUSE=1且TXENC64B66BUSE=0（即使能8b/10b编码器而禁用64b/66b编码器），则TXBYPASS8B10B[7:0]的每一位分别控制相应的TXDATA字节的旁路。如果某一位为1，则禁用相应通道中数据的编码。如果TXENC8B10BUSE=0且TXENC64B66BUSE=1（即禁用8b/10b编码器且启用64b/66b编码器），TXBYPASS8B10B[2:0]用于附加的64b/66b编码器块旁路控制。TXBYPASS8B10B[7:3]在此种配置中是无关的。位[2:1]为用于块旁路操作的替代同步报头(SH[1:0])；对于用户希望旁路的每个块，位[0]设置为0。

表1-4 设计原型端口（续）

端口	输入/输出（I/O）	端口大小	定义
TXCHARDISPMODE [7:0]	I	1, 2, 4, 8(1)	如果启用8b/10b编码，则此总线决定采用哪种不一致模式进行发送。如果禁用8b/10b编码，则对于由字节映射确定的每个字节而言，此信号表示由10位编码TXDATA总线发送的第1位（位“a”）。如果TXENC8B10BUSE端口信号为0，则TXCHARDISPMODE的信号没有意义。
TXCHARDISPVAL [7:0]	I	1, 2, 4, 8(1)	如果使能8b/10b编码，则此总线决定采用哪种不一致类型进行发送。如果旁路了8b/10b编码，则对于由字节映射确定的每个字节而言，此信号表示由10位编码TXDATA总线发送的第2位（位“b”）。如果TXENC8B10BUSE端口信号为0，则TXCHARDISPMODE的信号没有意义。
TXCHARISK[7:0]	I	1, 2, 4, 8(1)	如果TXENC8B10BUSE = 1（即使能8b/10b编码器），则TXCHARISK[7:0]指示相应的字节通道中TXDATA字节的K-定义（1表示此字节为K字符，0表示此字节为数据字符）。如果TXENC64B66BUSE=1（使能64b/66b编码器），则TXCHARISK[3:0]指示TXDATA的块格式定义（1表示此字节为控制字符，0表示此字节为数据字符）。在此种配置中，TXCHARISK[7:4]是无关的。
TXDATA[63:0]	I	8, 16, 32, 64(2)	从FPGA来的待传送数据，可以是1，2，4或8字节，取决于所使用的设计原型。TXDATA[7:0]通常是第一个发送的字节。第一个字节的位置取决于所选用的TX数据通道宽度。
TXDATAWIDTH[1:0]	I	2	(00, 01, 10, 11) 指示FPGA的并行总线宽度。
TXENC64B66BUSE	I	1	如果信号为1，则使用64b/66b编码器。如果信号为0，则禁用64b/66b编码器。
TXENC8B10BUSE	I	1	如果信号为1，则使用8b/10b编码器。如果信号为0，则禁用8b/10b编码器。

表1-4 设计原型端口（续）

端口	输入/输出（I/O）	端口大小	定义
TXGEARBOX 64B66BUSE	I	1	如果信号为1，则启用64b/66b自适应同步器(Gearbox)。如果信号为1，则禁用64b/66b自适应同步器(Gearbox)。TXSCRAM64B66USE = TXGEARBOX64B66BUSE
TXINHIBIT	I	1	如果信号为1，则TX差分输出端口被强制设为固定1/0。TXN=1，TXP = 0
TXINTDATAWIDTH [1:0]	I	2	(00, 01, 10, 11) 指示内部数据宽度。
TXKERR[7:0]	O	1, 2, 4, 8(1)	在64b/66b模式中，指示旁路的偶边界。
TXN	O	1	差分发送端口（FPGA外部）
TXOUTCLK	O	1	从RocketIO X transmitter得到的合成时钟。此时钟可能和BREFCLK有一定的比例关系（例如：64b/66b模式），取决于发送器所采用的特定运行模式。
TXP	O	1	差分发送端口（FPGA外部）
TXPOLARITY	I	1	决定是否对发送器的最终输出取反，即改变TXN和TXP的极性。信号为0表示正常极性，信号为1表示反极性。
TXRESET	I	1	TX系统的同步复位，实现将发送弹性缓冲器归位，同时复位8b/10b编码器和其他内部接收寄存器。但是此操作并不复位发送PLL。
TXRUNDISP[7:0]	O	1, 2, 4, 8(1)	用于指示相应字节的运行不一致（当此字节被编码之后）。信号为0表示负不一致性，而信号1表示正不一致性。此端口还被过载作为PMA属性总线的数据输出总线。
TXSCRAM64B66BUSE	I	1	如果信号为1，则使能64b/66b扰码器。如果信号为0，则旁路64b/66b扰码器。TXSCRAM64B66USE = TXGEARBOX64B66BUSE

表1-4 设计原型端口（续）

端口	输入/输出（I/O）	端口大小	定义
TXUSRCLK	I	1	从DCM输出的时钟，时钟和REFCLK（或其他参考时钟）同步。此时钟是写TX缓冲器的时钟信号，它相对REFCLK是频率锁定的。TXUSRCLK和TXUSRCLK2必须保持180度的相位差。
TXUSRCLK2	I	1	从DCM输出的时钟信号，为发送器的数据提供时钟，同时也是收发器和FPGA结构之间状态和重配置数据的时钟。TXUSRCLK和TXUSRCLK2之间的比例取决于TXDATA的宽度。TXUSRCLK和TXUSRCLK2必须保持180度的相位差。

注：

1 端口大小取决于所使用的设计原型（1，2，4，8字节）。

2 端口大小取决于所使用的设计原型（8，16，32，64字节）。

设计原型属性

设计原型中还包括一些其他的属性，其值被缺省设置，以控制具体设计原型的相关协议参数。这些属性包括：通道绑定设置（对于支持通道绑定的设计原型而言），和时钟修正序列。表1-5 给出了每个属性的大体描述。

表1-5 RocketIO X 收发器属性

属性	类型	描述
ALIGN_COMMA_WORD	Integer（整型）	整数(1, 2, 4) 控制在收发器的4字节宽数据通道中检测到的comma字符的对齐。
CHAN_BOND_64B66B_SV	Boolean（布尔）	TRUE/FALSE（真/假）。此信号保留为将来使用，其值必须保持为FALSE。
CHAN_BOND_LIMIT	Integer（整型）	整数1-63定义了从接收器在跟随通道绑定序列并保持成功对齐的情况下所能读取的最大字节数。此属性必须设置为16。

表1-5 RocketIO X 收发器属性（续）

属性	类型	描述
CHAN_BOND_MODE	String (字符串)	STRING OFF, MASTER, SLAVE_1_HOP, SLAVE_2_HOPS **OFF:** 此收发器没有进行通道绑定。 **MASTER:** 此收发器作为通道绑定的主收发器。其CHBONDO端口可以直接驱动一个或多个SLAVE_1_HOPS收发器的CHBONDI端口。 **SLAVE_1_HOP:** 此收发器作为通道绑定的从收发器。SLAVE_1_HOP的CHBONDI由主收发器的CHBONDO端口直接驱动。SLAVE_1_HOP的CHBONDO端口可以直接驱动一个或多个SLAVE_2_HOPS收发器的CHBONDI端口。 **SLAVE_2_HOPS:** 此收发器作为通道绑定的从收发器。SLAVE_2_HOPS的CHBONDI 由SLAVE_1_HOP的CHBONDO 端口直接驱动。
CHAN_BOND_ONE_SHOT	Boolean （布尔）	TRUE/FALSE（真/假），控制通道绑定重复执行。 **FALSE:** 只要ENCHANSYNC为高电平且RXRESET为低电平，主收发器在任何可能的时候（当输入检测到通道绑定序列时）发起通道绑定。 **TRUE:**当RXRESET为低且ENCHANSYNC为高时，主收发器可以进行通道绑定，并且只在第一次可能时（当输入检测到通道绑定序列时）执行通道绑定操作。当通道绑定完成之后，不会再进行通道绑定，直到RXRESET变为1并重新复0或者ENCHANSYNC变成0并恢复到1。从收发器的CHAN_BOND_ONE_SHOT必须设置为FALSE。

表1-5 RocketIO X 收发器属性（续）

属性	类型	描述
CHAN_BOND_SEQ_1_*[10:0]	11-bit vector（向量）	此属性定义了通道绑定的序列，这些向量的用途取决于CHAN_BOND_SEQ_LEN和CHAN_BOND_SEQ_2_USE。
CHAN_BOND_SEQ_1_MASK[3:0]	4-bit vector（向量）	向量中的每一位指示是否检测到特定的序列，而忽略序列具体值。如果位0为高，表示和CHAN_BOND_SEQ_1_1相匹配，而忽略序列具体值。
CHAN_BOND_SEQ_2_*[10:0]	11-bit vector（向量）	此属性定义了通道绑定的序列，这些向量的用途取决于CHAN_BOND_SEQ_LEN和CHAN_BOND_SEQ_2_USE。
CHAN_BOND_SEQ_2_MASK[3:0]	4-bit vector（向量）	向量中的每一位指示是否检测到特定的序列，而忽略序列具体值。如果位0为高，表示和CHAN_BOND_SEQ_2_1相匹配，而忽略序列具体值。
CHAN_BOND_SEQ_2_USE	Boolean（布尔）	FALSE/TRUE（真/假）， 控制第二通道绑定序列的使用。 **FALSE:** 通道绑定只使用一个通道绑定序列，由CHAN_BOND_SEQ_1_1 ... 4定义，或者是由CHAN_BOND_SEQ_1_X 和CHAN_BOND_SEQ_2_X联合定的8-字节序列。 **TRUE:** 通道绑定使用2个通道绑定序列，由CHAN_BOND_SEQ_1_1 ... 4 和CHAN_BOND_SEQ_2_1 ... 4定义，并且受CHAN_BOND_SEQ_LEN限制。
CHAN_BOND_SEQ_LEN	Integer（整型）	整数(1, 2, 3, 4, 8) 定义了通道绑定序列的长度。收发器在检测能否进行通道绑定时，此属性定义了用于匹配的序列长度。
CLK_COR_8B10B_DE	Boolean（布尔）	此信号选择时钟修正是和8b/10b码流的编码版本相关，还是和8b/10b码流解码版本相关。如果值为真，使用解码版本。如果值为假，则使用编码版本。此属性必须和RXDEC8B10USE联合设置，CLK_COR_8B10B_DE = RXDEC8B10BUSE

表1-5 RocketIO X 收发器属性（续）

属性	类型	描述
CLK_COR_MAX_LAT	Integer（整型）	(0-63) 整数，定义了接收FIFO的上边界。此属性推荐设置为48。
CLK_COR_MIN_LAT	Integer（整型）	(0-63) 整数，定义了接收FIFO的下边界。此属性推荐设置为32。
CLK_COR_SEQ_1_*[10:0]	11-bit vector（向量）	此属性定义了时钟修正序列，其使用方式取决于CLK_COR_SEQ_LEN和CLK_COR_SEQ_2_USE。
CLK_COR_SEQ_1_MASK [3:0]	4-bit vector（向量）	向量中的每一位指示是否检测到特定的序列，而忽略序列具体值。如果位0为高，表示和CLK_COR_SEQ_1_1相匹配，而忽略序列具体值。
CLK_COR_SEQ_2_*[10:0]	11-bit vector（向量）	此属性定义了时钟修正的序列，其属性的使用取决于CLK_COR_SEQ_LEN 和CLK_COR_SEQ_2_USE。
CLK_COR_SEQ_2_MASK [3:0]	4-bit vector（向量）	向量中的每一位指示是否检测到特定的序列，而忽略序列具体值。如果位0为高，表示和CLK_COR_SEQ_2_1相匹配，而忽略序列具体值。
CLK_COR_SEQ_2_USE	Boolean（布尔）	FALSE/TRUE（真/假）， 控制第二时钟修正序列的使用。 **FALSE:** 通道绑定只使用一个通道绑定序列，由CLK_COR_SEQ_1_1 ... 4定义，或者是由CLK_COR_SEQ_1_X 和CLK_COR_SEQ_2_X联合定的8-字节序列。 **TRUE:** 通道绑定使用2个通道绑定序列，由CLK_COR_SEQ_1_1 ... 4和CLK_COR_SEQ_2_1 ... 4,定义，并且受CLK_COR_SEQ_LEN限制。

表1-5 RocketIO X 收发器属性（续）

属性	类型	描述
CLK_COR_SEQ_DROP	Boolean（布尔）	FALSE/TRUE（真/假）。如果为真，时钟修正是通过idle时隙的移除实现的。如果为假，则时钟修正是通过idle时隙的插入或者移除来实现的。此属性必须被设置为FALSE。
CLK_COR_SEQ_LEN	Integer（整型）	整数(1, 2, 3, 4, 8) 定义了时钟修正序列的长度。此属性定义了收发器在检测能否进行时钟修正时，用于匹配的序列长度。此属性同时定义了修正的大小，因为收发器是通过重复或者跳过整个时钟修正序列，从而实现时钟修正的。
CLK_CORRECT_USE	Boolean（布尔）	FALSE/TRUE（真/假），控制时钟修正逻辑的使用。 **FALSE:** 永久禁用时钟修正（速率匹配）。此时，时钟RXUSRCLK必须和RXRECCLK保持频率锁定。 **TRUE:** 启用时钟修正（一般模式）。
COMMA_10B_MASK[9:0]	10-bit vector（向量）	输入的串行比特流和此属性所定义的标记做与操作，之后再和PCOMMA_10B_VALUE、MCOMMA_10B_VALUE相比较。
DEC_MCOMMA_DETECT	Boolean（布尔）	FALSE/TRUE（真/假），控制基于“-”comma字符的每个字节标记 RXCHARISCOMMA。
DEC_PCOMMA_DETECT	Boolean（布尔）	FALSE/TRUE（真/假），控制基于“+”comma的每个字节标记 RXCHARISCOMMA。

表1-5 RocketIO X 收发器属性（续）

属性	类型	描述
DEC_VALID_COMMA_ ONLY	Boolean（布尔）	FALSE/TRUE（真/假），当检测到合法comma字符时控制RXCHARISCOMMA的设置 **FALSE:** 当检测到xxx1111100（如果DEC_PCOMMA_DETECT为真）和/或xxx0000011（如果DEC_MCOMMA_DETECT为真）或者8b/10b发送comma字符时，设置RXCHARISCOMMA（忽略xxx位的内容）。 **TRUE:** 仅当在8b/10b传送中检测到合法字符时，设置RXCHARISCOMMA。
MCOMMA_10B_VALUE [9:0]	10-bit vector（向量）	此属性定义了“-”comma字符，用于设置RXCOMMADET和重新对齐串行比特流的字节边界。此定义不影响8b/10b编码/解码。参见COMMA_10B_MASK。
MCOMMA_DETECT	Boolean（布尔）	FALSE/TRUE（真/假），用于指示当检测到“-”comma字符时是否设置RXCOMMADET。
PCOMMA_10B_VALUE[9:0]	10-bit vector（向量）	此属性定义了“+”comma字符，用于设置RXCOMMADET和重新对齐串行比特流的字节边界。此定义不影响8b/10b编码/解码。参见COMMA_10B_MASK。
PCOMMA_DETECT	Boolean（布尔）	FALSE/TRUE（真/假），用于指示检测到“+”comma字符时是否设置RXCOMMADET。
PMA_PWR _CNTRL	Integer（整型）	此属性定义了PMA的启动序列，必须始终设置为全1。
PMA_SPEED	String（字符串）	(13_40) 选择PMA的模式，参见PMA部分内容以选择合适的模式。
RX_BUFFER_USE	Boolean（布尔）	FALSE/TRUE（真/假），建议始终设置为真。此属性用于启用接收侧缓冲器，当值为真时启用该缓冲器。

表1-5 RocketIO X 收发器属性（续）

属性	类型	描述
RX_LOS_INVALID_INCR[7:0]	Integer（整型）	平方值范围从1到128，用于指示由于同步的缺失，需要使用多少个合法字符来抵偿1个非法字符的出现。
RX_LOS_THRESHOLD	Integer（整型）	平方值范围从4到512。其值除以RX_LOS_INVALID_INCR后，表示使FSM传送进入“同步丢失”状态所需要接收到的非法字符数
RX_LOSS_OF_SYNC_FSM	Boolean（布尔）	未定义。
SH_CNT_MAX[7:0]	8-bit vector（向量）	8位二进制数，控制64b/66b同步状态机何时进入同步状态（最大同步报头数）
SH_INVALID_CNT_MAX[7:0]	8-bit vector（向量）	8位二进制数，控制64b/66b同步状态机何时离开同步状态（最大非法同步报头数）
TX_BUFFER_USE	Boolean（布尔）	当其值为真时，启用发送缓冲器。

可更改属性

如附录 F（RocketIO X 用户指南中的“Modifiable Attributes”）所示，只有部分属性可以被所有设计原型更改。这些属性用于帮助定义设计原型所选用的协议。只有 GT10_CUSTOM 设计原型允许用户更改所有的属性，这样定义的协议不能为其他收发器设计原型所支持，但这种方式可以有完全的机动性。其他的设计原型只允许用户修改串行数据线的模拟属性以及一些通道绑定参数。

字节映射

大多数的 8 位状态和控制总线与 TXDATA 或 RXDATA 的特定字节相关联。对应关系如表 1-6 所示。这种方法可以实现所有信号结合在一起，而不用考虑 GT10_CUSTOM 所需要的数据通道宽度。其他所有配备特殊数据宽度通道的设计原型以及所有字节映射端口都会为这种情况所影响。例如：一个 1 字节宽的数据通道仅有 1 位的控制和状态位(TXCHARISK[0])

和数据位 TXDATA[7:0]相关联。

表1-6 和数据总线字节通道相关的控制/状态总线

控制/状态位	数据位
[0]	[7:0]
[1]	[15:8]
[2]	[23:16]
[3]	[31:24]
[4]	[39:32]
[5]	[47:40]
[6]	[55:48]
[7]	[64:56]

数字设计应考虑的事项

值得注意的是 RocketIO X 的物理编码子层（Physical Coding Sublayer，PCS）部分相对 RocketIO 有了很大改进。RocketIO X PCS 支持 8b/10b 和 64b/66b 编码/解码，SONET 兼容性以及通用数据模式。RocketIO X 收发器可以工作在四个基本内部模式：16 位，20 位，32 位和 40 位。伴随物理介质接入层（Physical Media Attachment，PMA）的预定义模式，RocketIO X 有更多协议和数据速率组合供用户选择。使用定制的 RocketIO X 收发器来构建通信芯片历史上最高级和易配置的通信通道时，用户可以有很多种组合的选择方案。

RocketIO X PCS 在收发器的可配置性方面也有所改善。用户不仅可以实时地改变 PMA 的速率，而且可以改变 PCS 中的协议。无论是内部数据宽度、外部数据宽度或数据的路由，都是基于全时钟的环境。基于此项改进，用户可以低速初始化通信通道（例如，速率为 2.5Gb/s，使用 8b/10b，内部宽度 20 位），当通道稳定之后自动协调切换到 10.3125 Gb/s（使用 64b/66b，内部宽度 32 位）。

为帮助 RocketIO X 用户更好地理解设计原型中的独立属性和控制端口设置，此部分的内容提供了相关的信息以供参考。用户可以选择使用表 A-1-3（第 111 页）中所列出的 RocketIO X 支持的各种设计原型，而不需要阅读以下的内容。以下的内容可以帮助用户进一步理解 PCS 的配置，以便通过修改属性和端口值来实现特定的收发器配置。

顶层结构

发送结构

PCS 的发送结构如图 1-2 所示。关于如何禁用某个块的详细信息，参见该块的相关章节。

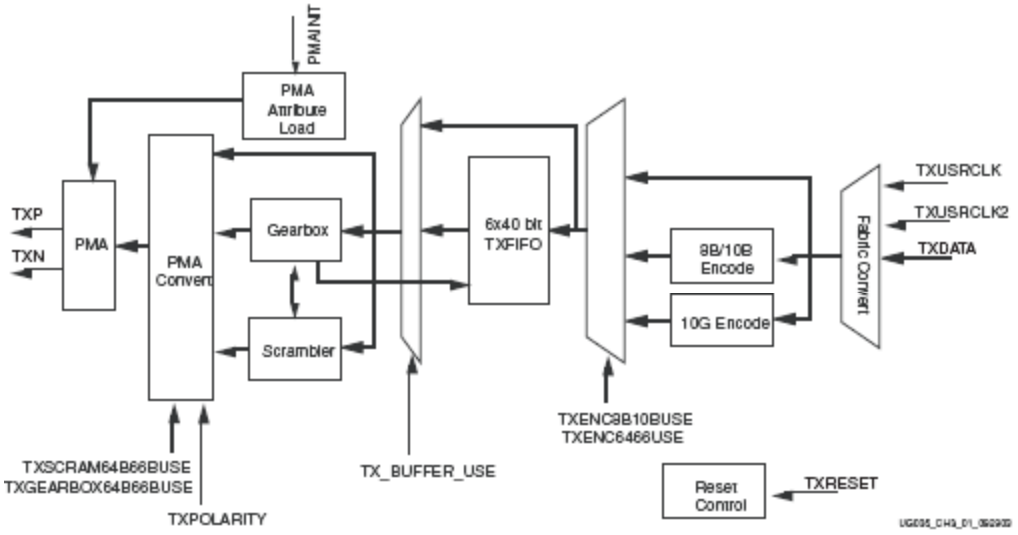


图 1-2 发送结构

接收结构

PCS 的接收结构如图 1-3 所示。关于如何禁用某个块的详细信息，参见该块的相关章节。

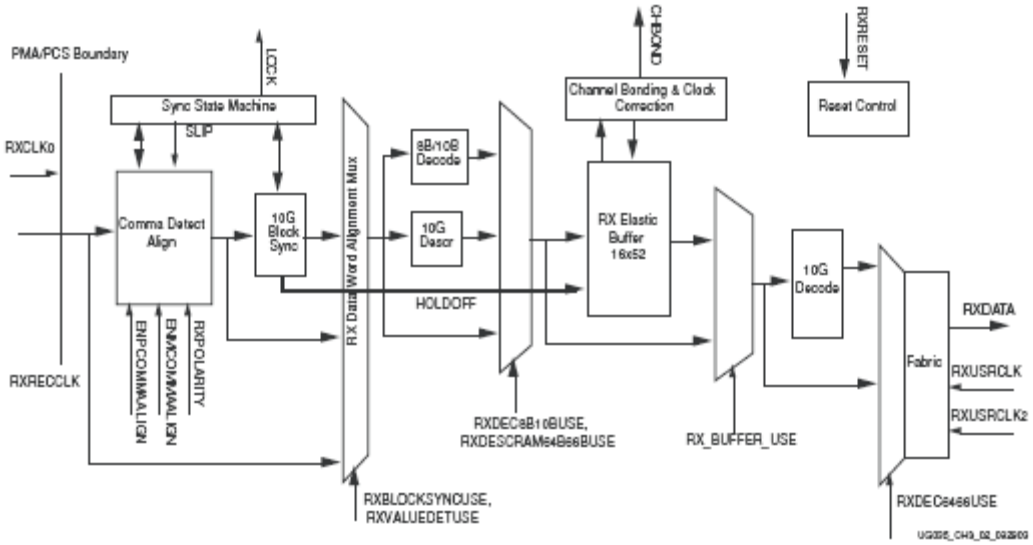


图 1-3 接收结构

运行模式

PCS 存在四种内部运行模式：16 位，20 位，32 位和 40 位。

从本质上看，PCS 工作在 2 字节模式或者 4 字节模式，2 字节模式对应 16 位和 20 位模式，4 字节模式对应 32 位和 40 位模式。当 PCS 工作在 2 字节模式时，外部接口宽度可以是 1，2 或 4 字节。当 PCS 工作在 4 字节模式时，外部接口宽度可以是 4 或者 8 字节。所以内部宽度为 2 字节而外部接口宽度为 8 字节的情况是不可能存在。同样，也不存在内部宽度为 4 字节而外部接口宽度为 1 字节的情况。如表 1-7 所示。

表1-7 PCS接口选择

速度	2 字节(内部)	4 字节(外部)
2.488 Gb/s	推荐	禁用
5 - 10.3125 Gb/s	禁用	推荐

通常来说，当串行速率低于 5Gb/s 时 PCS 使用 2 字节模式，当串行速率高于 5Gb/s 时使用 4 字节模式。工作在 2 字节模式时，PCS 会将 4 字节的数据分割为 2 字节的段进行处理。这种操作对于用户来说是透明的，但是和 Virtex-II Pro 收发器相比较，收发器之间的偏差会导致发送接口处发生更大的位偏差。在 2 字节模式和 4 字节模式中，都可以使用三种编码配

置（8b/10b 和 64b/66b 编码/解码，SONET 和通用数据模式）中的任一种，每个块都可以被旁路。

关于设置 PCS 模式的更多信息，请参见本指南中,总线接口的块功能定义。

块级功能

信号和过载的分类

本节描述了 PCS 接口的相关信号以及如何按优先顺序区分这些信号。关于特定信号的详细信息，参见 I/O 说明或特定的块功能。

静态信号（控制输入）

下面所列的静态信号用于控制 PCS 内部和外部运行模式。在运行模式选定之后，这些信号是应该最先考虑的典型信号：

- RXDATAWIDTH[1:0]
- RXINTDATAWIDTH[1:0]
- TXDATAWIDTH[1:0]
- TXINTDATAWIDTH[1:0]

下面所列的静态信号控制 PCS 内部的区块路由和特定块的旁路，从而使 PCS 的结构适合于用户的具体应用：

- RXBLOCKSYNC64B66BUSE
- RXDEC64B66BUSE
- RXDEC8B10BUSE
- RXDESCRAM64B66BUSE
- RXCOMMADETUSE
- TXENC64B66BUSE
- TXENC8B10BUSE
- TXGEARBOX64B66BUSE
- TXSCRAM64B66BUSE

下面所列的静态信号用于控制各种不同的功能，但这些信号通常都是在状态机的一开始或者在自定义序列之后一次性设置的。基于 clock-by-clock 准则，这些信号通常不会被改变。

- ALIGN_COMMA_WORD
- ENMCOMMAALIGN
- ENPCOMMAALIGN
- RXPOLARITY
- TXPOLARITY
- PMAINIT
- RXIGNOREBTF
- PMARXLOCKSEL[1:0]

下面所列的静态信号可以启动主功能模块复位或者用于故障恢复。这些信号主要用在初始化过程中，而通常不用在正常的运行过程中。

- LOOPBACK[1:0]（*注：此信号也可以是动态信号*）

- POWERDOWN
- RXRESET
- TXRESET
- TXINHIBIT

动态信号

下列的动态信号指示接收总线上接收到的数据，同时还指示 RXDATA 具体状态信息。这些动态信号的值定义了用户设置的应用。这些信号是配置完静态信号之后最重要的信号之一：

- MCOMMA_10B_VALUE
- PCOMMA_10B_VALUE
- COMMA_10B_MASK
- RXCHARISCOMMA[7:0]
- RXCHARISK[7:0]
- RXDISPERR[7:0]
- RXNOTINTABLE[7:0]
- RXDATA[63:0]

下列的动态信号指示发送总线上待传送的数据，同时还指示 TXDATA 如何通过 PCS 进行传送的状态信息。这些信号的值定义了用户设置的应用。这些信号是配置完静态信号之后最重要的信号之一：

- TXBYPASS8B10B[7:0]
- TXCHARDISPMODE[7:0]
- TXCHARDISPVAL[7:0]
- TXCHARISK[7:0]
- TXDATA[63:0]

下列的动态信号指示 PCS 当前状态及先前状态的各种状态信息：

- CHBONDDONE, RXBUFSTATUS[1:0], RXCLKCORCNT[2:0]
- CHBONDO[4:0]
- PMARXLOCK
- RXLOSSOFSYNC[1:0]
- RXREALIGN
- RXCOMMADET
- TXBUFERR
- TXKERR[7:0]
- TXRUNDISP[7:0]
- RXRUNDISP[7:0]

下列的动态信号控制 PCS 的内部状态：

- RXSLIDE
- CHBONDI[4:0]

下列的动态信号可以影响 PMA 的控制寄存器：

- PMAREGADDR[5:0]
- PMAREGDATAIN[7:0]

- PMAREGRW
- PMAREGSTROBE

总线接口

选择外部配置（逻辑接口）

通过使用信号 TXDATAWIDTH[1:0] 和 RXDATAWIDTH[1:0]，可以选定逻辑接口的结构。

表1-8 选择外部配置

RXDATAWIDTH/TXDATAWIDTH	数据宽度	内部总线需求
2’b00	8/10-bit (1 byte)	16-, 20-bit mode
2’b01	16/20-bit (2 byte)	16-, 20-bit mode
2’b10	32/40-bit (4 byte)	16-, 20-, 32-, 40-bit mode
2’b11	64/80-bit (8 byte)	32-, 40-bit mode

选择内部配置

表1-9 选择内部配置

RXINTDATAWIDTH/TXINTDATAWIDTH	内部数据宽度
2’b00	16-bit
2’b01	20-bit
2’b10	32-bit
2’b11	40-bit

时钟速率比

USRCLK2 是数据缓冲器的时钟。为了实现以四种不同的数据宽度传送并行数据给收发器，用户需要改变 USRCLK2 的频率。所以，USRCLK 和 USRCLK2 之间存在一定的频率比例关系。注意时钟的下降沿必须对齐。如表 1-10 所示。

表1-10 数据宽度，时钟比例

结构数据宽度	USRCLK\USRCLK2 的频率比	
	2-Byte内部数据宽度	4-Byte内部数据宽度
1 byte	1:2(1)	N/A
2 byte	1:1	N/A

表1-10 数据宽度，时钟比例*(续)*

结构数据宽度	USRCLK\USRCLK2 的频率比	
	2-Byte内部数据宽度	4-Byte内部数据宽度
4 byte	2:1(1)	1:1
8 byte	N/A	2:1(1)

注：

1. 慢时钟的每个边沿都必须和快时钟的边沿对齐。

8b/10b

RocketIO 收发器首先发送最高字节，而 RocketIO X 收发器首先发送最低字节。

本节将根据具体的功能对收发器的端口和属性进行分类，这些功能包括 8b/10b 编码/解码、64b/66b 编码/解码、SERDES 对齐、时钟修正（时钟恢复）、通道绑定、逻辑接口以及其他信号。

8b/10b 编码将待传送的 8 位并行数据转换为 10 位的串行数据流，转换和数据对齐如图 1-4 所示。串行端口首先传送 10 位数据的最低位“a”，逐位发送数据直到最高位“j”。通过这种方式，数据能够以附录 B（“8b/10b Valid Characters.”）中所列的各种格式进行读取和匹配。

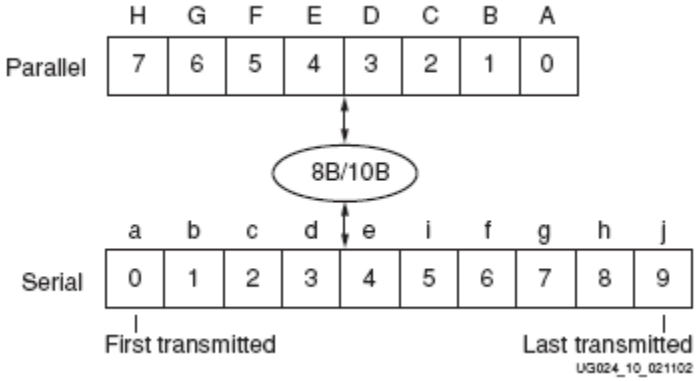


图 1-4 8b/10b 并串转换

串行数据比特序列取决于并行数据的宽度。无论是采用 1 字节、2 字节、4 字节或 8 字节中的任一种数据通道，首先发送的都是最低字节。最高字节总是最后发送。图 1-5 的例子中给出了对应于每个并行数据字节的串行数据。TXDATA[7:0]首先被串行化并发送，之后是

TXDATA[15:8]、TXDATA[23:16]，最后是 TXDATA[31:24]。如果是 2 字节通道，则首先发送 TXDATA[7:0]之后是 TXDATA[15:8]。

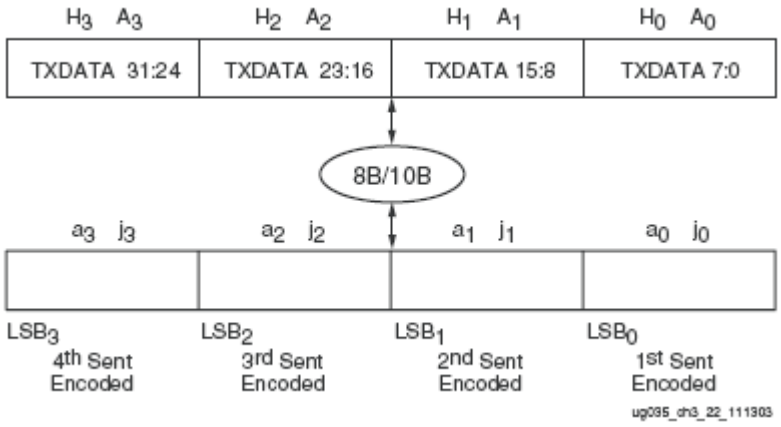


图 1-5　4 字节串行结构

编码器

收发器内部含有一个可旁路的 8b/10b 编码器。编码器使用通用的 256 个数据字符和 12 个控制字符（如附录 B，“8b/10b Valid Characters”所示）, Gigabit Ethernet,、XAUI、Fibre Channel 和 InfiniBand 均采用这些字符。

编码器接收 8 位的数据以及 1 位的 K 字符标记，每个提交的字符应为 9 个比特。如果 K 字符标记为高，数据编码为 8b/10b 码本中 12 个 K 字符的某一个。如果 K 字符标记为低，则 8 位数据编码为标准数据。

收发器中有两个端口用于启用 8b/10b 编码功能。TXBYPASS8B10B 是一个字节映射端口，其大小可以是 1、2、4 或 8 位，取决于收发器设计原型的数据宽度。这些位和数据通道的每个字节相关联。为了启用收发器的 8b/10b 编码功能，这些位需要设置为逻辑 0。在这种模式中，输入到 TXDATA 端口的待传送数据可以是 8 位、16 位、32 位或 64 位的未编码数据。尽管如此，如果要采用其他的编码机制，可以将所有位设置为逻辑 1，从而禁用 8b/10b 编码功能。额外的数据位则通过 TXCHARDISPMODE 和 TXCHARDISPVAL 总线来传输。

TXCHARDISPVAL 和 TXCHARDISPMODE

TXCHARDISPVAL和TXCHARDISPMODE是收发器的复用端口，其功能取决于是否启用8b/10b编码功能。表1-12给出了它们的双重功能。当启用编码功能时，这些端口作为字节映

射的控制端口，用于控制发送串行数据的运行不一致性（如表1-11所示）。

表1-11 运行不一致性控制

{txchardispmode, txchardispval}	功能
00	维持原来的运行不一致性
01	在字节编码前，反转原来的运行不一致性
10	在字节编码前，将运行不一致性设置为负。
11	在字节编码前，将运行不一致性设置为正。

在编码配置中，串行发送的不一致性可以由 TXCHARDISPVAL 端口和 TXCHARDISPMODE 端口来控制。当 TXCHARDISPMODE 设为逻辑 1，运行不一致性在编码该字节前设定。TXCHARDISPVAL 决定不一致性是正（设置为逻辑 1 ）还是负（设置为逻辑 0 ）。

当 TXCHARDSIPMODE 设置为逻辑 0 时，若 TXCHARDISPVAL 同样设置为逻辑 0，则维持原来的运行不一致性。如果 TXCHARDISPVAL 为逻辑 1，则在编码该字节前反转运行不一致性。

多数应用采用 TXCHARDISPMODE 和 TXCHARDISPVAL 均为逻辑 0 的模式。如果部分应用需要使用特殊的运行不一致性模式，也可以使用其他配置的模式，示例可以参见 “Vitesse Disparity Example,”第 139 页。

在禁用 8b/10b 编码的情况下，TXCHARDISPMODE[0]变成 10 位编码数据的第 9 位（在 20 位、 40 位和 80 位总线中，TXCHARDISPMODE[1:7]分别是第 19、29、39、49、59、69、79 位）。当 TXDATA 总线配置其总线时，TXCHARDISPVAL 成为传送数据总线的第 8、18、28、38、48、68、78 位。见表 1-12。

表1-12 8b/10b被旁路情况下信号的意义

信号		功能
TXBYPASS8B10B(1)	0	启用8b/10b编码 （未被旁路）
	1	旁路8b/10b编码 （禁用）

表1-12 8b/10b被旁路情况下信号的意义（续）

信号		功能	
TXCHARDISPMODE, TXCHARDISPVAL		**启用8b/10b编码**	**旁路8b/10b 编码**
	00	维持平常的运行不一致性	10位编码字节的部分(如图1-6所示): TXCHARDISPMODE[0], (or: [1] / [2] / [3] /[4]/[5]/[6]/[7]) TXCHARDISPVAL[0], (or: [1] / [2] / [3] /[4]/[5]/[6]/[7]) TXDATA[7:0] (or: [15:8] / [23:16] / [31:24]/ [39:32]/[47:40]/[55:48]/[63:56])
	01	在字节编码前，反转通常的运行不一致性	
	10	在字节编码前，将运行不一致性设置为负。	
	11	在字节编码前，将运行不一致性设置为正。	
TXCHARISK		接收到的字节为K字符	未使用

注：

1. 若禁用8b/10b编码，同时启用64b/66b编码时，此端口仍有定义

在发送期间，如果启用8b/10b编码，则串行发送的不一致性可以由TXCHARDISPVAL和TXCHARDISPMODE端口控制。如果禁用8b/10b编码，这些位变成收发器发送到接收端的10位编码数据的一部分，分别代表位“b”和“a”。图1-6是禁用8b/10b时TX的数据映射示意图。

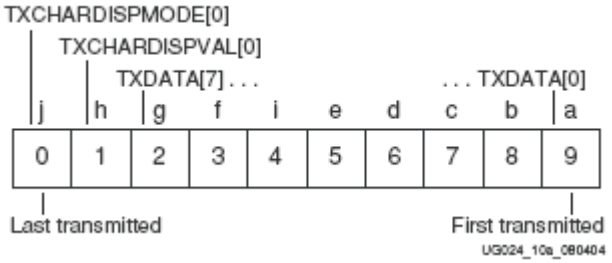


图1-6 旁路8b/10b期间10位TX数据映射示意图

TXCHARISK

TXCHARISK是一个字节映射的控制端口，只有启用8b/10b编码时才使用该端口。TXCHARISK用于指示待编码的TXDATA字节的类型，信号为1表示控制（K）字符，信号为0表示数据字符。如果8b/10b编码被禁用，则该端口没有定义。

TXRUNDISP

TXRUNDISP是字节映射到TXDATA的状态端口，此端口用于指示TXDATA字节编码后的运行不一致性。信号为1表示不一致性为正，信号为0表示不一致性为负。

解码器

接收器中有一个可选的8b/10b解码器。通过一个可编程选项，解码器可以被旁路。如果8b/10b被旁路，则10位的字符顺序如图1-7所示，图中给出了接收到的10位字符的图形示意。

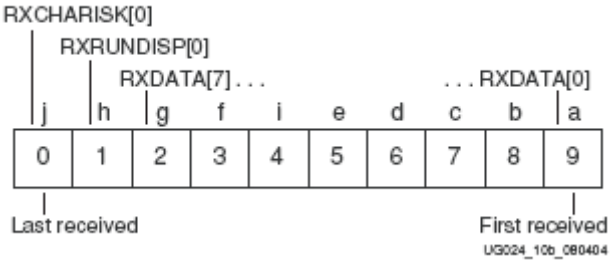


图1-7 旁路8b/10b解码器情况下10位RX数据映射示意图

解码器使用相同的字符集（如附录B“8b/10b Valid Characters”所示），这些字符集适用于Gigabit Ethernet、Fibre Channel以及InfiniBand。除了解码所有数据和K字符外，解码器还有不少扩展功能。解码器分别检测“不一致性错误”和“带外(编码以外)”错误。运行不一致错误是指：接收到的10位字符在8b/10b字符集中，但不一致性是错误的。“带外(编码以外)”错误是指：接收到的10位字符不在8b/10b字符集中。在没有发生“不一致性错误”的情况下，“带外”错误仍然可能出现。通常情况下，在没有出现“带外”错误的情况下，发生“不一致性错误”。无论字符是合法还是非法的，相应的及合法的不一致性都会被计算。当前的运行不一致性可以从RXRUNDISP信号获取。

当8b/10b解码器检测到“带外(编码以外)”数据时，会执行特定操作。解码器检测到“带外”数据时，会给出错误通知，同时非法的10位数据会被传送到输出端口。这些操作在调试时十分有用。

解码器接收到控制字符时，同样也会给出相应通知。另外，解码器还含有一个可编程的comma字符检测逻辑。当接收到comma+、comma-或同时接收到两者时，comma检测逻辑会标记相应的寄存器。comma字符被定义成7位字符，包含了数个“带外”的字符。还有另一种选择，解码器只会检测三种已定义的comma字符（K28.1, K28.5, 和K28.7），以及comma+、comma–或两者都有。所以，总共有6种可能，三种针对合法字符，另外三种针对“所有comma字符”。

并不是RX FPGA接口接收到的所有字节（1、2、4或8）都有自己独立的8b/10b指示器（K字符、不一致性错误、“带外”错误、当前运行不一致性和comma字符检测）。

在接收期间，如果启用8b/10b解码，则收发器可以通过RXRUNDISP端口获取串行传送数据的运行不一致性，RXCHARISK用于指示接收到K字符。如果旁路8b/10b解码，则这些位依然是收发器传送给用户逻辑的数据中的一部分，分别是位“b”和“a”。表1-13给出了旁路8b/10b解码情况下的RX数据映射。

表1-13 旁路8b/10b解码情况下各信号的意义

信号		功能	
RXCHARISK		接受到的字符为K字符	10位编码数据的一部分 (见图 1-7): RXCHARISK[0], (or: [1] / [2] / [3] /[4]/[5]/[6]/[7]) RXRUNDISP[0], (or: [1] / [2] / [3] /[4]/[5]/[6]/[7]) RXDATA[7:0] (or: [15:8] / [23:16] / [31:24]/ [39:32]/[47:40]/[55:48]/[63:56})
RXRUNDISP	0	指示不一致性为负	
	1	指示不一致性为正	
RXDISPERR		当前字节发生不一致性错误	未使用

RXCHARISK和RXRUNDISP

RXCHARISK和RXRUNDISP是接收器的两用端口，其功能取决于是否启用8b/10b解码器。其双重功能如图1-10所示。当使能8b/10b解码时，这些端口作为接收数据的字节映射状态端口。

在使能解码器的情况下，如果接收到的字符为控制（K）字符，则RXCHARISK信号变成1。否则，接收到的字符数据字符，RXCHARISK信号为0。（参见附录B“8b/10b Valid Characters”）。RXRUNDISP用于指示接收字节的不一致性。RXRUNDISP为1表示’+’不一致性。参见 “Vitesse Disparity Example” 第139页。

如果禁用解码器，则RXCHARISK和RXRUNDISP都是10位、20位、40位或80位总线的附加数据位。和传送端的情况类似，RXCHARISK[0:7]对应于数据总线的位9, 19, 29, 39, 49, 59, 69,和79，而RXRUNDISP对应于位8, 18, 28, 38, 48, 58, 68, 和78。如图1-10所示。

RXDISPERR

RXDISPERR是字节映射到RXDATA的接收器状态端口。RXDISPERR的某一位为1表示对应

的接收数据中发生了不一致性错误。此端口指示的错误通常包括数据中出现错误位、传送非法控制字符或者不一致性发生错误（参见 “Vitesse Disparity Example” 第139页）。

RXNOTINTABLE

如果接收到的数据不在 8b/10b 字符集中，则设置 RXNOTINTABLE 为 1。RXNOTINTABLE 被标记的字节是非法字节。只有在启用 8b/10b 解码器时才使用 RXNOTINTABLE 端口，此端口也是字节映射到 RXDATA 的。

Vitesse 不一致性示例

为了支持其他的协议，收发器可以改变串行发送数据的不一致性模式。例如，Vitesse 的通道到通道对齐协议，发送如下：

K28.5+ K28.5+ K28.5- K28.5-

或

K28.5- K28.5- K28.5+ K28.5+

代替

K28.5+ K28.5- K28.5+ K28.5-

或

K28.5- K28.5+ K28.5- K28.5+

TXCHARDISPVAL必须设置为1，使串行数据发送两个”-”运行不一致性字符。

Vitesse发送通道绑定序列

```
TXBYPASS8B10B
|  TXCHARISK
|  |  TXCHARDISPMODE
|  |  |  TXCHARDISPVAL
|  |  |  |  TXDATA
|  |  |  |  |
0 1  0  0  10111100     K28.5+ (or K28.5-)
0 1  0  1  10111100     K28.5+ (or K28.5-)
0 1  0  0  10111100     K28.5- (or K28.5+)
0 1  0  1  10111100     K28.5- (or K28.5+)
```

RocketIO X接收这些数据，但是在传送期间如果TXCHARDISPVAL信号为高，则disp_err位必须设置为高，同时CHAN_BOND_SEQ要进行相应设置。

Vitesse接收通道绑定序列

在接收端，通道绑定序列使用disp_err位来具体指示不一致性的反转。

```
                    10-bit literal value
                    |  disp_err
                    |  |  char_is_k
                    |  |  |  8-bit_byte_value
                    |  |  |  |
CHAN_BOND_SEQ_1_1  = 0 0 1 10111100     matches K28.5+ (或 K28.5-)
CHAN_BOND_SEQ_1_2  = 0 1 1 10111100     matches K28.5+ (或 K28.5-)
```

CHAN_BOND_SEQ_1_3 = 0 0 1 10111100 matches K28.5- (或 K28.5+)
CHAN_BOND_SEQ_1_4 = 0 1 1 10111100 matches K28.5- (或 K28.5+)
CHAN_BOND_SEQ_LEN = 4
CHAN_BOND_SEQ_2_USE = FALSE

Comma 字符检测

摘要

Comma 字符检测在原有的10位符号检测和对齐的基础上有了很大扩展，可以实现8位符号检测和16位、20位、32位和40位通道的对齐。检测逻辑检测符号，之后对齐到1字、2字或4字的边界。在SONET应用中， RXSLIDE端口允许用户在16位、20位、32位、40位模式中对边界进行1位的移动。

下列的信号和属性可以影响comma检测模块的功能：

- RXCOMMADETUSE
- ENMCOMMAALIGN
- ENPCOMMAALIGN
- ALIGN_COMMA_WORD[1:0]
- MCOMMA_10B_VALUE[9:0]
- DEC_MCOMMA_DETECT
- PCOMMA_10B_VALUE[9:0]
- DEC_PCOMMA_DETECT
- COMMA_10B_MASK[9:0]
- RXSLIDE
- RXINTDATAWIDTH[1:0]

旁路

只要将RXCOMMADETUSE设置为0，即可禁用符号检测。如果RXCOMMADETUSE信号为1，则启用符号检测。

符号检测

通过MCOMMA_10B_VALUE, DEC_MCOMMA_ DETECT, PCOMMA_10B_VALUE, DEC_PCOMMA_DETECT, 和COMMA _10B_MASK这些信号的使用，在两种不同符号值的设置下，可以实现任意的8位或10位符号检测。

检测10位符号时，COMMA_10B_MASK[9:0]应当初始化为10’b11111_11111。运行期间每位都可以被改变，以影响其标记。

检测8位符号时，COMMA_10B_MASK[9:0]应初始化为10’b00_1111_1111。最高的两位必须为0，后8位的值都可被修改，以影响其标记。

The MCOMMA_10B_VALUE[9:0]和PCOMMA_10B_VALUE[9:0]用于指示比较逻辑中所使用的comma符号定义，例如：在为建立对齐而搜寻comma字符时，和输入数据相比较的对齐符号。

DEC_MCOMMA_DETECT和DEC_PCOMMA_DETECT指示在检测对齐过程中，用于和

输入数据相比较的符号。

表1-4 符号检测

MCOMMA_DETECT	PCOMMA_DETECT	功能
0	0	不进行符号检测
0	1	如果输入数据检测并对齐到由PCOMMA_10B_VALUE定义的符号时，RXCOMMADET的值设置为1。
1	0	如果输入数据检测并对齐到由MCOMMA_10B_VALUE定义的符号时，RXCOMMADET的值设置为1。
1	1	如果输入数据检测并对齐到由PCOMMA_10B_VALUE或MCOMMA_10B_VALUE定义的符号时，RXCOMMADET的值设置为1。

设置MCOMMA_10B_VALUE, PCOMMA_10B_VALUE, 和 COMMA_10B_MASK (特别须知)

MGT使用属性、MCOMMA_10B_VALUE, PCOMMA_10B_VALUE, 和 COMMA_10B_MASK这些端口来定义字符检测模块中检测和对齐的字符。值设定之后，comma字符检测模块在数据流中检测这些字符，并将通道位置对齐到数据流中这些字符出现的位置。Virtex-II Pro X 用户需要注意这些值的设定和Virtex-II Pro是相反的。这是因为Virtex-II Pro主要用于支持8b/10b应用，而Virtex-II Pro X采用了更通用的方法以支持更多的应用。

图1-8给出了Virtex-II Pro X 8b/10b 符号检测的例子，其检测是针对PCS/PMA接口接收到的数据流的。注意：对Virtex-II Pro而言，M/PCOMMA_10B_VALUE [9:0]的值必须设置为10'b0011111010，而Virtex-II Pro X的值应当为10'b0101111100。两种器件的COMMA_10B_MASK端口设置也不相同，Virtex-II Pro的值为10'b1111111000，而Virtex-II Pro X的值应设置为10'b0001111111。

基于这些改进，检测模块不再仅仅是作为一个comma字符检测模块，更要考虑作为多种数据的检测模块。为了检测8b/10b字符集中所列的值，只需要反转相应的值即可。如果检测的是SONET类型的值，则不需要进行反转。

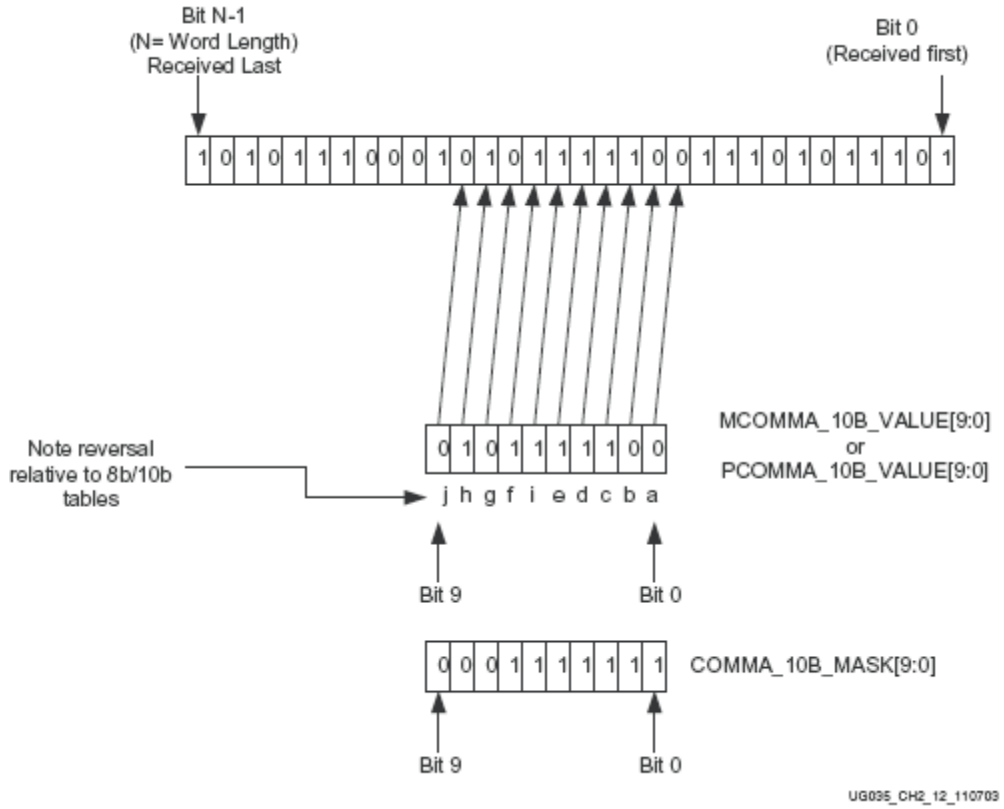


图1-8　8b/10b comma检测示例

对齐

在检测到正comma符号或负comma符号后，数据对齐到该符号。使用ENMCOMMAALIGN, ENPCOMMAALIGN, ALIGN_COMMA_WORD, 和RXSLIDE这些信号，可以完全控制各数据通道的对齐。见表1-15。

表1-15 数据对齐

ENMCOMMAALIGN	ENPCOMMAALIGN	Function(1)
0	0	没有发生对齐
0	1	若检测到正符号，则对齐到该符号的位置。

表1-15 数据对齐（续）

ENMCOMMAALIGN	ENPCOMMAALIGN	Function(1)
1	0	若检测到负符号，则对齐到该符号的位置。
1	1	若检测到正或负符号，则对齐到该符号的位置。

注：

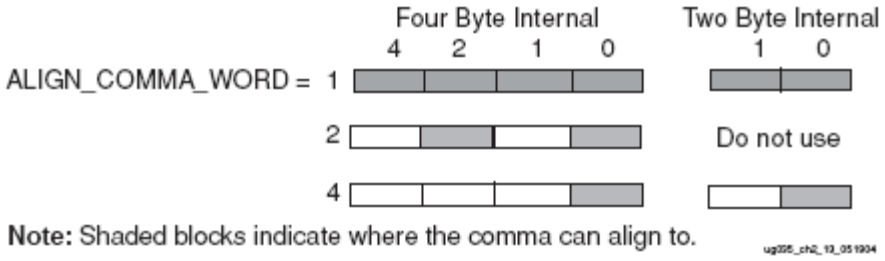
1. 这里提及的符号是由P/MCOMMA_10B_VALUE所定义的。

ALIGN_COMMA_WORD

若对齐的符号是基于字节-字节比较的，ALIGN_COMMA_WORD属性控制何时进行重对齐。如果当前检测得到的符号位置和先前符号的位置在字节的部分位不同，则进行重新对齐而不需考虑ALIGN_COMMA_WORD的值。

- ALIGN_COMMA_WORD有三种可选方式：1字节、2字节和4字节。当ALIGN_COMMA_WORD值为1时，检测电路可以逐个字节检测符号。当ALIGN_COMMA_WORD值为2时，检测电路每2个字节进行1次检测。当ALIGN_COMMA_WORD值为4时，检测电路每4个字节进行1次检测（如图1-9所示）。

	16/20	32/40
1	1字节对齐	1字节对齐
2	N/A	2字节对齐
4	2字节对齐	4字节对齐



注：阴影部分表示comma字符可以对齐的地方

图1-9: **ALIGN_COMMA_WORD 示意图**

RXSLIDE

RXSLIDE可以实现将对齐的数据移动1位。当信号为高时，RXSLIDE发挥作用，对齐位置逐位增1，直到最高位，即字长-1。RXSLIDE信号由高变低后，在再次变为高电平前，必须至少保持两个时钟周期的低电平。这个功能可以用在诸如SONET之类的应用中。

64b/66b

编码器

旁路

64b/66b编码器有两种旁路方式。用户可以将编码器时钟完全旁路，或者使用编码器并基于clock-by-clock准则进行旁路。

如果TXENC64B66BUSE为低，则全部的64b/66b都被禁用。如果在模块中已经进行了编码，则同步报头SH[0:1]必须赋给TXCHARDISPVAL[0]和TXCHARDISPMODE[0]。

如果TXENC64B66BUSE为高，则TXBYPASS8B10B的第0位决定是否旁路64b/66b编码器，旁路是基于clock-by-clock准则进行的，所以完整的模块旁路过程需要2个时钟周期。同步头可以从TXCHARDISPMODE[0:1]获得。为实现基于clock-by-clock准则的旁路，需要在交换接口处指示出偶边界，包含在TXKERR的第0位中。TXCHARISK信号实现TXC相应的功能。

表1-16 64b/66b 旁路

信号	功能	
TXENC64B66BUSE	0 禁用整个64b/66b编码器	
	1 基于时钟对时钟的旁路	
TXBYPASS8B10B[0]	基于clock-by-clock的64b/66b旁路	禁用整个64b/66b编码器
	0 表示不进行旁路	如表 1-18 中定义
	1 表示旁路该模块	
TXCHARDISPMODE[0:1]	如图1-11所示的同步头(和 SH[0:1]相同)	
TXKERR[3]	指示基于时钟准则旁路的偶边界	
TXCHARISK[3:0]	实现TXC的功能	指示该字符为控制（K）字符

发送端的64b/66b编码器借用了TXCHARISK总线的低4位（位[3：0]），用于传送控制信号给编码器。TXC的四位分别伴随着TXDATA_IN四个字节传送（TXC[0]对应TXDATA_IN[7:0]），用于指示数据块格式。传送交换接口逻辑（当数据从交换接口传送给PMA时，该逻辑最先进行监视）通过TXC的四位驱动编码器，如下所示：

表1-17 发送端64b/66b编码器控制映射表

TXC[3:0] (TXCHARISK[3:0])	块格式
1111	空闲或者伴随空闲的终止
0001	帧的开始或排序（ordered-set）
1110	终止在第2位置
1100	终止在第3位置
1000	终止在第4位置
0000	数据或者错误（没有控制字符）

TXC值中的每个“1”表示1个控制字符匹配，表示对应的字节是某种特殊的控制字符（例如idle, start, terminate，或ordered-set）。

标准操作

64b/66b编码器的编码块格式功能如图1-11所示：

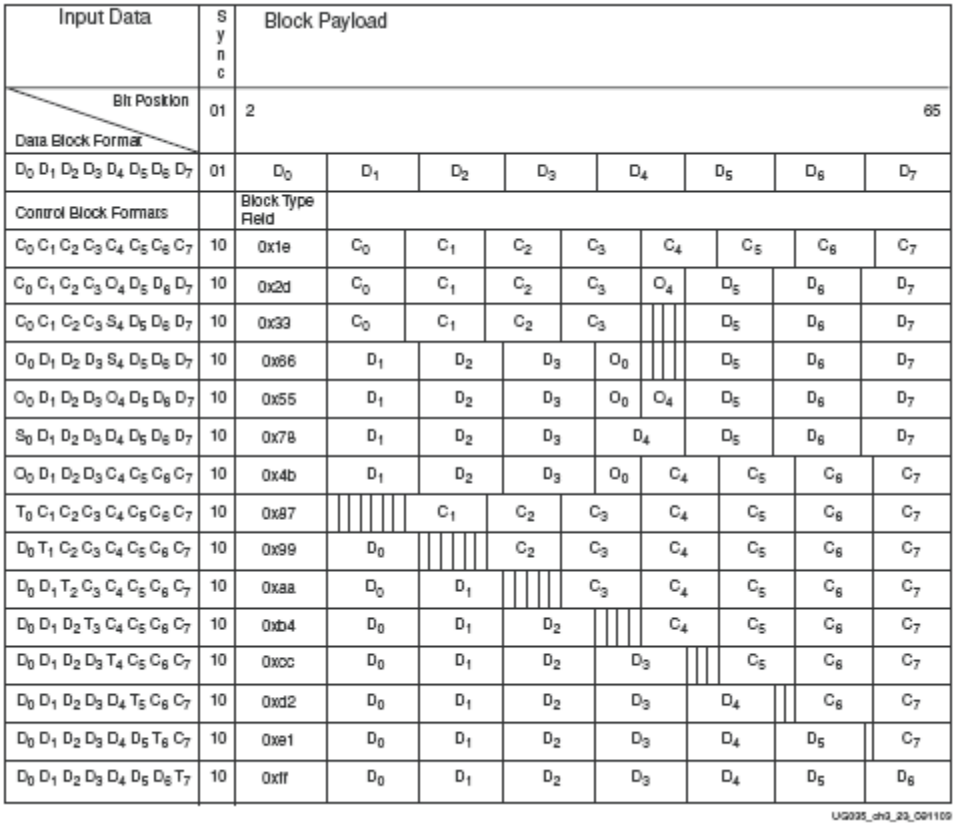


图1-10 块格式功能

控制代码具体如表1-18所示：

表1-18 控制代码

控制字符	符号	XGMII 控制代码	10GBASE-R 控制代码	10GBASE-R 0代码	8b/10b 代码
idle	/I/	0x07	0x00		K28.0 or K28.3 or K28.5
start	/S/	0xfb	由块类型域编码		K27.7
terminate	/T/	0xfd	由块类型域编码		K29.7
error	/E/	0xfe	0x1e		K30.7
Sequence ordered_set	/Q/	0x9c	由块类型域加上O模式编码	0x0	K28.4
reserved0	/R/	0x1c	0x2d		K28.0
reserved1		0x3c	0x33		K28.1
reserved2	/N/	0x7c	0x4b		K28.3
reserved3	/K/	0xbc	0x55		K28.5
reserved4		0xdc	0x66		K28.6
reserved5		0xf7	0x78		K23.7
Signal ordered_set	/Fsig/	0x5c	由块类型域加上O模式编码	0xF	K28.2

扰码器

旁路

如果TXSCRAM64B66BUSE信号为低，则扰码器被禁用。注意扰码器工作在发送FIFO的读取端。

标准操作

如果TXSCRAM64B66BUSE信号为高，则启用扰码器。扰码器使用如下的多项式

$$G(x) = 1 + x39 + x58$$

对64b/66b的有效载荷数据进行扰码。扰码器和自适应同步器(Gearbox)协同工作以完成正确的数据扰码和数据格式定义。

TXSCRAM64BB66USE	0 禁用扰码器
	1 使能扰码器

使用扰码器时，必须同时启用自适应同步器(Gearbox)（通常设置TXSCRAM64BB66USE = TXGEARBOX64B66BUSE）。

自适应同步器(Gearbox)

旁路

如果TXGEARBOX64B66BUSE信号为0，则禁用自适应同步器(Gearbox)。当使用64b/66b协议时，通常自适应同步器(Gearbox)总是被激活的。

标准操作

如果TXGEARBOX64B66BUSE信号为1，则启用自适应同步器。自适应同步器(Gearbox)需要成帧64b/66b数据并送到PMA。

TXGEARBOX64B66BUSE	0
	1 当启用扰码器和解扰码器时,其值常设为1

解码器

旁路

如果RXDEC64B66BUSE信号为低，则仅用整个64b/66b解码器。

标准操作

如果RXDEC64B66BUSE信号为高，则64b/66b解码器参照图1-7中的块格式进行解码。

如果RXIGNOREBTF信号为1，未知块类型域的块依旧被传送；如果信号为0，则发生错误时传送错误块/E/。当启用解码器时，RXCHARISK和RXC是相同的。

RXDEC64B66BUSE	0 禁用解码器
	1 启用解码器

RXIGNOREBTF	启用64b/66b解码器时的功能	禁用64b/66b解码器时的功能
	0 出现未知块类型域时传送错误块/E/	未定义
	1 出现未知块类型域时依旧正常传送	
RXCHARISK	和TXC相同	为8b/10b解码器所用

解扰码器

旁路

如果RXDESCRAM64B66BUSE信号为低，则禁用解扰码器。

标准操作

如果RXDESCRAM64B66BUSE信号为高，则启用解扰码器。解扰码器使用如下的多项式：

$$G(x) = 1 + x39 + x58$$

块同步

标准操作

块同步设计和commaDet模块联合工作。首先，32位未对齐的扰码数据从PMA传送给commaDet模块。之后，commaDet传送2比特的同步报头（或其认为可以作为同步报头的数据，基于当前的标记值）给块同步模块。test_sh的值变为1，通知块同步模块检验同步报头中的值。块同步模块分析同步报头并检验其是否合法，sh_cnt计数器增1。如果报头为非法值，sh_cnt计数器和sh_invalid_cnt计数器均增1，并且bit_slip信号出现1个周期的高电平。bit_slip信号反馈回commaDet模块，通知commaDet模块将桶形移位器移1位。移位和同步报头检查的过程一直持续到块锁定完成为止。

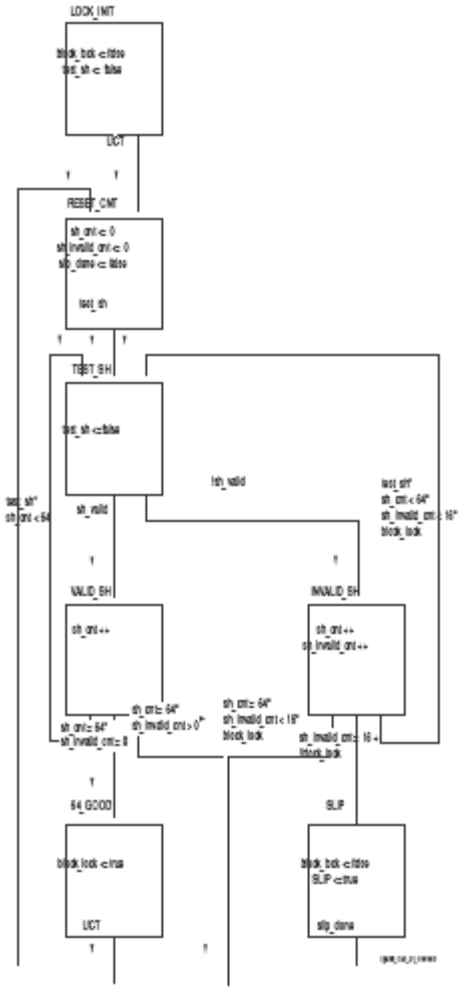

LOCK_INIT
UCT
RESET_CNT
wait_sh
TEST_SH
sh_valid
VALID_SH
INVALID_SH
64_GOOD
SLIP
UCT
slip_done

图1-11 块同步状态机

状态机通过跟踪合法和非法的同步报头来完成其工作。系统复位后，block_lock 信号为 0，此时状态为 LOCK_INIT。下一个状态是 RESET_CNT，所有的计数器清零。若 test_sh 为 1，则下一个状态是 TEST_SH，即检测同步报头的合法性。如果报头是合法的，下一个状态是 VALID_SH，否则下一个状态为 INVALID_SH。

从VALID_SH状态起，如果sh_cntis比属性sh_cnt_max小并且test_sh为1，则下一个状态为TEST_SH。如果sh_cnt等于sh_cnt_max并且sh_invalid_cnt为0，则下一个状态为GOOD_64，此时block_lock信号设置为1。之后开始重复上述过程，并且计数器均清零。

如果TEST_SHsh_cnt和sh_cnt_max相等，但是sh_invalid_cntis大于0，则下一个状态为RESET_CNT。从INVALID_SH状态起，如果sh_invalid_cnt等于sh_invalid_cnt_max或者block_lock信号为0，则下一个状态为SLIP，即bit_slip信号变为1，之后跳转到RESET_CNT状态。如果sh_cnt和sh_cnt_max相等，并且sh_invalid_cnt小于sh_invalid_cnt_max，且block_lock信号为1，此时跳转回RESET_CNT状态，而不改变block_lock或bit_slip的值。

最后，如果test_sh为高且sh_cnt 小于sh_cnt_max，sh_invalid_cnt小于sh_invalid_cnt_max，同时block_lock为1，则跳转回TEST_SH状态。需要注意的是为了完成块锁定，状态机必须连续接收到sh_cnt_max个合法同步报头，而没有接收到非法同步报头。尽管如此，一旦完成块锁定，在接收到sh_cnt_max number个合法同步报头的同时允许接收sh_invalid_cnt_max-1个非法同步报头，而不会发生失锁。可见，一旦块锁定之后，失锁的情况很难发生。

适用于所有协议的功能

时钟修正

当接收FIFO写入端的输入数据速率和读取端的输出数据速率不一致时，需要进行时钟修正。输入FIFO的输入数据速率是由RXRECCLK的频率决定的。从FIFO读取端输出的数据速率是由RXUSRCLK的频率决定的。

此处仅有一种时钟修正模式：插入/移除idle时隙的时钟修正模式。

插入/移除idle时隙的时钟修正

当CLK_COR_SEQ_DROP属性为低且CLK_CORRECT_USE属性为高时，启用插入/移除idle时隙的时钟修正。

插入/移除idle时隙的时钟修正模式：首先在比特流中寻找idle时隙，之后在这些idle时隙位置插入或者移除时隙，从而实现时钟修正。

为使插入/移除idle时隙的时钟修正常运作，用户需要对几个属性进行适当设置。CLK_COR_MAX_LAT属性设置通过接收FIFO的最大延迟。如果通过接收FIFO的延迟超过了该值，则移除idle时隙，从而使通过接收FIFO的延迟小于CLK_COR_MAX_LAT的值。

属性CLK_COR_MIN_LAT设置通过接收FIFO的最小延迟。如果通过接收FIFO的延迟小于这个值，则插入idle时隙，从而使通过接收FIFO的延迟大于CLK_COR_MIN_LAT的值。由于延迟超过CLK_COR_MAX_LAT而引起的延迟修正，修正后的延迟不会低于CLK_COR_MIN_LAT。同样，由CLK_COR_MAX_LAT引起的延迟修正，修正后的延迟不会高于CLK_COR_MAX_LAT。

时钟修正序列

时钟修正的核心工作是在比特流中搜寻idle时隙。idle时隙的搜寻启动时钟修正进程。

时钟修正电路检测的idle时隙由下列属性的低10位具体指定：

- CLK_COR_SEQ_1_1
- CLK_COR_SEQ_1_2
- CLK_COR_SEQ_1_3
- CLK_COR_SEQ_1_4
- CLK_COR_SEQ_2_1
- CLK_COR_SEQ_2_2
- CLK_COR_SEQ_2_3
- CLK_COR_SEQ_2_4

每个时钟修正序列属性的第11位决定进行8位比较还是进行10位比较。

时钟修正序列包括8个字（每个字10位），在比特流中检测这些序列。时钟修正序列的长度可以是 1 、 2 、 3 、 4 或者 8 字节。

当用户指定的修正序列长度为 1 － 4 之间时，CLK_COR_SEQ_1_*是最先首先搜索的模板。CLK_COR_SEQ_1_1是最低字节，在发送端最先发送，同时在接收端最先检测到。如果修正序列长度在 1 － 4 之间且CLK_COR_SEQ_2_USE信号为高，则由CLK_COR_SEQ_2_*定义的序列作为匹配的第二模板。在这种情况下，只要和序列 1 或序列 2 的模板中的一个相匹配，即是正确的时钟修正序列。

因为有 2 个或 3 个字节没有使用，所以CLK_COR_SEQ_MASK的相应位必须设置为逻辑 1 以表示未使用这些字节。

若用户指定的长度为 8 字节，则CLK_COR_SEQ_1_*承载前 4 个字节，而CLK_COR_SEQ_2_*承载后 4 个字节。CLK_COR_SEQ_1_1是最低字节，在发送端最先发送，同时在接收端最先检测到。此时CLK_COR__SEQ_2_USE必须设置为高。

时钟修正序列将接收数据（作为RXRECCLK的映射）和RXUSRCLK相适配，消除两者之间的频率差异。大多数的设计原型都有相应协议的默认设置。只有GT_CUSTOM设计原型允许把修正序列设置为任意类型协议。序列包括11位，其中10位为串行数据。第11位有两种不同的格式，典型的使用方式是：

- 0，不一致性错误标志，串行数据是K字符或者8位数据（在8b/10b解码之后，取决于CLK_COR_8B10B_DE）。
- 0，10位数据（没有8b/10b解码时，取决于CLK_COR_8B10B_DE）
- 1，xx，同步字符（使用64b/66b编码）
- 1，xx，8位数据值

表1-19给出了时钟修正的例子，包括11位数据的属性设置、字符值、CHARISK值和并行数据接口以及相互之间搭配。

表1-19 16位数据端口的时钟修正序列和数据间的关系

属性设置	字符	CHARISK	TXDATA (hex)
CLK_COR_SEQ_1_1 = 00110111100	K28.5	*1*	*BC*
CLK_COR_SEQ_1_2 = 00010010101	D21.4	*0*	*95*
CLK_COR_SEQ_1_3 = 00010110101	D21.5	*0*	*B5*
CLK_COR_SEQ_1_4 = 00010110101	D21.5	*0*	*B5*

注：

1. CLK_COR_8B10B_DE = TRUE。

选择合适的CLK_COR_MIN_LAT

选择CLK_COR_MIN_LAT的合适值时，应当注意满足以下几个条件：

- CLK_COR_MIN_LAT应该小于或等于12。
- CLK_COR_MIN_LAT和CLK_COR_ MAX _LAT应该是CCS/CBS长度和ALIGN_COMMA_WORD的倍数。
- 对于小于8字节的符号，(CLK_COR_MIN_LAT – CHAN_BOND_LIMIT) > 12
- 对于大于8字节的符号，(CLK_COR_MIN_LAT – CHAN_BOND_LIMIT) > 16

通道绑定

通道绑定是一种将多个串行通道绑定在一起形成一个汇聚通道的技术。在发送端，多个通道连接到同一个并行总线；在接收端是类似的，也使用同样的并行总线。最多可以将20对差分串行通道绑定在一起。很多设计原型均支持通道绑定，包括GT10_CUSTOM, GT10_INFINIBAND, GT10_XAUI, 和GT10_AURORA。

通道绑定匹配逻辑跨越字节边界搜寻CB字符，并对数据进行“comma”类型的重对齐。数据通道在复位之前均是字节扰乱的，如下例所示（额外的comma对齐不会重新对齐数据）。所以，用户在选取通道绑定字符时应当十分小心，一般最好使用不会在正常数据流中出现的特殊字符。

例如：

通道绑定字符是0x000000FF，如果发送数据序列如下：

000000FF
01020304
05060708
09000000
FF010203
04050607

结果是：

000000FF

01020304
05060708
000000FF
01020304
050607xx

绑定在一起的通道包括1个主收发器以及1-19个从收发器。收发器的CHBONDI/CHBONDO总线以菊花链的形式连接在一起，如图1-12所示。

当主收发器在数据流中检测到通道绑定对齐序列时，主收发器通过驱动CHBONDO总线通知从收发器进行通道绑定。如表1-20所示：

表1-20通道绑定对齐序列

	CHBONDO 总线
没有通道绑定	$XX000_2$
通道绑定 – 字节 0	$XX100_2$
通道绑定 – 字节 1	$XX101_2$
通道绑定 – 字节 2	$XX110_2$
通道绑定 – 字节 3	$XX111_2$

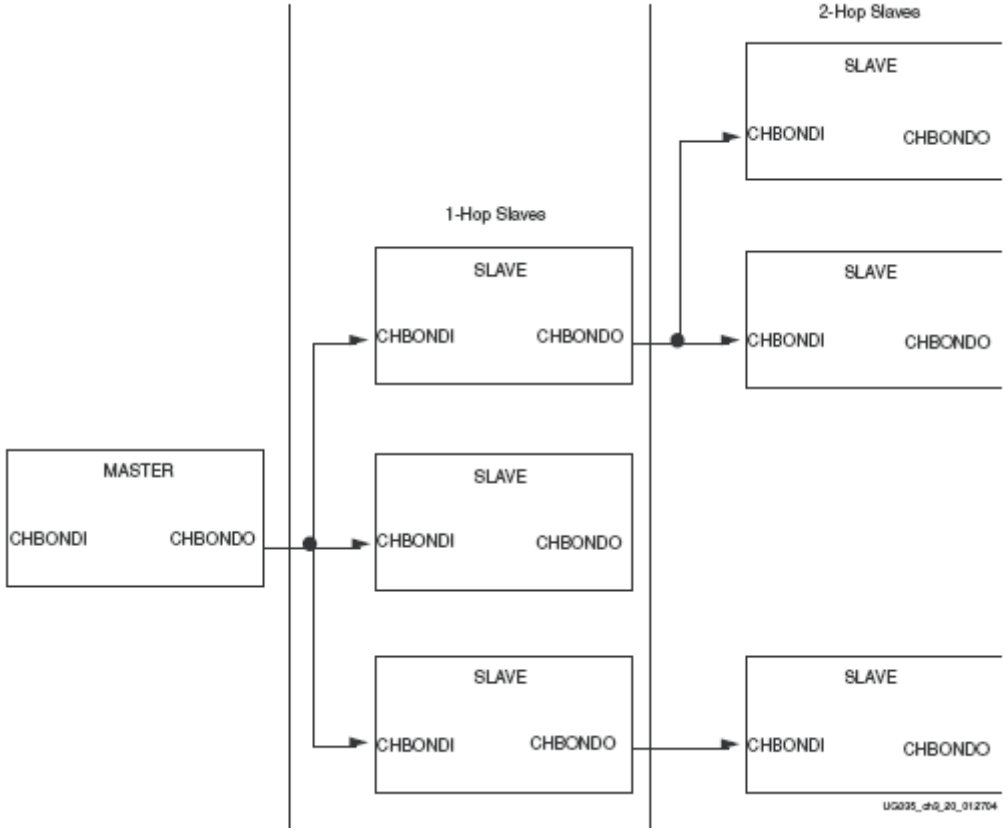


图1-12 收发器的CHBONDI/CHBONDO总线菊花链结构

无论1级从收发器还是2级从收发器，内部逻辑使得由主收发器的CHBONDO总线传送来的数据可以同时被所有的从收发器识别，并且数据是确定性的。因此，很重要的一点是，CHBONDO-to-CHBONDI的内部连接不可以含有通道进程，数据必须在一个时钟周期内从CHBONDO传送到CHBONDI。

输入到不同通道绑定收发器的数据流之间在时间上是可以有偏差的。通道绑定逻辑所允许的最大字节偏差由MC_CHAN_BOND_LIMIT属性设定。在通道绑定操作期间，从收发器通过CHBONDO总线接收主收发器的通知（指示主收发器的对齐代码位置）。如果从收发器检测到它的对齐代码位置在CHAN_BOND_LIMIT（来自主收发器）规定窗口范围之外，则从收发器不进行通道绑定并且设置通道绑定错误标志。如果通道绑定成功，从收发器输出其相对于主收发器的偏移量。偏移量和通道绑定错误标志可以通过RXBUFSTATUS总线获取。

收发器对位置和通道是有约束的，在通道绑定完成之后，这是很必要的。由于CHBONDO端口到CHBONDI端口的延迟限制，主收发器到1级从收发器间的连接可以运行在X或Y方向上，但不能同时运行在两个方向上。

图1-13中，两个1级从收发器单向连接到主收发器。若要连接到其他从收发器，则需要在X和Y两个方向上转移数据。此配置需要使用1级的菊花链，这也是2级收发器配置的基础。

图1-13和图1-14给出了XC2VPX20和XC2VPX70的通道绑定模式及连接，XC2VPX70芯片拥有更多的收发器（20个）。为了确保CHBONDO和CHBONDI间的连接的延迟满足要求，需要添加约束来检查其延时。

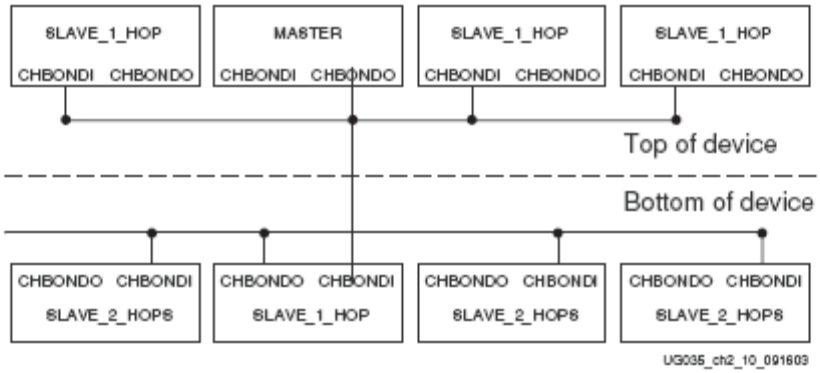


图1-13 XC2VPX20 器件结构

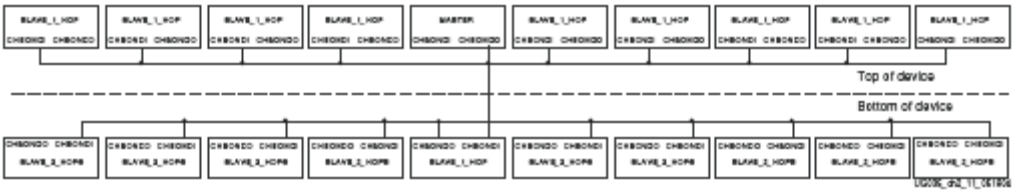


图1-14 XC2VPX70 器件结构

状态和事件总线

Virtex-II Pro X 收发器将多个信号融合在一起，从而在 Virtex-II Pro™的基础上可以提供更多的额外功能。在旧的设计中，CHBONDDONE, RXBUFSTATUS, 和 RXCLKCORCNT 这几个信号之间相互独立，分别用于指示某种状态。而在 Virtex-II Pro X 的设计中，这些信号联合在一起形成状态和事件总线。

这个联合总线有两种模式：状态模式和事件模式。在状态模式中，总线指示接收端 FIFO 的读取指针和写入指针间的差值或者最后一个通道绑定时间的偏差量。

状态指示

在状态模式中，RXBUFSTATUS 和 RXCLKCORCNT 引脚在缓冲器指针差值和通道绑定偏移量之间交替。当运行在 32 位或 40 位内部数据宽度模式时，此协议由 3 个串行时钟周期描

述（状态和数据各持续一个时钟周期）；当运行在 16 位或 20 位内部数据宽度模式时，则需要 6 个串行时钟周期（状态和数据各持续 2 个时钟周期）。

<STATUS INDICATOR> <DATA0><DATA1>

STATUS INDICATOR 指示其为指针差值还是通道绑定偏差量，DATA0 为状态数据 5：3，DATA0 为状态数据 2：0。

表 1-21 列出了指针差值状态时的各信号值，变量 pointer-Diff[5：0]保存接收端写指针和读指针之间的差值。如果 pointer-Diff[5:0] < 6’b000110，则 RXFIFO 接近下溢。如果 pointer-Diff[5:0] > 6’b111001，则 RXFIFO 接近上溢。

表1-21 指针不同状态的各信号值

Status	CHBONDDONE	RXBUFSTATUS	RXCLKCORCNT
STATUS INDICATOR	1'b0	2’b01	3’b000
DATA0	1'b0	2’b00	pointerDiff[5:3]
DATA1	1'b0	2’b00	pointerDiff[2:0]

表 1-22 给出了指示通道绑定偏差量时的各信号值。cbSkew [5:0]中保存接收读指针和写指针差值。

表1-22 指示通道绑定偏差量时的各信号值

状态	CHBONDDONE	RXBUFSTATUS	RXCLKCORCNT
STATUS INDICATOR	1'b0	2’b01	3’b001
DATA0	1'b0	2’b00	cbSkew[5:3]
DATA1	1'b0	2’b00	cbSkew[2:0]

事件指示

可以发生两种类型的事件。如表 1-23。一旦发生事件，事件指示可以覆盖状态指示。事件只能持续一个时钟周期，并由 CHBONDDONE 设置为 1 或 RXBUFSTATUS 值等于 2'b10 来指示相应事件。

表1-23指示事件时各信号值

事件	CHBONDDONE	RXBUFSTATUS	RXCLKCORCNT
Channel Bond Load	1'b1	2’b00	3’b111
Clock Correction	1'b0	2’b10	3’bxxx

事件指示可以覆盖状态指示，但当事件完成之后，状态指示继续在指针差值和通道绑定偏移量之间变化。

Verilog 示例

下面的示例代码在选用 32 位或 40 位内部数据通道的情况下，用于决定 RX 缓冲器的上溢和下溢。

```
module   status_decoder (
     RXUSRCLK2,
     DCM_LOCKED_N,
     PMARXLOCK,
     CHBONDDONE,
     RXBUFSTATUS,
     RXCLKCORCNT,

     cc_event_insert,      // Clock Correction Insertion Event
     cc_event_remove,      // Clock Correction Removal Event
     cb_event_load,        // Channel Bonding Load Event
     err_event_cc,         // Clock Correction Error Event
     err_event_cb,         // Channel Bonding Error Event

     pointerDiff,          // RX Elastic Buffer Pointer Difference
     rxbuf_almost_err,     // RX Elastic Buffer Almost Error

     cbSkew);

     input         RXUSRCLK2;
     input         DCM_LOCKED_N;
     input         PMARXLOCK;
     input         CHBONDDONE;
     input   [1:0] RXBUFSTATUS;
     input   [2:0] RXCLKCORCNT;
     output        cc_event_insert;
     output        cc_event_remove;
     output        cb_event_load;
     output        err_event_cc;
```

```
    output       err_event_cb;
    output [5:0] pointerDiff;
    output       rxbuf_almost_err;
    output [5:0] cbSkew;

////////////////////////////////////////////////////////////////////////
//
//Signal declaration
////////////////////////////////////////////////////////////////////////
//
reg         cc_event_insert;
reg         cc_event_remove;
reg         cb_event_load;
reg         err_event_cc;
reg         err_event_cb;
reg [5:0]   pointerDiff;
reg [2:0]   pointerDiff_hi;
reg [1:0]   pointerDiff_valid;
reg [5:0]   cbSkew;
reg [2:0]   cbSkew_hi;
reg [1:0]   cbSkew_valid;
reg         rxbuf_almost_err;

wire [5:0] status_event_bus;
wire [2:0] status_bus;

parameter CC_EVENT_INSERT_C = 6'b010001;
parameter CC_EVENT_REMOVE_C = 6'b010000;
parameter CB_EVENT_LOAD_C   = 6'b100111;
parameter ERR_EVENT_CC_C    = 6'b011000;
parameter ERR_EVENT_CB_C    = 6'b011001;

parameter STATUS_INDICATOR_C= 3'b001;
parameter STATUS_DATA_C     = 3'b000;

assign status_event_bus = {CHBONDDONE, RXBUFSTATUS[1],
RXBUFSTATUS[0], RXCLKCORCNT[2],
RXCLKCORCNT[1], RXCLKCORCNT[0]};
assign status_bus       = {CHBONDDONE, RXBUFSTATUS[1],
RXBUFSTATUS[0]};


////////////////////////////////////////////////////////////////////////
//
//Logic to decode events
////////////////////////////////////////////////////////////////////////
//
always @(posedge RXUSRCLK2 or posedge DCM_LOCKED_N)
  begin
```

```
    if (DCM_LOCKED_N) begin
      cc_event_insert <= 1'b0;
      cc_event_remove <= 1'b0;
      cb_event_load   <= 1'b0;
      err_event_cc    <= 1'b0;
      err_event_cb    <= 1'b0;
     end
    else begin
      cc_event_insert <= status_event_bus == CC_EVENT_INSERT_C;
      cc_event_remove <= status_event_bus == CC_EVENT_REMOVE_C;
      cb_event_load   <= status_event_bus == CB_EVENT_LOAD_C;
      err_event_cc    <= status_event_bus == ERR_EVENT_CC_C;
      err_event_cb    <= status_event_bus == ERR_EVENT_CB_C;

    end
 end


////////////////////////////////////////////////////////////////////////
//
// Logic to decode the cbSkew value and pointerDiff value
////////////////////////////////////////////////////////////////////////
//
always @(posedge RXUSRCLK2 or posedge DCM_LOCKED_N)
  begin
    if (DCM_LOCKED_N) begin
       pointerDiff_valid <=  2'b00;
       cbSkew_valid      <=  2'b00;
      end
    else if ((status_bus == STATUS_INDICATOR_C) & ~RXCLKCORCNT[2] &
~RXCLKCORCNT[1] ) begin
       pointerDiff_valid <=  {1'b0, ~RXCLKCORCNT[0]};
       cbSkew_valid      <=  {1'b0,  RXCLKCORCNT[0]};
      end
    else if (status_bus == STATUS_DATA_C) begin
       pointerDiff_valid[1] <=  pointerDiff_valid[0];
       pointerDiff_valid[0] <=  1'b0;
       cbSkew_valid[1]      <=  cbSkew_valid[0];
       cbSkew_valid[0]      <=  1'b0;
     end
   else begin // clear the valid signal if the status is interrupted
by an event.
       pointerDiff_valid <=  2'b00;
       cbSkew_valid      <=  2'b00;
     end
  end

always @(posedge RXUSRCLK2 or posedge DCM_LOCKED_N)
  begin
```

```
   if (DCM_LOCKED_N || ~PMARXLOCK) begin // reset the value to neutral
position
       pointerDiff <=  32;
       pointerDiff_hi <= 4;
       cbSkew      <=  32;
       cbSkew_hi   <=  4;
      end
   else if (status_bus == STATUS_DATA_C) begin

       if (pointerDiff_valid[0])  // register higher 3 bits
         pointerDiff_hi   <=  RXCLKCORCNT;
       else if (pointerDiff_valid[1]) // update entire register when
all 6 bits are acquired.
         pointerDiff      <=  {pointerDiff_hi , RXCLKCORCNT};

       if (cbSkew_valid[0])      // register higher 3 bits
         cbSkew_hi        <=  RXCLKCORCNT;
      else if (cbSkew_valid[1]) // update entire register when all 6
bits are acquired.
         cbSkew           <=  {cbSkew_hi , RXCLKCORCNT};
   end
 end

//////////////////////////////////////////////////////////////////////
//
// Generate RX Elastic Buffer almost error
//////////////////////////////////////////////////////////////////////
//
always @(posedge RXUSRCLK2 or posedge DCM_LOCKED_N)
 begin
    if (DCM_LOCKED_N)
      rxbuf_almost_err <= 1'b0;
    else
      rxbuf_almost_err <= (pointerDiff < 6) | (pointerDiff > 57);
end

endmodule
```

附录 B

8b/10b 列表

本信息来自RocketIO™ X收发器用户指南，想了解最新信息，请访问：
http://www.xilinx.com/cn/bvdocs/userguides/ug035.pdf

有效的数据字符和控制字符

RocketIO X 收发器8b/10b编码包括一组数据字符与K字符。8位数值被编码为10位，以保持串行线路的直流平衡。K字符是利用CHARISK指定的特殊数据字符，用于专用信息命名。表1-1和表1-2分别为有效数据字符表和有效K字符表。

表1-1: 有效数据字符

数据字节名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
D0.0	000 00000	100111 0100	011000 1011
D1.0	000 00001	011101 0100	100010 1011
D2.0	000 00010	101101 0100	010010 1011
D3.0	000 00011	110001 1011	110001 0100
D4.0	000 00100	110101 0100	001010 1011
D5.0	000 00101	101001 1011	101001 0100
D6.0	000 00110	011001 1011	011001 0100
D7.0	000 00111	111000 1011	000111 0100
D8.0	000 01000	111001 0100	000110 1011
D9.0	000 01001	100101 1011	100101 0100

1-1：有效数据字符（续）

数据字节名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
D10.0	000 01010	010101 1011	010101 0100
D11.0	000 01011	110100 1011	110100 0100
D12.0	000 01100	001101 1011	001101 0100
D13.0	000 01101	101100 1011	101100 0100
D14.0	000 01110	011100 1011	011100 0100
D15.0	000 01111	010111 0100	101000 1011
D16.0	000 10000	011011 0100	100100 1011
D17.0	000 10001	100011 1011	100011 0100
D18.0	000 10010	010011 1011	010011 0100
D19.0	000 10011	110010 1011	110010 0100
D20.0	000 10100	001011 1011	001011 0100
D21.0	000 10101	101010 1011	101010 0100
D22.0	000 10110	011010 1011	011010 0100
D23.0	000 10111	111010 0100	000101 1011
D24.0	000 11000	110011 0100	001100 1011
D25.0	000 11001	100110 1011	100110 0100
D26.0	000 11010	010110 1011	010110 0100
D27.0	000 11011	110110 0100	001001 1011
D28.0	000 11100	001110 1011	001110 0100
D29.0	000 11101	101110 0100	010001 1011
D30.0	000 11110	011110 0100	100001 1011
D31.0	000 11111	101011 0100	010100 1011
D0.1	001 00000	100111 1001	011000 1001
D1.1	001 00001	011101 1001	100010 1001
D2.1	001 00010	101101 1001	010010 1001

表1-1：有效数据字符（续）

数据字节名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
D3.1	001 00011	110001 1001	110001 1001
D4.1	001 00100	110101 1001	001010 1001
D5.1	001 00101	101001 1001	101001 1001
D6.1	001 00110	011001 1001	011001 1001
D7.1	001 00111	111000 1001	000111 1001
D8.1	001 01000	111001 1001	000110 1001
D9.1	001 01001	100101 1001	100101 1001
D10.1	001 01010	010101 1001	010101 1001
D11.1	001 01011	110100 1001	110100 1001
D12.1	001 01100	001101 1001	001101 1001
D13.1	001 01101	101100 1001	101100 1001
D14.1	001 01110	011100 1001	011100 1001
D15.1	001 01111	010111 1001	101000 1001
D16.1	001 10000	011011 1001	100100 1001
D17.1	001 10001	100011 1001	100011 1001
D18.1	001 10010	010011 1001	010011 1001
D19.1	001 10011	110010 1001	110010 1001
D20.1	001 10100	001011 1001	001011 1001
D21.1	001 10101	101010 1001	101010 1001
D22.1	001 10110	011010 1001	011010 1001
D23.1	001 10111	111010 1001	000101 1001
D24.1	001 11000	110011 1001	001100 1001
D25.1	001 11001	100110 1001	100110 1001
D26.1	001 11010	010110 1001	010110 1001
D27.1	001 11011	110110 1001	001001 1001
D28.1	001 11100	001110 1001	001110 1001

表1-1：有效数据字符（续）

数据字节名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
D29.1	001 11101	101110 1001	010001 1001
D30.1	001 11110	011110 1001	100001 1001
D31.1	001 11111	101011 1001	010100 1001
D0.2	010 00000	100111 0101	011000 0101
D1.2	010 00001	011101 0101	100010 0101
D2.2	010 00010	101101 0101	010010 0101
D3.2	010 00011	110001 0101	110001 0101
D4.2	010 00100	110101 0101	001010 0101
D5.2	010 00101	101001 0101	101001 0101
D6.2	010 00110	011001 0101	011001 0101
D7.2	010 00111	111000 0101	000111 0101
D8.2	010 01000	111001 0101	000110 0101
D9.2	010 01001	100101 0101	100101 0101
D10.2	010 01010	010101 0101	010101 0101
D11.2	010 01011	110100 0101	110100 0101
D12.2	010 01100	001101 0101	001101 0101
D13.2	010 01101	101100 0101	101100 0101
D14.2	010 01110	011100 0101	011100 0101
D15.2	010 01111	010111 0101	101000 0101
D16.2	010 10000	011011 0101	100100 0101
D17.2	010 10001	100011 0101	100011 0101
D18.2	010 10010	010011 0101	010011 0101
D19.2	010 10011	110010 0101	110010 0101
D20.2	010 10100	001011 0101	001011 0101
D21.2	010 10101	101010 0101	101010 0101
D22.2	010 10110	011010 0101	011010 0101

表1-1：有效数据字符（续）

数据字节名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
D23.2	010 10111	111010 0101	000101 0101
D24.2	010 11000	110011 0101	001100 0101
D25.2	010 11001	100110 0101	100110 0101
D26.2	010 11010	010110 0101	010110 0101
D27.2	010 11011	110110 0101	001001 0101
D28.2	010 11100	001110 0101	001110 0101
D29.2	010 11101	101110 0101	010001 0101
D30.2	010 11110	011110 0101	100001 0101
D31.2	010 11111	101011 0101	010100 0101
D0.3	011 00000	100111 0011	011000 1100
D1.3	011 00001	011101 0011	100010 1100
D2.3	011 00010	101101 0011	010010 1100
D3.3	011 00011	110001 1100	110001 0011
D4.3	011 00100	110101 0011	001010 1100
D5.3	011 00101	101001 1100	101001 0011
D6.3	011 00110	011001 1100	011001 0011
D7.3	011 00111	111000 1100	000111 0011
D8.3	011 01000	111001 0011	000110 1100
D9.3	011 01001	100101 1100	100101 0011
D10.3	011 01010	010101 1100	010101 0011
D11.3	011 01011	110100 1100	110100 0011
D12.3	011 01100	001101 1100	001101 0011
D13.3	011 01101	101100 1100	101100 0011
D14.3	011 01110	011100 1100	011100 0011
D15.3	011 01111	010111 0011	101000 1100
D16.3	011 10000	011011 0011	100100 1100

表1-1：有效数据字符（续）

数据字节名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
D17.3	011 10001	100011 1100	100011 0011
D18.3	011 10010	010011 1100	010011 0011
D19.3	011 10011	110010 1100	110010 0011
D20.3	011 10100	001011 1100	001011 0011
D21.3	011 10101	101010 1100	101010 0011
D22.3	011 10110	011010 1100	011010 0011
D23.3	011 10111	111010 0011	000101 1100
D24.3	011 11000	110011 0011	001100 1100
D25.3	011 11001	100110 1100	100110 0011
D26.3	011 11010	010110 1100	010110 0011
D27.3	011 11011	110110 0011	001001 1100
D28.3	011 11100	001110 1100	001110 0011
D29.3	011 11101	101110 0011	010001 1100
D30.3	011 11110	011110 0011	100001 1100
D31.3	011 11111	101011 0011	010100 1100
D0.4	100 00000	100111 0010	011000 1101
D1.4	100 00001	011101 0010	100010 1101
D2.4	100 00010	101101 0010	010010 1101
D3.4	100 00011	110001 1101	110001 0010
D4.4	100 00100	110101 0010	001010 1101
D5.4	100 00101	101001 1101	101001 0010
D6.4	100 00110	011001 1101	011001 0010
D7.4	100 00111	111000 1101	000111 0010
D8.4	100 01000	111001 0010	000110 1101
D9.4	100 01001	100101 1101	100101 0010

表1-1：有效数据字符（续）

数据字节名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
D10.4	100 01010	010101 1101	010101 0010
D11.4	100 01011	110100 1101	110100 0010
D12.4	100 01100	001101 1101	001101 0010
D13.4	100 01101	101100 1101	101100 0010
D14.4	100 01110	011100 1101	011100 0010
D15.4	100 01111	010111 0010	101000 1101
D16.4	100 10000	011011 0010	100100 1101
D17.4	100 10001	100011 1101	100011 0010
D18.4	100 10010	010011 1101	010011 0010
D19.4	100 10011	110010 1101	110010 0010
D20.4	100 10100	001011 1101	001011 0010
D21.4	100 10101	101010 1101	101010 0010
D22.4	100 10110	011010 1101	011010 0010
D23.4	100 10111	111010 0010	000101 1101
D24.4	100 11000	110011 0010	001100 1101
D25.4	100 11001	100110 1101	100110 0010
D26.4	100 11010	010110 1101	010110 0010
D27.4	100 11011	110110 0010	001001 1101
D28.4	100 11100	001110 1101	001110 0010
D29.4	100 11101	101110 0010	010001 1101
D30.4	100 11110	011110 0010	100001 1101
D31.4	100 11111	101011 0010	010100 1101
D0.5	101 00000	100111 1010	011000 1010
D1.5	101 00001	011101 1010	100010 1010
D2.5	101 00010	101101 1010	010010 1010
D3.5	101 00011	110001 1010	110001 1010

表1-1：有效数据字符（续）

数据字节名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
D4.5	101 00100	110101 1010	001010 1010
D5.5	101 00101	101001 1010	101001 1010
D6.5	101 00110	011001 1010	011001 1010
D7.5	101 00111	111000 1010	000111 1010
D8.5	101 01000	111001 1010	000110 1010
D9.5	101 01001	100101 1010	100101 1010
D10.5	101 01010	010101 1010	010101 1010
D11.5	101 01011	110100 1010	110100 1010
D12.5	101 01100	001101 1010	001101 1010
D13.5	101 01101	101100 1010	101100 1010
D14.5	101 01110	011100 1010	011100 1010
D15.5	101 01111	010111 1010	101000 1010
D16.5	101 10000	011011 1010	100100 1010
D17.5	101 10001	100011 1010	100011 1010
D18.5	101 10010	010011 1010	010011 1010
D19.5	101 10011	110010 1010	110010 1010
D20.5	101 10100	001011 1010	001011 1010
D21.5	101 10101	101010 1010	101010 1010
D22.5	101 10110	011010 1010	011010 1010
D23.5	101 10111	111010 1010	000101 1010
D24.5	101 11000	110011 1010	001100 1010
D25.5	101 11001	100110 1010	100110 1010
D26.5	101 11010	010110 1010	010110 1010
D27.5	101 11011	110110 1010	001001 1010
D28.5	101 11100	001110 1010	001110 1010
D29.5	101 11101	101110 1010	010001 1010

表1-1：有效数据字符（续）

数据字节名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
D30.5	101 11110	011110 1010	100001 1010
D31.5	101 11111	101011 1010	010100 1010
D0.6	110 00000	100111 0110	011000 0110
D1.6	110 00001	011101 0110	100010 0110
D2.6	110 00010	101101 0110	010010 0110
D3.6	110 00011	110001 0110	110001 0110
D4.6	110 00100	110101 0110	001010 0110
D5.6	110 00101	101001 0110	101001 0110
D6.6	110 00110	011001 0110	011001 0110
D7.6	110 00111	111000 0110	000111 0110
D8.6	110 01000	111001 0110	000110 0110
D9.6	110 01001	100101 0110	100101 0110
D10.6	110 01010	010101 0110	010101 0110
D11.6	110 01011	110100 0110	110100 0110
D12.6	110 01100	001101 0110	001101 0110
D13.6	110 01101	101100 0110	101100 0110
D14.6	110 01110	011100 0110	011100 0110
D15.6	110 01111	010111 0110	101000 0110
D16.6	110 10000	011011 0110	100100 0110
D17.6	110 10001	100011 0110	100011 0110
D18.6	110 10010	010011 0110	010011 0110
D19.6	110 10011	110010 0110	110010 0110
D20.6	110 10100	001011 0110	001011 0110
D21.6	110 10101	101010 0110	101010 0110
D22.6	110 10110	011010 0110	011010 0110
D23.6	110 10111	111010 0110	000101 0110

表1-1：有效数据字符（续）

数据字节名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
D24.6	110 11000	110011 0110	001100 0110
D25.6	110 11001	100110 0110	100110 0110
D26.6	110 11010	010110 0110	010110 0110
D27.6	110 11011	110110 0110	001001 0110
D28.6	110 11100	001110 0110	001110 0110
D29.6	110 11101	101110 0110	010001 0110
D30.6	110 11110	011110 0110	100001 0110
D31.6	110 11111	101011 0110	010100 0110
D0.7	111 00000	100111 0001	011000 1110
D1.7	111 00001	011101 0001	100010 1110
D2.7	111 00010	101101 0001	010010 1110
D3.7	111 00011	110001 1110	110001 0001
D4.7	111 00100	110101 0001	001010 1110
D5.7	111 00101	101001 1110	101001 0001
D6.7	111 00110	011001 1110	011001 0001
D7.7	111 00111	111000 1110	000111 0001
D8.7	111 01000	111001 0001	000110 1110
D9.7	111 01001	100101 1110	100101 0001
D10.7	111 01010	010101 1110	010101 0001
D11.7	111 01011	110100 1110	110100 1000
D12.7	111 01100	001101 1110	001101 0001
D13.7	111 01101	101100 1110	101100 1000
D14.7	111 01110	011100 1110	011100 1000
D15.7	111 01111	010111 0001	101000 1110
D16.7	111 10000	011011 0001	100100 1110
D17.7	111 10001	100011 0111	100011 0001

表1-1：有效数据字符（续）

数据字节名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
D18.7	111 10010	010011 0111	010011 0001
D19.7	111 10011	110010 1110	110010 0001
D20.7	111 10100	001011 0111	001011 0001
D21.7	111 10101	101010 1110	101010 0001
D22.7	111 10110	011010 1110	011010 0001
D23.7	111 10111	111010 0001	000101 1110
D24.7	111 11000	110011 0001	001100 1110
D25.7	111 11001	100110 1110	100110 0001
D26.7	111 11010	010110 1110	010110 0001
D27.7	111 11011	110110 0001	001001 1110
D28.7	111 11100	001110 1110	001110 0001
D29.7	111 11101	101110 0001	010001 1110
D30.7	111 11110	011110 0001	100001 1110
D31.7	111 11111	101011 0001	010100 1110

列表1-2:有效控制“K”字符

专用代码名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
K28.0	000 11100	001111 0100	110000 1011
K28.1	001 11100	001111 1001	110000 0110
K28.2	010 11100	001111 0101	110000 1010
K28.3	011 11100	001111 0011	110000 1100
K28.4	100 11100	001111 0010	110000 1101
K28.5	101 11100	001111 1010	110000 0101
K28.6	110 11100	001111 0110	110000 1001
K28.7(1)	111 11100	001111 1000	110000 0111
K23.7	111 10111	111010 1000	000101 0111

列表1-2:**有效控制“K”字符*(续)***

专用代码名称	HGF EDCBA位	Current RD – abcdei fghj	Current RD + abcdei fghj
K27.7	111 11011	110110 1000	001001 0111
K29.7	111 11101	101110 1000	010001 0111
K30.7	111 11110	011110 1000	100001 0111

注：

1. 仅用于测试与特性报告。

附录 C

两种不同的 FPGA-to-FPGA 数据链路的比较

作者：Carl Christensen

Thomson Multimedia 公司 2002 年 11 月 6 日

摘要

本文针对同一系统中两种截然不同的 FPGA-to-FPGA 数据链路设计进行了详尽的比较。文中详细介绍了如何使用 Xilinx RocketIO™ SERDES 来实现高速数据链路。此高速链路运行在 3.125 Gb/s 速度下，并使用 8b/10b 编码，从而在不同底板的 FPGA 之间通过数米的电缆实现了有效载荷速率高达 2.5Gb/s 的传输。文中同时还描述了在同样的系统中如何来实现速率较低的数据链路，通过使用双倍数据速率（double data rate，DDR）LVDS 并配合时钟的传送，链路可以在两个处于相同底板但是不同 PCB 板上的 FPGA 之间以 156MHz 的时钟速率实现 400Mb/s 的有效载荷传输。本文的内容涉及到 PCB/模拟设计的一些关键点，但主要内容还是集中在数字实现、综合、仿真、接口、PAR 和时序约束这几部分上。

系统开发开始后，对于设计师而言高速数据链路看上去会比较难实现，而低速链路会简单些。尽管在设计中两种链路都成功实现了，但是最难实现的数据链路却是出人意料的。

简介

当我们开始为新产品开发系统结构时，我们认识到我们的新产品需要两种截然不同的数据链路。一种数据链路用于满足约 400Mb/s 的数据传输需求，另一种链路则用于约 2.5Gb/s 的数据传输。较慢的数据链路用于 FPGA 间的连接，每个 FPGA 位于不同的 PCB 板上但在同一块底板上。高速数据链路也是用于连接两个 FPGA，但是两个 FPGA 位于不同的底板上。随着开发的开始，高速数据链路看上去会比较难实现，而低速链路会简单些。

链路要求及系统结构的关键因素/功能

虽然两种链路有不同的要求，但是两者的功能集是类似的，功能集对研发过程中各种不同的解决方案会提供不同的优缺点。

低速链路

种种的迹象显示低速链路要相对简单些。低速链路工作在约 400Mb/s 的速率下而且不需要额外的电缆线路，同时还有许多选项可供选择，包括：典型的并行总线结构、双倍数据速率、或者时钟转发的双倍数据速率（CF_DDR）。为了简化时钟分配和同步，我们可以选择时钟转发的双倍数据速率(CF_DDR)，同时鉴于低摆幅差分信号的各种优点，我们选择低压差分信号（low voltage differential signaling，LVDS）作为信号标准。

单一时钟域

我们最初的计划是使用频率在 110 － 120MHz 之间的时钟来运行我们的低速链路，但是随着设计开发的推进，我们觉得使用单一时钟域可能会更好些。尽管如此，单一时钟域仍然要求所有的逻辑电路和 CF_DDR 链路工作在 156MHz。Xilinx 有一个 CF_DDR 的参考设计，其工作频率远高于 156MHz，而且我的 Xilinx 现场应用工程师（FAE）觉得提高速率并不会是个太大的问题。我们的现场应用工程师现在关心的问题是：由于在一个单元中有 19 条这种低速链路，但并没有足够的时钟管理器，所以我们没法使用全局时钟资源或者时钟管理器(DCM)。所以我们开发了数个使用单一时钟域而不涉及 DCM 的设计，因为我们知道单一时钟域对于时序约束和布局布线（PAR）来说是相当优越的，而且易于工程师们进行设计。

分帧要求

我们需要定义一种方法用于标记数据帧的开始以及区分数据帧和控制帧。CF_DDR 链路需要的数据比特数大于 1，但又不需要全部的 2 比特。我们可以开发或者借用一种编码方法来利用分帧产生的额外带宽，但是想到还有为系统保留的连接器引脚，所以似乎没有理由不再添加 1 比特。第 3 位用于指示其他的 2 位是数据还是控制指令，而且第 3 位的传输可以实现数据和控制指令的分帧（如图 0-1 所示，cmd）。

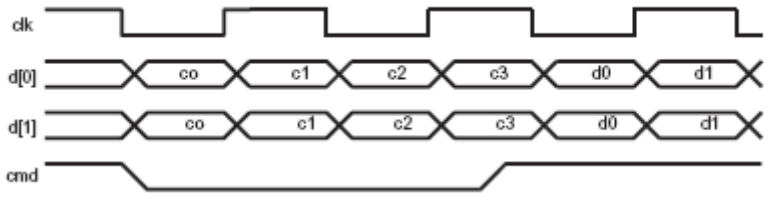


图 0-1 分帧要求

高速链路

我们对高速链路有更多的担心，所以我们更多地分析、探讨高速链路的开发。当时我们希望使用即将推出的 Virtex-II Pro FPGA 中内部集成的 SERDES，但是我们觉得那个时候采用这种方式会给我们的设计带来很大风险。Virtex-II Pro FPGA 还没有正式发布。而且我们还

有许多疑问：Xilinx 是否把这些专门的模拟电路集成到其工艺过程中？Virtex-II Pro 是否能足够快地完成并上市，从而得以应用在我们的设计中呢？我们在研究的过程中，还观察了其他一些选择，包括一些网络链路和基于光纤的链路。尽管如此，我们的所有研究结果表明：我们还是应该选用简单的时钟嵌入链路或者带有专有协议的 SERDES。

成本考虑

使用 Virtex-II Pro FPGA 还可以帮助我们的设计满足另一个要求：低成本。虽然此时对我们的产品进行定价是为时过早的，但是 Xilinx 的承诺常常在我的脑海中出现：“RocketIO™ 和 PowerPC™是免费的”。我可以使用 Virtex-II Pro 器件并同时拥有一个集成的 SERDES，或者以同样的价格购买可编程逻辑器件然后再购买一个外部的 SERDES。从成本的角度来看，答案是显而易见的，但是从风险和时间安排的角度来看，这个选择却没有那么诱人。所以当时我想也许我们就使用外部的 SERDES。那样的话，就算我们使用 Mindspeed™，以后可能也可以用 Virtex-II Pro FPGA 来代替。

集成的 SERDES 对比外部的 SERDES

我们彻底研究了使用外部的 SERDES 的方案后，我们发现集成了内部 SERDES 的 Virtex-II Pro 器件反而更有吸引力了，这是因为集成 SERDES 内部的 I/O 数量和相关的同步转换输出功能很有优势。我们的系统需要在 FPGA 间建立 19 条低速 400Mb/s 链路和 4 条高速链路。图 0-2 给出了可能的集成 SERDES 方案的示意图，而图 0-3 给出了使用外部 SERDES 方案的示意图。

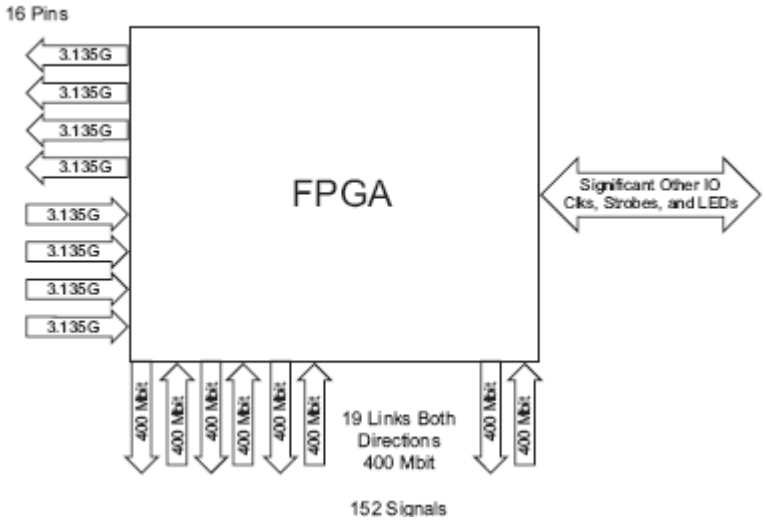


图 0-2　集成 SERDES

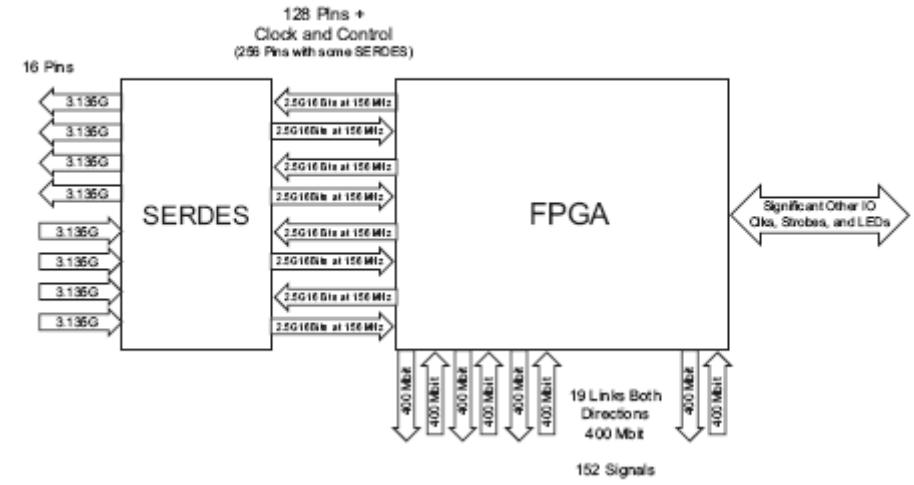


图 0-3　外部 SERDES

输入/输出要求

设计的 I/O 要求越变越苛刻。如果每个低速链路需要 4 对线，则其内部需要 152 个引脚，而且 SERDES 还需要额外的 128 根数据线以及控制线。所以 I/O 需要超过 300 个引脚，而且我们还没有算上其他的 I/O 连接，例如时钟、复位、时序选通和 LED 等。我们开始为各种不同的 I/O 标准决定存储分配引脚数，但是不久之后我们就发现了这种方法行不通。

同步转换输出

被推荐使用的 FPGA 内部更大的潜在问题出现在 I/O 环上。Xilinx（以及其他所有 FPGA 厂家）在设计大型 FPGA 时作了折衷。一个折衷是不提供足够的 I/O 环电源和地连接，以提供在 SSO 环境下对所有 I/O 的支持。部分因设计而定的特殊电路可能会恶化这个问题，从而会带来更多的挑战。输出流（多数的外部 SERDES 是 4 条 16 位总线）中通常包含的是同一份数据。因此，如果一个输出正在转换，那么很可能四个输出同时都在转换。

分帧要求

高速链路同样需要对齐、分帧和转换，以此来保证时钟恢复。显然，我们可以选择使用 8b/10b 编码和解码来保证为嵌入时钟恢复提供足够的切换，并且为分帧提供“K”字符。但是这种方式在带来方便的同时，需要付出的代价是链路的物理速度由 2.5Gb/s 增加到 3.125Gb/s。这可能也会是个问题。但是，基于所有这些好处，而且 8b/10b 编码器和解码器均集成在千兆位级收发器（MGT）内部，我们开始研究如何通过电缆传输 3.125Gb/s 信号。

电缆、板、背板和连接器

我们需要使用铜线来实现至少 3 米的背板到背板通信。另一个有趣的要求是安置 FPGA 的 PCB 必须是可热拔插的（从系统单元前端），而且所有的互联电缆必须从系统单元的背部输入。

Xilinx 一开始并不能确保这种方法是可行的，因为他们只在仅带 2 个连接器的数英寸厚的 FR4 PC 板上进行了测试。不管怎样，Xilinx 告诉我如果我们希望尝试新的方案，他们也愿意进行测试。幸运的是，我有一个可用的解决方案—当时有一些工业界最优秀的高速信号操作工程师和我一起工作，而且他们可以设计我所需要的东西。他们在将高精确度的串行数字视频信号传输通过连接器、背板和电缆这方面已经有多年的经验。但是这些信号的速率只有 1.5Gb/s，所以还是会有潜在的长时间的无切换时隙。频谱通常被有趣地称作"DC to light"，它是个外表令人讨厌的信号。我需要使用更高的速率以及由 8b/10b 编码/解码提供的有保证转换。我的高速工程专家们很有信心可以让电缆和连接器工作得相当完美，但是他们很担心数字噪声会进入模拟电路。现在看来值得庆幸的是，我们最终决定尝试这种方法。

实现细节

系统结构已经就位，现在应该开始写 Verilog HDL 来实现我们的硬件设计。

低速链路

CF_DDR 链路的编程并不是相当困难的，但是也不像我们所预期的那么容易。我们一开始计划使用 Xilinx 网站上的参考设计作为起点，但是参考设计使用的是结构化设计原型而不是行为描述方法。我们不想去实例化设计原型，而且我们懂得寄存器传输语言（Register Transfer Language，RTL）Verilog，同时能够使用工具调通程序。

随着设计的开始，问题开始增加。我们怎样保证输入/输出块（IOB）的 DDR 功能的使用？怎样才能使 LVDS I/O 标准适用？不同综合工具的方法可能会稍有不同。

最后，一些设计原型的实例是必需的。在我们的设计中，经过实例化而并非只是推断的设计原型如表 1 中所列。

表 C-1 实例化过的设计原型

RAM16X1D
FDDRRSE
IBUFDS
OBUFDS

不久后我们就使用 RTL 完成了设计并且通过了仿真验证。同时，我们还仔细研究了综合的结果，确定我们使用了 IOB 的 DDR 功能。单发送器和单接收器的运转告诉我们：156MHz 的时钟速率还是比较容易接合的。至此，我们已经完成了我们的设计，并向公司内部发布以提供给需要此模块的各种设计使用。

高速链路

我们并不确定在我们的设计中对 MGT 进行编程时会出现什么问题。当我们开始我们的设计时，MGT 产品还没有正式发布而且各种运转资料是无法获得的。我们只能把希望寄托于参考 Mindspeed 的资料文档。但是当我们发现 Xilinx 已经开发了一种新的针对 Mindspeed 内核的数字接口时，我们唯一能做的事就是等待我们能获得的资料。最后，我接到了我的 Xilinx

代表的电话。Xilinx 可以提供 一些数据给我，但是要我做好心理准备，因为资料非常多。先前我就知道 PowerPC 是相当复杂的而且它有许多文档，但是我提醒他我并不是要使用 CPU 内核。他笑着回答我说：单独 MGT 部分的文档就超过 1000 页。我挂断电话之后，就意识到我们 MGT 部分的设计会是一个艰巨的任务。MGT 是一个复杂的内核而且我们需要很好地理解它。我知道我必须找一个学识渊博的设计人员来全职研究 MGT。

开发程序代码本身是很简单的。我们只需要剪切和粘贴一些样本实例，再进行部分修改以满足我们设计的需求即可。更困难的事情是设置如此多的参数和配置端口，还好并没有我预料中的那么困难。如果我们计划使用一种预定义的标准例如 XAUI，那么事情会更简单。Xilinx 已经为各种预定义的标准设置好所有的参数，但是我们是针对定制应用，所以还需要花点功夫。

仿真同样也有一些改变。MGT 的最初模型是 SWIFT 或者加密后的模型。但是随着模型升级到 Model Sim SE，同时作为 MGT 开发套件一部分的脚本也有一些修改，我们就可以对我们的 3Gb/s 链路进行仿真了。

证明我们的想法

由于存在如此多的风险，我们决定制作一块验证板来实现和证明我们的方法。伴随着一些反思，我们决定同时也制作一个低速链路的原型。

就在验证板临近完工的时候，我们收到 Xilinx 的通知，Xilinx 不能象预期中的那么早提供 Virtex-II Pro 器件给我们。尽管如此，我们还是决定制作一些验证板来测试其他的一些高风险电路。我们本来认为对工程师来说是很简单的数据链路却突然变的困难了。

测试低速链路

我们使用工具进行速率验证显得有些太过匆忙了，而且我们发现结果是有缺陷的。我们使用了简单的没有任何偏差量的周期约束，而没有特别考虑半周期通道的问题，同时还认为周期约束会解决任何半周期问题，但实际情况却不是这样的。原型链路不能工作，在 156MHz 和 96MHz 均不行。我们看得越细致，问题就变得越困难。

在其他器件空缺的电路中测试这些链路确实是很让人苦恼的。布局器（placer）会把逻辑电路布置在每个地方、任何地方。当我们察看参考设计的详细资料时，我们发现了 I/O 区块。由于它们在文档中没有很好的介绍，所以很难确定它们是如何工作的。我们还试图帮助布局器（placer）完成正确的时序约束设置。本以为会很简单的事情，却变得十分困难。

输入/输出区块和引脚布局

随着对 I/O 区块的逐步理解，我们发现广告中所宣称的 DDR 速率是可以达到的，但是只有在单独的手工设计解决方案中才能实现。这样的链路在我们的普通设计方案（带时序驱动 PAR 的 RTL，选用方便板极布线的 I/O 放置方式）中是不能达到那么高的运行速度的。

输入/输出区块是由数个特定的组构成，每个组是 4 个相邻的 IOB 以及一些特殊的附加高速互连线。I/O 区块只出现在器件的两个竖边，并且只以 4 个一组的形式出现。在过去的几年里，我们都是基于逻辑/数据流来选取引脚。但是随着引脚数和布线量的增加，所以我们现在通常让 PCB 设计师来选取这些引脚。图 1-1 是一个由逻辑/数据流选择的引脚示例。图 1-2 是由 PCB 设计师选择的引脚示例。

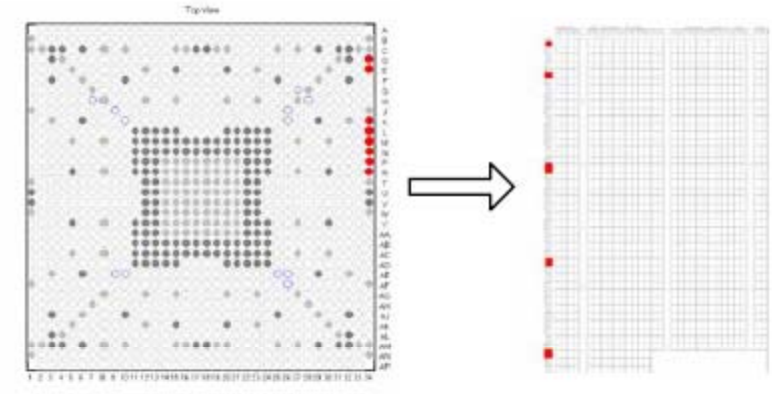


图 1-1 基于逻辑/数据流的引脚选取

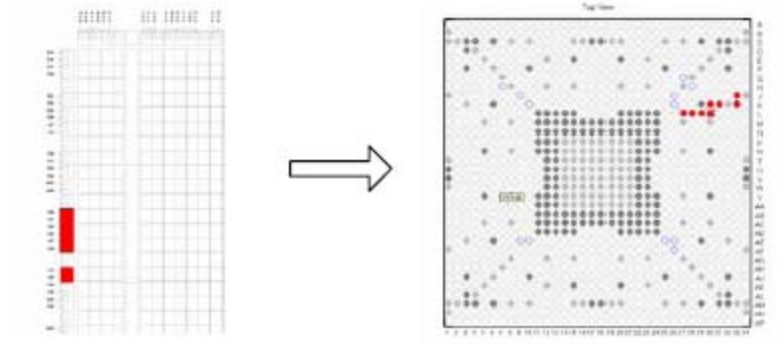


图 1-2 由 PCB 设计师选择的引脚

这通常并不是个棘手的问题，但是如果我们希望 I/O 在同一个 I/O 区块中，那么必须在相邻的 IOB 中。幸运的是，我们的部分 I/O 确实在竖边的相邻 IOB 中。但是这并不会有太大的帮助，哪怕我们只有四个信号（三个数据一个时钟），因为我们使用了 LVDS。由于使用了差分信号，所以在一个区块中只能有两个信号。我们担心会发生下面的事情：如果 2 个信号在同一个区块中而另 2 个信号不在同一区块，此时的偏差问题是否会比各信号都在不同区块中的情况更严重？最后，我们决定忽略 I/O 区块问题，而选取相邻的 IOB 以改善局部时钟安排。

局部时钟

我们不能使用全局时钟资源、全局缓存（BUFG）或 DCM 来作为伴随数据传送的时钟，因为我们在单个 FPGA 中需要 19 条低速链路。局部时钟安排不合理是导致原型失败的真正原因。

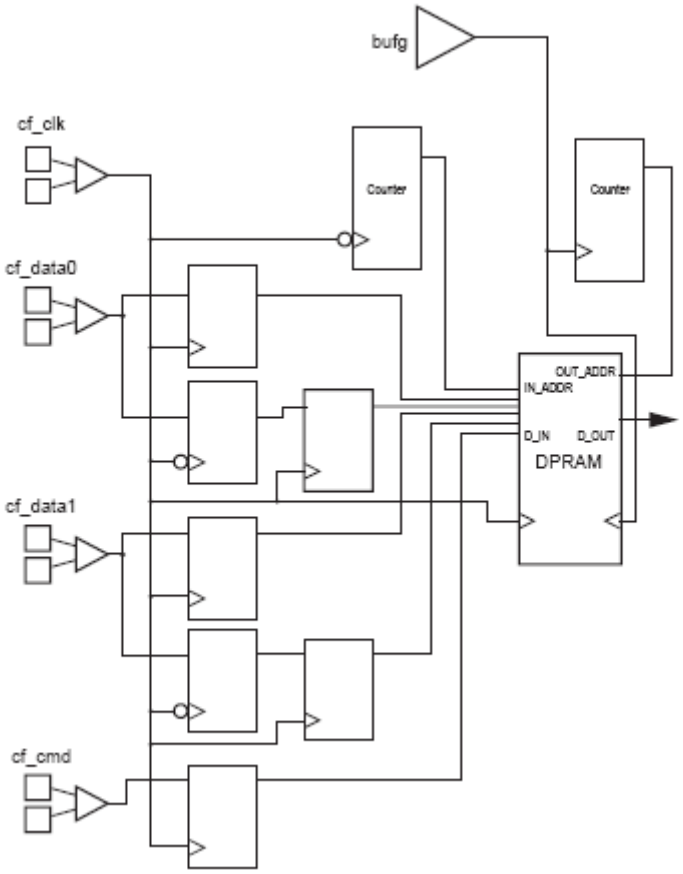


图 1-3 ： 新的结构框图

除了 CF_DDR 链路和相关的测试电路，我们设计中的其他部分并没有很多逻辑电路，但是布局器（placer）却试图将逻辑电路分布在整个芯片上。当我们开始将逻辑电路分布调整合理时，我们发现仅仅控制逻辑电路的分布显然是不够的。我们需要改变逻辑电路的结构，简化的框图如图 1-3 所示。

Arbuckle 方法

在 CF_DDR_RX 结构的设计中需要有一个特殊的技巧。我们称之为 Arbuckle 方法，因为它是由 Arbuckle 最先在 Xilinx 的设计中发现的，Xilinx 的时钟域比全局时钟缓冲器还要多。局部时钟的问题是偏差。如果没有全局时钟缓冲器，两个相关的触发器的边沿很容易偏差过多，从而导致逻辑错误。我们通常认为这个问题的起因是其中的一个触发器过早触发。这个

问题的解决方法是添加一个由相反边沿触发的触发器来管理时钟，从而确保数据不会比时钟快。在 CF_DDR_RX 中实现 Arbuckle 方法是很复杂的，图 1-4 中给出了简化的示意框图。

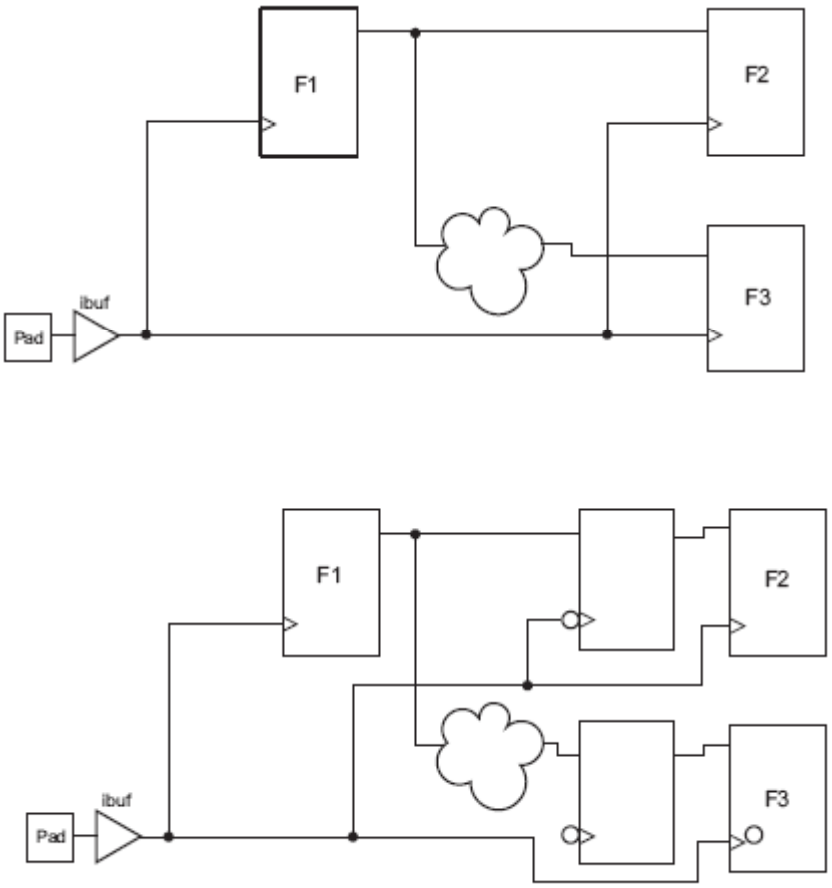


图 1-4 简化的示例

在图 1-4 的第一个框图中，如果时钟到达 F1 和 F2（或者 F3）的偏差高于 F1 的输出到达 F2（或者 F3）的延迟，则电路会出现错误操作。在图 1-4 的第二个框图（代表 Arbuckle 方法）中，添加了负边沿触发的触发器来保证 F1 的输出到达 F2 和 F3 时不会快于时钟。这种方法对长期进行严格同步设计的设计师来说可能会有点奇怪，但是它确实起作用。

双倍数据速率的困惑

我们在低速链路设计中还会遇到的一个问题是：需要涉及时钟的两个边沿。多位的 DDR 很难调试，因为数据通常是混合、交叉和令人困惑的。

时钟传送_双倍数据速率发送器

整体来说，发送器是设计中比较简单的部分。其结构基本就是标准的 DDR 输出。发送器的主要问题是数据和时钟的相位调整。我们最初计划是使传送时钟的相位和数据相差 90 度（图 1-5）。这样时钟边沿会出现在数据稳定的区域内。

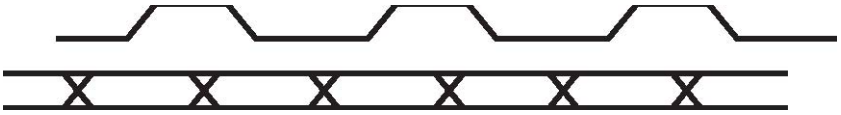


图 1-5 异相的时钟和数据

问题出现在接收器上。单一扇出的数据引脚和较高扇出的时钟线之间的偏差会导致触发器出现问题，而且问题会一直保持。我们曾尝试使用 DCM 的移相器，但最终还是决定同相传送时钟和数据（图 1-6）。

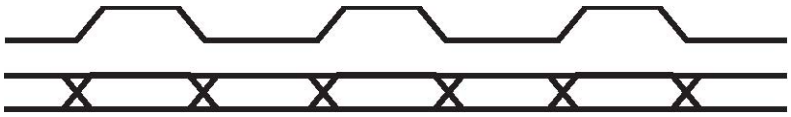


图 1-6 相位的时钟和数据

测试高速链路

基于 MGT 的链路在许多方面都要简单些，包括时序约束，布局和检验。用户时钟的周期约束是唯一的时序约束。布局器（placer）和固定内核协同工作时不会出现不对齐布局的情况。而且，由于它是一个硬内核，所以我需要关心的唯一问题就是接口配置。接口配置是比较简单的，因为数据经过解串之后被传送到有组织的 16 位总线上。

其他 Thomson 多媒体设计工程师也考虑在公司其他产品中使用 Virtex-II Pro MGT，他们向我咨询了我们使用这些器件的开发进程。他们很关心我们从获得实验原型板到实际可以在两卡之间传送数据所花费的开发时间。他们很想知道是“数天还是数周”？我告诉他们完成这个任务只花费了数个小时。实际上，如果我们相信我们的接收指示灯，而不花费那么多时间去连接逻辑分析仪来观察接收图形的话，那么整个任务只需要数分钟。它确实是马上就可以实现的！时钟修正最初工作不正常，所以后来我们修改了加重参数和其他一些选项。但是，从根本上说，在我的初次尝试中确实通过铜线电缆在两板之间实现了 3.123Gb/s（有效载荷速率 2.5Gb/s）的数据传输。

高速链路的问题和解决方案

我并没有说 MGT 设计是不会遇到问题的。可能出现的问题包括工具问题、文档问题和其他一些意料不到的问题。但是，因为我们已经有所准备，所以我们在开发过程中可以很快地适应并解决这些问题。

差分 BREF 时钟引脚

就在我们要订制原材料板之前，发生了一件意料不到的事情，已经完成的板设计被暂时搁置。我的 Xilinx FAE 打电话告诉我们如果希望运行在 3.2Gb/s，我们必须注意特殊的时钟引脚。显然，这里面包括直接进入 MGT 的特殊低偏差时钟（BREFCLK）。问题是我们中没有人知道哪些引脚由 BREFCLK 提供时钟的。我们的设计因此搁置了两天，这两天里我们沿着神秘的 BREF 时钟引脚进行了追踪。

我们最终确定了 BREF 的引脚数，但是 BREF 的问题并没有就此结束。BREF 引脚很难识别，因为 Xilinx 认为这些引脚是不需要的。实际上，Xilinx 已经从最初版本的 Virtex-II Pro FPGA 支持工具中删除了针对 BREF 通道的支持。幸运的是，还保留了一个插件，但是这个插件和多数 EDA 插件不同。除了运行脚本，在每次检查 PAR 时都需要不断的在 FPGA 编辑器上点击。虽然最后我们还是解决了这个问题，但是我们多少会有点失望。

混乱的文档

混乱的文档起初确实让我们有点失望。尽管如此，最混乱的还是时钟修正和 comma/“K”字符问题。这些问题多数并不是错误，但是，其措词确实是十分混乱的。

参数管理

MGT 另一个让人失望的地方是在 Verilog 中的参数管理。一共需要设置 46 个参数，这确实要费不少劲。而且由于不同的综合工具和仿真的语法是不同的，所以很有可能出现仿真条件和实验室中不一样的情况。但我们从未真正有过问题，因为我们已经有心理准备。

大型的设计原型中应该去掉那些不是注释的注释，并且应该定义所有工具都支持的参数，就象语言中的定义一样。下面是 4 种工具中 2 种工具的语法示例。注意第一句必须是独立的一行。

```
/* synthesis xc_props = "ALIGN_COMMA_MSB=TRUE, CHAN_BOND_MODE=OFF,
CHAN_BOND_OFFSET=0000, CHAN_BOND_ONE_SHOT=FALSE,
CHAN_BOND_SEQ_1_1=10001000001, CHAN_BOND_SEQ_1_2=10001000010,
CHAN_BOND_SEQ_1_3=10001000011, CHAN_BOND_SEQ_1_4=10001000100,
CHAN_BOND_SEQ_2_1=10001001001, CHAN_BOND_SEQ_2_2=10001001010,
CHAN_BOND_SEQ_2_3=10001001011, CHAN_BOND_SEQ_2_4=10001001100,
CHAN_BOND_SEQ_2_USE=FALSE, CHAN_BOND_SEQ_LEN=2, CHAN_BOND_WAIT=5,
CLK_CORRECT_USE=TRUE, CLK_COR_INSERT_IDLE_FLAG=FALSE,
CLK_COR_KEEP_IDLE=FALSE, CLK_COR_REPEAT_WAIT=00000,
CLK_COR_SEQ_1_1=00110111100, CLK_COR_SEQ_1_2=00010010101,
CLK_COR_SEQ_1_3=00010111100, CLK_COR_SEQ_1_4=00010110101,
CLK_COR_SEQ_2_1=10010001001, CLK_COR_SEQ_2_2=10010001010,
CLK_COR_SEQ_2_3=10010001011, CLK_COR_SEQ_2_4=10010001100,
CLK_COR_SEQ_2_USE=FALSE, CLK_COR_SEQ_LEN=4, COMMA_10B_MASK=1111111000,
DEC_MCOMMA_DETECT=TRUE, DEC_PCOMMA_DETECT=TRUE, DEC_VALID_COMMA_ONLY=FALSE,
MCOMMA_10B_VALUE=1100000000, MCOMMA_DETECT=FALSE, PCOMMA_10B_VALUE=0
```

```
11111000, PCOMMA_DETECT=TRUE, RX_BUFFER_SE=TRUE, RX_DATA_WIDTH=2,
RX_DECODE_USE=TRUE, RX_LOSS_OF_SYNC_FSM=FALSE, TERMINATION_IMP=50,
SERDES_10B=FALSE, TX_BUFFER_USE=TRUE, TX_DATA_WIDTH=2,
TX_DIFF_CTRL=400, TX_PREEMPHASIS=3" */;

// XST Primitive Attributes
// synthesis attribute ALIGN_COMMA_MSB          of gb_transceiver_m
is "TRUE"
// synthesis attribute CHAN_BOND_MODE           of gb_transceiver_m
is "OFF"
// synthesis attribute CHAN_BOND_OFFSET         of gb_transceiver_m
is "0000"
// synthesis attribute CHAN_BOND_ONE_SHOT       of gb_transceiver_m
is "FALSE"
// synthesis attribute CHAN_BOND_SEQ_1_1        of gb_transceiver_m
is "10001000001"
// synthesis attribute CHAN_BOND_SEQ_1_2        of gb_transceiver_m
is "10001000010"
// synthesis attribute CHAN_BOND_SEQ_1_3        of gb_transceiver_m
is "10001000011"
// synthesis attribute CHAN_BOND_SEQ_1_4        of gb_transceiver_m
is "10001000100"
// synthesis attribute CHAN_BOND_SEQ_2_1        of gb_transceiver_m
is "10001001001"
// synthesis attribute CHAN_BOND_SEQ_2_2        of gb_transceiver_m
is "10001001010"
// synthesis attribute CHAN_BOND_SEQ_2_3        of gb_transceiver_m
is "10001001011"
// synthesis attribute CHAN_BOND_SEQ_2_4        of gb_transceiver_m
is "10001001100"
// synthesis attribute CHAN_BOND_SEQ_2_USE      of gb_transceiver_m
is "FALSE"
// synthesis attribute CHAN_BOND_SEQ_LEN        of gb_transceiver_m
is "2"
// synthesis attribute CHAN_BOND_WAIT           of gb_transceiver_m
is "5"
 // synthesis attribute CLK_COR_INSERT_IDLE_FLAG  of gb_transceiver_m
 is "FALSE"
 // synthesis attribute CLK_COR_KEEP_IDLE        of gb_transceiver_m
 is "FALSE"
 // synthesis attribute CLK_COR_REPEAT_WAIT      of gb_transceiver_m
 is "00000"
 // synthesis attribute CLK_COR_SEQ_1_1          of gb_transceiver_m
 is "00110111100"
 // synthesis attribute CLK_COR_SEQ_1_2          of gb_transceiver_m
 is "00010010101"
 // synthesis attribute CLK_COR_SEQ_1_3          of gb_transceiver_m
 is "00010111100"
 // synthesis attribute CLK_COR_SEQ_1_4          of gb_transceiver_m
 is "00010110101"
```

```
// synthesis attribute CLK_COR_SEQ_2_1          of gb_transceiver_m
is "10010001001"
// synthesis attribute CLK_COR_SEQ_2_2          of gb_transceiver_m
is "10010001010"
// synthesis attribute CLK_COR_SEQ_2_3          of gb_transceiver_m
is "10010001011"
// synthesis attribute CLK_COR_SEQ_2_4          of gb_transceiver_m
is "10010001100"
// synthesis attribute CLK_COR_SEQ_2_USE        of gb_transceiver_m
is "FALSE"
// synthesis attribute CLK_COR_SEQ_LEN          of gb_transceiver_m
is "4"
// synthesis attribute COMMA_10B_MASK           of gb_transceiver_m
is "1111111000"
// synthesis attribute DEC_MCOMMA_DETECT        of gb_transceiver_m
is "TRUE"
// synthesis attribute DEC_PCOMMA_DETECT        of gb_transceiver_m
is "TRUE"
// synthesis attribute DEC_VALID_COMMA_ONLY     of gb_transceiver_m
is "FALSE"
// synthesis attribute MCOMMA_10B_VALUE         of gb_transceiver_m
is "1100000000"
// synthesis attribute MCOMMA_DETECT            of gb_transceiver_m
is "FALSE"
// synthesis attribute PCOMMA_10B_VALUE         of gb_transceiver_m
is "0011111000"
// synthesis attribute PCOMMA_DETECT            of gb_transceiver_m
is "TRUE"
// synthesis attribute RX_BUFFER_USE            of gb_transceiver_m
is "TRUE"
// synthesis attribute RX_DATA_WIDTH            of gb_transceiver_m
is "2"
// synthesis attribute RX_DECODE_USE            of gb_transceiver_m
is "TRUE"
// synthesis attribute RX_LOSS_OF_SYNC_FSM      of gb_transceiver_m
is "FALSE"
// synthesis attribute SERDES_10B               of gb_transceiver_m
is "FALSE"
// synthesis attribute TERMINATION_IMP          of gb_transceiver_m
is "50"
// synthesis attribute TX_BUFFER_USE            of gb_transceiver_m
is "TRUE"
// synthesis attribute TX_DATA_WIDTH            of gb_transceiver_m
is "2"
// synthesis attribute TX_DIFF_CTRL             of gb_transceiver_m
is "400"
// synthesis attribute TX_PREEMPHASIS           of gb_transceiver_m
is "3"
```

比预期更困难的板级设计

基于 MGT 的数据链路的另一挑战就是板级设计，它比预期要难得多。板的阻抗需要特别小心控制。对于我这个 FPGA 人员来说，支线（包括由于连接不同层的过孔而产生的支线）、电源相关的特殊要求以及旁路等等都是很陌生的。但是高速专家们知道应该怎么做。

结论

我们最初的估计是，基于 MGT 的链路设计要比传送时钟的低速链路难上几个数量级。但实际上，两者的工作量差不多是相同的。

传送时钟的多位 DDR 是两个 FPGA 之间中等速率传输的一种合理方式，同样也适用于全局时钟资源可以作为传送时钟的情况。如果希望提高速度或者避免使用全局时钟资源，那么还需要不少额外的工作。

由于有更少的引脚，更高的速度以及其他一些优点，集成的 SERDES 可能会是更好的解决方案。我们还会继续使用 MGT 链路，同时建议 Xilinx 将来可以同时提供高速和低速 SERDES。

致谢

在此向 Thomson 公司的 Lynn Arbuckle、Dave Bytheway 和 Randall Redondo 致以诚挚的谢意。他们完成了文中所述的绝大部分的实际工作。Barry Albright 和 Marc Walker 是高速设计专家，他们同样来自 Thomson 公司。同时还要感谢 Jeff Hutchings 和 Tim Hemphill 的大力支持，eff Hutchings 是我的 Xinlinx FAE，Tim Hemphil 是我的 Xilinx 销售代表。最后还要感谢我的技术编辑，同时也是我最亲爱的妻子 Sandy Christensen，感谢她对我的支持。

附件D

术语表

4b/5b：类似于 8b/10b 编码器，但是要简单些。顾名思义，这种机制将4个比特编码成5个比特。4b/5b的编码器和解码器要比8b/10b简单一些。但是4b/5b的控制字符要少一些，并且不能处理直流平衡和不一致性问题。由于同步开销相同但是功能却比较少，4b/5b编码机制并不经常使用。

64b/66b：一种为10G以太网开发的新型线路编码机制，它使用了带有非扰码同步字符和控制字符的扰码方式。

8b/10b：是IBM开发的，已经被广泛采用。8b/10b编码机制是一种数值查找类型的编码机制，它将8位的字转化为10位符号。这些符号可以保证有足够的切换用于时钟恢复。

AC耦合：通过串联电容器将接收器连接到发射器的方法。修正接收器和发射器之间的直流偏置差值。

有源均衡器：**与信号**频率关联的放大器/衰减器。

寻址/交换/转发：如果串行协议是针对点对点的应用，则不需要寻址方案。而更复杂的协议通常含有寻址方案，在寻址的基础上可以实现转发和交换。

Advanced Switching：**一种**交换结构协议，基于同 PCI Express 相同的物理和数据级协议。是交换结构领域的新兴标准，将会成为重要的标准。

对齐序列：串行流中一种独特的位模式，可用于确定字和符号的对齐。经常包括一对保留符号，该符号可形成在串行流其他位置不可能产生的独特位序列。

ASIC：专用集成电路。

ASSP：专用标准产品。

ATCA：ATCA 的全称是 Advanced Telecom Computing Architecture，即高级电信计算架构，又称作 AdvancedTCA。这个 PICMG 标准是为下一代电信设备制定的。它的目标旨在提供一个十分灵活、可扩展的系统的同时，尽量简化不同制造商生产的设备之间的互操作。针对某个主题，此标准可以提供多种实现方案，包括星形连接结构的背板、冗余星形连接结构的背板和全连接的网状结构。

Aurora：Aurora是一个相对简单的协议，只控制链路层和物理层。

后钻孔：在电镀后，过孔未用部分通过钻孔方法来消除。

背板：计算机机箱背面的公用总线，将各个电路卡槽连接到系统的其他部分，如计算机主板。它也向各个插槽提供经过滤波与未滤波的低压交流和直流电源。作为很结实的电路板，背板可以支持更高的连接速度和更多的逻辑。它们用于大型网络交换机和路由器。

BERS：误码率。

BGA：球形栅格阵列（有时缩写为BG）。与插脚栅格阵列（PGA）相反，球形栅格阵列是一种微芯片连接方法。球形栅格阵列芯片通常使用一组呈同心矩形排列的焊接点或球，来连接到电路板。

桥接器：在基于（OSI 参考模型）数据链路层信息的网段之间转发数据的设备。这些网段有共同的网络层地址。桥接器可以连接不同种类的网络（如无线局域网到以太网）。

BUFG： 全局缓冲器。
Cat 5：一种双绞线等级，这种双绞线通常安装在办公楼内。
通道绑定：吸收两个或多个MGT之间的偏差，将数据提交给用户，就像只使用一条链路进行传输一样。
CLK：时钟。计算机中的一种电路，利用石英晶体产生一串规则脉冲并发送到CPU。时钟信号是计算机的心跳。时钟发送脉冲时，计算机内部进行开关操作。时钟速度越快，计算机每秒可执行的指令越多。
时钟修正和通道绑定：修正发送时钟和接收时钟之间的偏差，同时也可实现多通道的歪斜修正。（通道绑定是可选的，并不一定包含在SERDES中。）
时钟域： 由同一时钟源锁定的逻辑电路称为处于同一个时钟域内。
转发时钟：转发时钟（cf）或时钟转发是源同步的另一个术语。
时钟管理器：管理各种时钟操作，包括时钟倍频、时钟分频、时钟恢复。
时钟恢复：从输入比特流的比特翻转中进行时钟恢复操作。
时钟树：集成电路中的走线和驱动器，用于在特定的歪斜规格范围内将时钟分配给逻辑电路。
CML：电流模式逻辑；一种基于差分的电气接口，很适合于千兆位链路。
CMOS：互补金属氧化物半导体。
comma字符：用于表示对齐序列的一个或两个符号。该序列通常可以在收发器中设置，但在某些情况下，也可以预定义。
宇宙射线：用于描述几种高能量电磁辐射的通用术语，包括伽马射线以及 α 和 β 粒子辐射。
CRC：循环冗余码校验；确定数据发送错误的一种方法。
串扰：由于另一信号的干扰而导致的信号质量下降。
CSMA/CD：载波监听多路访问/冲突检测。
数据格式：视频和音频协议的值定义；如何通过 0 和 1 来代表特定的值或特定含义。
数据报：包含自身地址信息的数据包，它可以独立地从源端发送到目的计算机。
数据提取： 协议的一个通常功能是定义如何将数据和头部分离。这个功能通常称作数据提取或者反嵌入。
DCA：数字通信分析仪；包含有采样示波器，同时还提供很多其他功能。
直流平衡：比特流的平均直流电压。如果比特流的“1”多于“0”，那么比特流将有正直流平衡。通常，理想的比特流是零直流平衡（相同数量的“1”和“0”）。
直流偏置：发射器或接收器工作的固定直流电源。
DC耦合：通过直接连接（而不是通过交流耦合中使用的串联电容器）将接收器连接到发射器的方法。接收器和发射器的直流偏置必须相同。
DCM：数字时钟管理器。
DDR：双数据速率（见RAM和寄存器）。
去加重：每次比特翻转的起始处过量驱动，或欠量驱动任何连续相同的比特位。（预加重的别名。）
解串器：将速率为n*y的串行数据转变成速率为y的n位宽的并行数据。

确定性抖动：由具体眼图或者事件确定的抖动。引起确定性抖动的来源包括：不对称的上升/下降时间、字符交叉干扰、电源馈电、振荡器、来自其他信号的串扰。通常也简称 DJ 或 dj。

数字通信分析仪：DCA —一种采样示波器，添加了对分析数据通信信号有用的功能。

数字存储型示波器：将输入信号转化为数字采样并保存，最后使用这些采样数据在显示器上重建信号。

差分信号：使用正负线路对的信号。信号值由该线路对的两个信号之间的电压差值确定。差分信号比非差分或单端信号具有更强的抗干扰能力。

DLL：延迟锁定环（也称为数字延迟锁定环）

丢弃：在FIFO存储寄存器快满时，在下一个时钟校正序列中不会将输入数据序列写入存储器内，数据被“丢弃”。

DSP：(1) 数字信号处理。利用计算机处理信号，如声音、视频以及已经转换成数字形式的其他模拟信号。DSP的用途：对调制解调器发出的调制信号进行解码，以不同的方式处理声音、视频和图像，并获知声纳、雷达和地震读数中的数据。（2）数字信号处理器。专用于数字信号处理的CPU。数字信号处理器的用途是支持调制解调器和声卡。

EDIF：电子设计交互格式

ef：帧结束

EISA：早期PC使用的总线架构。

嵌入：协议通常还定义怎样将数据嵌入到协议流或分组中。对于遵从协议栈模型的协议而言，这个功能是很必要的。

EMI：电磁浸入。辐射电磁能量通常受到政府和标准组织制定的辐射标准的控制。

ESR：等效串联电阻。不希望有，但确实存在的电容阻抗。

ESL：等效串联电感。不希望有，但确实存在的电容电感。

以太网：普通的局域网协议。

眼图：数字采样示波器上通常显示的波形。眼图用于指示信号质量。抖动、阻抗匹配以及幅度都可以用眼图来刻画。

FEC：**前向纠错—**添加额外的位，用于帮助恢复错误数据。

铁氧体磁珠：用铁氧体材料磁珠制造而成的感应器，通常用于抑制或滤除某一频率范围的能量。

FiberChannel：FiberChannel 一直以来都是串行协议，不过其速度在不断增加。随着铜线链路的改进，FiberChannel 即可以用于光纤通道也可以用于铜线通道。

Field Solver：一种工具/数学模型，可确定特定阻抗的实际大小。

FIFO：先进先出（与LIFO-后进先出相反） 。一种数据结构或硬件缓冲器，数据项按照它们进入时的相同次序出来。

FireWire（火线）：高性能串行总线以前的名称。苹果计算机公司和德州仪器公司联合开发的一种串行总线（IEEE 1394）。高性能串行总线可按照树形菊花链布局连接多达63个器件，发送数据的速度可达到400Mbps。它支持即插即用和对等通信。

Flip-chip（倒装片）：一种表面安装芯片技术，芯片封装在板上，然后用环氧树脂填满下面。一种通用的固定方法是将焊球安放在芯片上，将芯片“倒装”到板上，然后熔化焊料。

流量控制：协议中往往还定义流量控制。流量控制的内容很多，包括动态缩放子通道的带宽分配，以及调整空闲时隙的插入速率以满足时钟修正的需要。

Fmax：特定技术或产品中，触发器的最高翻转率。

FOLS：光纤局域网部门（电信行业协会）。

FR-4：一种普通的PCB 组装材料。

分数相位检测器： 一种相位检测器，在相位检测中利用同一时钟的多个相位。
HSTL：高速晶体管逻辑。
IBIS 模型：基于文字的电路行为描述。在 1GHz 以下足够精确。不提供电路结构的具体资料。
IC几何尺寸：根据多边形的大小或尺寸对集成电路进行分类的方法。实例包括0.25微米和90纳米。
空闲符号或序列：没有发送任何数据时的符号。
IEEE 1394 FireWire™： 高性能串行总线，适用于即插即用和对等网络。
Infiniband：一个 box-to-box 的协议，可以运行在铜线或者光纤上。Infiniband 类型的电缆广泛应用于短距离（几米）的千兆位级链路。此协议适用于多种设备，而其支持很大的复杂性。协议内还定义了中继器、交换机和集线器的规范，以增加可连接设备的数目。Infiniband 还可以用于使用 Infiniband 交换和控制台的复杂系统配置。
ILD：注入式激光二极管。
阻抗：电容、电感和阻抗对信号产生的综合效应。根据欧姆定律，电压是在特定的频率条件下电流和电阻的乘积。
阻抗度量电压经过电阻时，电流所遇到的阻力。
阻抗的测量单位是欧姆，是电压与容许电流的比值。
实例化：编程中，通过用值（或其他变量）替代变量，产生更确定的版本。如同逻辑编程一样，这意味着将逻辑变量（类型变量）绑定到某些值（类型）。
IP：知识产权（如IP核）。
ISA：一种很早以前的PC总线架构。
码间干扰（ISI）：符号间的干扰－如果串行流包含有多个位时间的相同数值数据，而其后跟着短（1或2）位时间的相反数据数值时，则会发生符号间干扰。
等时同步：频率匹配但是相位不一定匹配。
抖动：完美的过零交叉和实际的过零交叉之间的差别。
LFSR：线性反馈移位寄存器。
线路编码器：将线路上的编码数据分解成原始数据。（这是一个可选模块，编码可能在SERDES外完成。）
线路编码器机制：将数据编码成适应不同线路的格式。编码器通常会消除长的无转变位的序列，同时还可以平衡数据中0、1的出现次数。（这是一个可选模块，某些SERDES可能没有。）
线路编码方案：一种数据编码方法，以便在串行流上进行传输。
LVCMOS：低压互补金属氧化物半导体。
LVDS： 低压差分信令。
LVTTL：低压晶体管到晶体管逻辑。
包：一种确切定义的字节集合，包括头部、数据和尾部。
无源均衡器：无源电路，其频率响应可以补偿传输衰减，和滤波器类似。
MAC （媒体接入控制）：OSI数据链路层的较低子层。节点的逻辑链路控制和网络的物理层之间的接口。对于不同的物理介质，MAC不一样。
微带：带形细传输线，用于传输微波频率；通常安装在平面介电基质上，该介电基质安装在地平面上。
MII：独立于媒体的接口。
MSB：最高有效位。在二进制数中，这是最左边的位，具有最高的加权系数。
OC：光学载波，是SONET规范中定义的传输速度。OC定义光学设备的传输，STS定义电子设备的传输。

OC-192：光学载波192 (10 Gb/s).
PAR： 布局和布线。
PCB组装：组装的电路板。
PCI：外设部件互连。英特尔公司设计的个人计算机局部总线，工作频率为33MHz，支持即插即用功能。它提供与外设的高速连接，允许连接7个外设。它主要用于奔腾计算机，但独立于处理器，因此能够与其他处理器一起工作。它插入主板上的PCI插槽内，可以与ISA或EISA总线一起使用。
PCI Express：由旧的并行 PCI 结构改进得到的快速串行结构。上层的协议依旧是兼容的，可以很容易适配到旧的 PCI 系统。
PCMCIA：笔记本电脑常用的总线架构。
物理接口： 协议具体定义驱动电平、预加重等等，以确保各器件间的兼容性。
物理层接口：处理信号的物理/电气特性的协议部分，也称为PHY。
PICMG：PICMG 是一个超过 600 家公司的联盟，该联盟致力于制定高性能标准化背板结构的开放标准。PICMG 的很多标准使用其它的工业标准，例如 PCI 和 Infiniband。

PLL：锁相环是一种根据参考时钟和输入信号而能够产生锁定于输入信号的新时钟的电路。

功率完整性工具： 这些工具有助于设计功率输送（旁通、滤波等）系统。
许多是信号完整性分析工具的附件。如同SI工具为信号提供的功能一样，这些工具为功率系统提供同类功能。
PPM：百万分之一；用来描述非常小的比率。
预加重：切换起始处的有意过量驱动。
全内反射原理：如果入射角大于临界值，则光线不会透射而会全部反射回来。
随机抖动：随机抖动是由差分模式和普通模式的随机噪声引起，例如电源噪声和热噪声。随机抖动也称作 rj，RJ 或者不确定性抖动。
接收FIFO：在接收数据被提取之前，暂时保存数据。在需要时钟修正的系统中，接收FIFO是必须的。
接收线路接口：模拟接收电路，包括差分接收器，还可能包括有源或者无源均衡电路。
Reed-Soloman：一种流行且有效的向前纠错法。
重复：如果FIFO接近于空，下一次发现时钟校正序列时，它将向FIFO进行两次写入操作。
上升时间：信号从0转变为1所花费的时间。
ROGERS 3450：一种用于组装电路板或PCB的材料。在高频条件下，该特殊材料比大多数材料的损耗都小。
RS-232：推荐标准232。这是通过PC串行端口进行通信的实际标准。它可以指支持RS-232标准的电缆和端口。
RTL：寄存器传送级别/语言（软件）。一种硬件描述语言（HDL），用于描述计算机或数字电子系统的寄存器以及这些寄存器之间进行的数据传送方式。用于有众多寄存器的机器的一种中间码。
S-参数：基于文字的描述，可以描述频率很高的电路、板上走线或者连接器。最初用在微波设计中，现在 s-参数常用于对高速板和连接器组合进行更有效的建模。s-参数描述在传输线中微波传输的散射和反射。
采样示波器：将信息数字化并保存。如果信号的速度超出了 AD 转换器所能支持的范围，则示波器在每个周期内只采样很少的样点。通过逐次移动采样，示波器可以获取足够的信息以代表重复性的信号。
扰码：一种将数据重新排列或者编码以使其随机化的方法，但必须能解扰恢复。

SCSI：小型计算机系统接口；一种并行总线架构，现在主要用于快速硬盘驱动器。早期版本用于小型计算机系统内部和外部，以实现扩展。
SDH：同步数字体系；是在光网络上传输数字信息的国际标准。它是指ANSI标准的ITU条款—SONET。

自同步：两块芯片之间的通信，其中发送芯片产生的数据流同时包括数据和时钟信息。

Serial RapidIO：旧的并行协议的串行版本，RapidIO 相当灵活，可以用于多协议间的接口（例如 PCI 和 Infiniband）。
串行器：将速率为y的n位宽并行数据转变成速率为n*y速率的串行数据。
sf: 帧开始
信号完整性：若要获得信号完整性，信号必须是可独立的（可重复和可预测的）。信号也必须是真实或纯粹的，并且没有被破坏。
信号完整性（Signal Integrity，SI）分析器：信号完整性分析器常常是 PCB 布局工具的一个可选组件。信号完整性分析器可以分析 PCB 布局的信号完整性。通常分析器还允许通过添加连接器和电缆模型来进行多 PCB 系统的分析。另一个很有用的功能是分析器支持 IC 模型，特别是千兆位级的收发器。
单端信令：利用信号线进行的信息传输。
SNA型同轴电缆：一种小型连接器电缆系统，经常用于较高的频率。
SONET：同步光网络；使不同厂商的电信产品能够在高速光纤网络上进行通信。
源同步：两个IC间进行通信时，发送IC生成一个伴随发送数据的时钟信号。接收IC利用该转发时钟进行数据接收。
SPICE 模型：基于文字描述的电路行为描述。十分精确，而且能够展示电路结构的详细资料。
SPICE 仿真器：在 MGT 模拟仿真和分析中，SPICE 仿真器的主要作用是作为 SI 分析工具的行为模型引擎。通常 MGT 制作商会以 SPICE 模型的形式提供行为模型，但是由于 SPICE 模型可以从本质上很好地描述电路，所以大部分人都采用加密的 SPICE 模型。这些 SPICE 模型通常需要使用为 IC 开发专门设计的高端 SPICE 工具。
SSTL：残余串行终结逻辑电路。
层叠：组成印刷电路板的铜和玻璃纤维层的属性、厚度和位置规范。
带状线：PCB上的可控阻抗传输线路，包括外层上的迹线和邻近层上的基准面。
强耦合：差分线路对阻抗匹配的两条线路是相邻的。两条线路之间的间隔使得两线路相互耦合。如果两线路相隔较远，则称作弱耦合（图 3-30）。如果相隔较近，则为强耦合。
子通道：通常在一个链路中需要有多个不同的通道，子通道的主要用途包括控制、状态和辅助数据通道。
系统同步：在2个IC上采用通用时钟来完成数据传输和接收的情况下进行的2个IC间的通信。
TCP：传输控制协议。这是一种TCP/IP网络中的主要协议。IP协议只处理包。TCP协议使两个主机能够建立连接，并交换数据流。该协议保证数据和包传送顺序与它们原来的进入顺序相同。
TDR：时域反射仪。
tf：下降时间的缩写。
令牌环：早期的局域网协议。
tr：上升时间的缩写。
迹线：在印刷电路板表面或内部夹层上用导电材料（譬如铜、银或金）制成的线路或“导线”。这些迹线通常单独称为一条路线。迹线将电信号或其他形式的电流从一点传输到另一点。位

于印刷电路板表面上的迹线都覆盖有非导电涂层（接触点或焊接点除外），以防无意接触其他导电表面。

发送FIFO（先进先出）：在输入数据发送之前，暂时保存数据。

发送线路接口：模拟发送电路，可以支持多种驱动负荷。通常还带有预加重部分。

TTL：晶体管到晶体管逻辑。早期的逻辑系列。

Turbo 乘积码：十分有效的向前纠错法。

twidth：用时间表示的脉冲宽度。

UART：通用异步收发器。这是一种对串行通信进行标准化的芯片。它的功能是将字节转换成电脉冲的标准序列。

UDP：用户数据报协议。一种连接协议，类似于TCP，在IP网络之上运行。

与TCP/IP不一样，UDP/IP提供极少的错误恢复服务，而是提供直接的方式在IP网络上发送和接收数据报。它主要用于在网络上进行报文广播。

UI：时间间隔；等价于一个符号的时间长度，例如：0.2UI = 20%的符号时间。

USB：通用串行总线。一种外设接口标准，通过电缆利用12Mb/s双向串行传输来实现计算机和外设之间的即插即用通信。

Via（过孔）：馈通。印刷电路板上的电镀过孔，用于在电路板上垂直布设迹线，也就是从一层到另一层。

Viterbi：功能强大的向前纠错法。

VME：Versa模块 Eurocard 总线。摩托罗拉、Signetics、Mostek和Thompson CSF开发的32位总线。它广泛用于工业、商业和军事应用，全球VEMbus产品的制造商超过300家。它符合IEEE标准1014-1987。

VME64 标准是扩展版本，提供64位数据传输和寻址。

弱耦合：差分线路对阻抗匹配的两条线路是相邻的。两条线路之间的间隔使得两线路相互耦合。如果两线路相隔较远，则称作弱耦合（图 3-30）。如果相隔较近，则为强耦合。

XAUI：4 通道接口（2.5 Gb/s 有效载荷，3.125 Gb/s 传输线速度），用于 10G 以太网。

轻松实现高速串行 I/O

FPGA应用设计者指南

如果获取 10-Gbps 的 I/O 性能?

高速串行 I/O 可以用于解决系统互联设计挑战。这类 I/O 如果被整合到高度可编程数字环境中（如 FPGA），您就可以创建原来根本不可能实现的高性能设计。本书讨论了高速串行设计的方方面面，并提供实例来说明如何实现设计，包括：

- **I/O 基本原理** - 差分信号、系统同步和源同步设计技巧。
- **不同实现方式的优缺点** - 如何评估成本优势、降低 EMI、最大数据流等。
- **SERDES 设计** - 基础理论、如何实现高效的串行-并行通道、编码机制等。
- **设计考虑事项** - 标准和定制协议、信号完整性、阻抗、屏蔽等。
- **测试** - 解释眼图、减少抖动、协同能力考虑事项、误码测试仪等。

Xcell PUBLICATIONS

Xcell 出版物帮您解决设计挑战，为您带来了最新工具、器件和技术方面的资讯，以及如何最有效地实现设计和下一步该如何实现解决方案方面的知识。请参照我们的书籍、杂志、技术杂志、解决方案指南和宣传册：*www.xilinx.com/cn/xcell* 。

1.0版
2005 年 4 月

PN0402399