摘 要

随着大规模集成电路设计和制造技术的飞速发展，以 Shannon 均匀采样定理为基础的数字信号处理技术得到了飞速发展和广泛应用，但是，在具体的工程实践中， Shannon 采样定理也显现出了它的局限性。本文主要探讨了非均匀采样的基本理论及实现，并研制出一套基于 FPGA 的非均匀采样系统。

数字前端是现在移动终端中最重要的信号处理设备，降低其对功率和频率的依赖具有重要的理论和现实意义。本文通过可编程逻辑器件(FPGA)的特有结构，设计了一种简捷高效的随机时钟产生模块，并利用所产生的随机时钟作为数字前端中模数转换器(ADC)的工作时钟频率，以达到产生不均匀采样 ADC 的目的，从而有效地降低 ADC 或数字前端的功率消耗。通过仿真实验结果表明，本文所设计的 FPGA 模块能够产生随机性很好的时钟信号，而且优于现有的设计方法。同时本文通过实验发现随机性越大的时钟信号能有效降低数字前端的功率消耗。

文章首先是绪论。简单的介绍了采样理论及采样方式；并概述了非均匀采样的理论及其国内外发展现状，然后在第二部分详细介绍了非均匀采样的基础理论。在第三部分用 MATLAB 对均匀采样和非均匀采样进行了仿真和频谱分析，对非均匀采样的可靠性进行了验证。第四部分是本文的重点部分。详细地介绍了整个随机时钟发生模块的硬件设计与实现，软件设计与实现。首先是整个硬件系统的设计框图；然后介绍了 FPGA 与 ADC 接口的硬件设计；最后对 FPGA 进行软件编制及功能的实现，并对 ADC 在采用随机时钟和均匀时钟时的功率消耗进行对比。最后是本文的结束语。总结了本课题的相关内容，以及今后的研究方向。

关键字：随机时钟；可编程逻辑器件(FPGA)；数字前端；数模转换器(ADC)

Abstract

The theory of digital signal processing, based on Shannon uniform sampling theorem, is perfect and has been applied widely in various fields. But, in the practical engineering, the disadvantages of Shannon sampling theorem result in some problems. This paper illuminates the theory and application of non-uniform sampling, and develops a hardware system to implement non-uniform sampling based on FPGA.

Recently, digital front-end is one of the most important parts in mobile terminal devices. Decreasing power consumption and sampling frequency in Digital front-end have considerably theoretical and practical significances. In this article, we propose to using FPGA to design a circuit which can generate the random clock simply and efficiently. Then we use this random clock as the working clock of ADC, and the random clock can sample the signal randomly and decrease the power consumption of the ADC or Digital front-end efficiently. By the simulation and experiment results, we validate that the design of circuit in this article can generate highly randomly clock signal, and we also proved that greater randomness of clock is, the lower power consumption of ADC has.

The article is an introduction first.Simple theory and sample method for introducing a sample;Combined to outline the theory of nonuniform sample and it deveolps present condition at home and abroad, then introduces the foundation theory of not- even sample in detail in the second fraction.Use in the third fraction MATLAB carried on to imitate vs even sample and non-blance sample really with frequency chart analytical, carried on a verification to the reliability of not- even sample.Number the four-part deci is a textual point fraction.In detail introduced whole immediately the hardware design and realize of the clock occurrence mold mass, software design and realize.Is the design frame of the whole hardware system first chart;Then introduced the

hardware design of FPGA and ADC connecting orifice;Finally carry on software compilation and the realize of function to FPGA, and vs ADC the power dissipation progress while adopting random clock and even clock contrast.The end is a textual end language.Tallied up the related contents of this topic, and aftertime's search direction.

Key words: random clock; FPGA; Digital front-end; ADC

Contents

第一章 绪论

1.1 研究的背景和意义

随着计算机技术的发展，实际应用场合对信息处理的要求越来越高，使得数字信号处理理论逐步成熟，并形成了具有强大生命力的学科。利用计算机来处理连续时间信号，首要问题就是对连续信号进行采样，将连续信号转换成离散信号，得到数字信号。所谓采样，就是按照一定的时间间隔 Δt 获取连续时间信号 $f(t)$ 的一系列采样值 $f(n*\Delta t)$(n=1,2,3,…,∞)。采样技术是由一个采样保持电路和一个 A/D(模拟信号/数字信号)变换器来实现的。A/D 输出的数字信号提供给数字信号处理单元进行统计、分析、处理以及显示，得到各种结果。一个典型的信号处理过程如下图 1-1 所示。

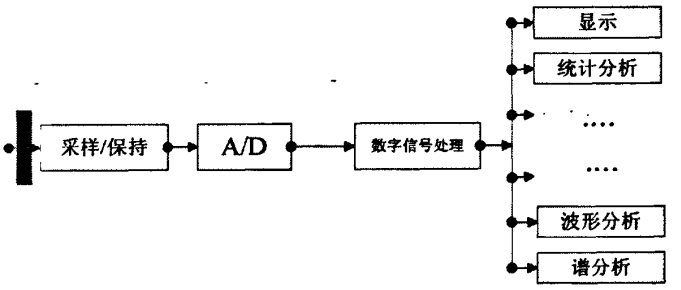


图1-1 典型信号处理过程

Fig.1-1 Traditional signal processing

实际上，一个完整的 A/D 转换器包含以下三个部分：

(1) 采样，即时间量化，将连续时间信号转变成采样信号；

(2) 幅值量化，将离散时间信号的幅值分成若干等级；

(3) 编码，即数字量化，给每个幅值等级分配一个代码。

其中，时间量化决定着 A/D 的采样速率；幅值量化决定着 A/D 的数据位数；数字量化决定着 A/D 的编码方式。采样是其中关键的环节，在数字信号处理学科中，采样理论和技术是信号处理理论的基础。

从采样时间间隔角度上可以将采样分成均匀采样和非均匀采样两种。均匀采样的采样时间间隔是完全相等，不过是一种理想的采样方法，实际

中由于采样设备和被采样信号的限制，完全均匀采样是无法实现的。但随着电子技术的发展，采样设备可以尽量做到近似完全均匀采样，虽然仍然存在采样时间间隔不等的问题，但这些差别已经很小，这些微小的时间误差在一般的工业应用中将不再影响信号处理结果。本文是研究非均匀采样，所以对均匀采样在此处就不再赘述。

采样方式可分为两大类：均匀采样和非均匀采样。从广义上说，均匀采样和非均匀采样的主要区别在于采样间隔是否相等，均匀采样的采样间隔是等间隔，非均匀采样的采样间隔是变化的、非恒定的。

由于 Shannon 采样定理的建立[1]，以均匀采样为基础的数字信号处理方法得到了飞速的发展和广泛的应用。均匀采样的优点非常明显：(1)均匀采样是最简单的采样方式，并且非常直观，易于实现；(2)均匀采样得到的离散序列非常适合数字化处理，易于实现快速算法。但是，其存在明显的缺点，根据 Shannon 采样定理，均匀采样时的采样频率必须大于信号带宽的 2 倍，于是在信号频率很高时，采样频率会高的在工程实践中无法实现或者实现成本很高，例如，目前软件无线电主要在中频实现，主要原因就是因为不易制造出高速 ADC 器件。

非均匀采样有很多种，一般来说只要采样间隔不是恒定的，就可以认为是非均匀采样，但是对于大多数非均匀采样其并不具有特别的性能。这里，我们所研究的非均匀采样特指两种情况：随机采样和伪随机采样。随机采样中每个采样点的选择是完全随机的，是理想化的非均匀采样；伪随机采样中每个采样点的选择是经过挑选的伪随机数。本课题就是基于伪随机采样的时钟信号来驱动数字前端进行采样，通过不均匀采样理论中对于采样频率的灵活要求，降低数字前端的采样频率，从而达到降低数字前端的功率消耗的目的。

1.2 非均匀采样的理论及其国内外发展现状

早期由 BLACK 首先提出了非均匀采样理论的最初形式,它提出了非均匀采样时信号重建的条件和可能性；1956 年 Yen 提出了更加详尽的非均匀采样理论，即：如果信号是一个随时间变化的幅值函数，信号中的最高频

率分量的频率为 W，如果时间可分为以 T 秒为宽度的若干相等区域，其中 T=NW/2 且在每个区域中采样点以任意方式排列情况下：

(1) 当每个区域的采样点数为 N 时，通过采样时间和采样幅值，原信号可以被唯一确定；

(2) 当采样点小于 N 时，则称为欠确定情况，此时只有在附加条件的情况下，信号才能被唯一确定；

(3) 反之，当采样点超过 N 时，则称为过确定情况，信号不能被任意赋值，还需要满足一定的严格条件。

随着采样理论不断发展，Sankur 和 Gerhardt 从指导非均匀采样信号重建的实际应用出发，对非均匀采样信号重建的几种常用技术进行了系统的分析，这些技术包括：低通滤波器，Karhunen Loeve 内插，样条函数，多项式内插，Yen 内插等[2]。随后美国科学家 Higgins 用抽象数学研究了非均匀采样序列集合的结构，提出了一条基本性质，即：在非均匀采样情况下，带限信号的采样序列可分解为两个集合，一个是单位脉冲($\sin \pi t/\pi t$)的变换集，另一个是拉格朗日内插函数集。接着以后几年美国科学家 Papoulis 用多维线性系统理论讨论了具有一般性的采样问题。显然从理论上说，一般性采样问题的理论也应该适用于非均匀采样问题，但文中并没有给出如何应用的说明。接着 Edwin 采用柯西残差理论推导出一种可用于有限点的非均匀采样信号重建公式。近些年来，由于快速采样系统中出现了输入多路并联，输出多路复用技术，国际国内的科技工作者开始从工程技术的角度研究非均匀采样问题。Jenq 首先提出了分析方法，其特点是，将一个非均匀序列分解为 M 个均匀序列，这样一来，非均匀采样序列就可用 M 个均匀序列的组合来表示，从而求出了被采样信号的模拟频谱与该信号经非均匀采样后，用 DFT 所得的数字频谱之间的普遍关系，目前这一理论仍处于发展之中。

近几年随着数字通信系统和无线数字通信的不断发展，原本是两个不同领域开始出现了交集。不均匀采样在硬件上的实现，给无线通信系统对于数字信号处理上小体积、低功耗、低成本的要求有了极大的改进，并且不均匀采样对于采样频率限制低，对多频带信号处理灵活，因而成为现在数字信号处理在硬件上的实现的研究热点。

1.3 可编程逻辑器件(FPGA)的发展

FPGA 是英文 Field Programmable Gate Array 的缩写，即现场可编程门阵列，它是在可编程阵列逻辑 PAL(Programmable Array Logic)、门阵列逻辑 GAL(Gate Array Logic)、可编程逻辑器件 PLD(Programmable Logic Device)等可编程器件的基础上进一步发展的产物。它是作为专用集成电路 ASIC(Application Specific Integrated Circuit)领域中的一种半定制电路而出现的，既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点。FPGA 能完成任何数字器件的功 能，上至高性能 CPU,下至简单的 74 系列电路，都可以用 FPGA 来实现[3]。

FPGA 具有体系结构和逻辑单元灵活、集成度高以及适用范围宽等特点。 兼容了 PLD 和通用门阵列的优点，可实现较大规模的电路，编程也很灵活。与门阵列等其它 ASIC 相比，它又具有设计开发周期短、设计制造成本低、开发工具 先进、标准产品无需测试、质量稳定以及可实时在线检验等优点，因此被广泛应用于产品的原型设计和产品生产(一般在 10,000 件以下)之中。几乎所有应用 门阵列、PLD 和中小规模通用数字集成电路的场合均可应用 FPGA。

FPGA 采用了逻辑单元阵列 LCA(Logic Cell Array)这样一个新概念，内部包括可配置逻辑模块 CLB(Configurable Logic Block)、输出输入模块 IOB(Input Output Block)和内部连线(Interconnect)三个部分[4]。FPGA 的基本特点主要有：一是采用 FPGA 设计 ASIC 电路，用户不需要投片生产，就能得到合用的芯片。二是 FPGA 可做其它全定制或半定制 ASIC 电路的中试样片。三是 FPGA 内部有丰富的触发器和 I / O 引脚。四是 FPGA 是 ASIC 电路中设计周期最短、开发费用最低、风险最小的器件之一。五是 FPGA 采用高速 CHMOS 工艺，功耗低，可以与 CMOS、TTL 电平兼容。可以说，FPGA 芯片是小批量系统提高系统集成度、可兼容性的最佳选择之一。目前 FPGA 的品种很多，有 XILINX 的 XC 系列、TI 公司的 TPC 系列、ALTERA 公司的 FIEX 系列等。FPGA 是由存放在片内 RAM 中的程序来设置其工作状态的，因此，工作时需要对片内的 RAM 进行编程。用 户可以根据不同的配置模式，采用不同的编程方式。加电时，FPGA 芯片将

EPROM 中数据读入片内编程 RAM 中，配置完成后，FPGA 进入工作状态。掉电 后，FPGA 恢复成白片，内部逻辑关系消失，因此，FPGA 能够反复使用。FPGA 的编程无须专用的 FPGA 编程器，只须用通用的 EPROM、PROM 编 程器即可。当需要修改 FPGA 功能时，只需换一片 EPROM 即可。这样，同一片 FPGA，不同的编程数据，可以产生不同的电路功能。因此，FPGA 的使用 非常灵活。FPGA 有多种配置模式：并行主模式为一片 FPGA 加一片 EPROM 的方式；主从模式可以支持一片 PROM 编程多片 FPGA；串行模式可以采用 串行 PROM 编程 FPGA；外设模式可以将 FPGA 作为微处理器的外设，由微处理器对其编程。FPGA 具有掩膜可编程门阵列的通用结构，它由逻辑功能块排成阵列组成，并由可编程的互连资源连接这些逻辑功能块来实现不同的设计[5]。

FPGA 一般由三种可编程电路和一个用于存放编程数据的静态存储器 SRAM 组成。这三种可编程电路是：可编程逻辑块(Configurable Logic Block，CLB)、输入/输出模块(I/O Block，IOB)和互连资源(Interconnect Resource，IR)。FPGA 的基本结构如图 1.19 所示，可编程逻辑块(CLB)是实现逻辑功能的基本单元，它们通常规则地排列成一个阵列，散布 于整个芯片；可编程输入/输出模块(IOB)主要完成芯片上的逻辑与外部封装脚的接口，它通常排列在芯片的四周；可编程互连资源(IR)包括各种长度的连线线段和一些可编程连接开关，它们将各个 CLB 之间或 CLB、IOB 之间以及 IOB 之间连接起来，构成特定功能的电路[6]。

FPGA 的功能由逻辑结构的配置数据决定。工作时，这些配置数据存放在片内的 SRAM 或熔丝图上。基于 SRAM 的 FPGA 器件，在工作前需要从芯 片外部加载配置数据，配置数据可以存储在片外的 EPROM 或其他存储体上。用户可以控制加载过程，在现场修改器件的逻辑功能，即所谓的现场编程。FPGA 由 6 部分组成，分别为可编程输入/输出单元、基本可编程逻辑单元、嵌入式块 RAM、丰富的布线资源、底层嵌入功能单元和内嵌专用硬核等[7][8]。

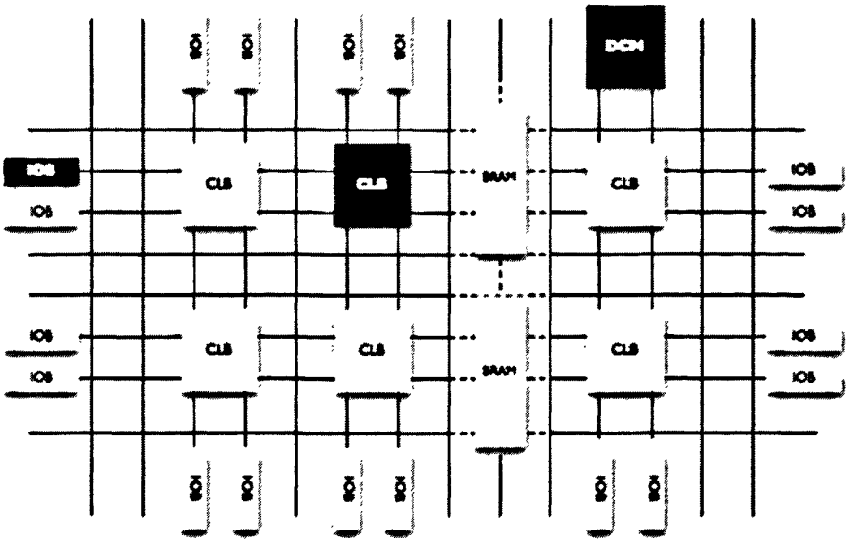


图1-1 FPGA基本结构

Fig.1-1 Artrutecture

FPGA 技术正处于高速发展时期，新型芯片的规模越来越大，成本也越来越低，低端的 FPGA 已逐步取代了传统的数字元件，高端的 FPGA 不断在争夺 ASIC 的市场份额。先进的 ASIC 生产工艺已经被用于 FPGA 的生产，越来越丰富的处理器内核被嵌入到高端的 FPGA 芯片中，基于 FPGA 的开发成为一项系统级设计工程。随着半导体制造工艺的不同提高，FPGA 的集成度将不断提高，制造成本将不断降低，其作为替代 ASIC 来实现电子系统的前景将日趋光明。

(1) 大容量、低电压、低功耗 FPGA

大容量 FPGA 是市场发展的焦点。FPGA 产业中的两大霸主：Altera 和 Xilinx 在超大容量 FPGA 上展开了激烈的竞争。2007 年 Altera 推出了 65nm 工艺的 StratixIII 系列芯片[9]，其容量为 67200 个 L E (Logic Element，逻辑单元)，Xilinx 推出的 65nm 工艺的 VitexVI 系列芯片，其容量为 33792 个 Slices (一个 Slices 约等于 2 个 L E)。采用深亚微米(DSM)的半导体工艺后,器件在性能提高的同时，价格也在逐步降低。由于便携式应用产品的发展，对 FPGA 的低电压、低功耗的要日益迫切。

(2) 系统级高密度 FPGA

随着生产规模的提高,产品应用成本的下降，FPGA 的应用已经不是过

去的仅仅适用于系统接口部件的现场集成，而是将它灵活地应用于系统级(包括其核心功能芯片)设计之中。在这样的背景下，国际主要 FPGA 厂家在系统级高密度 FPGA 的技术发展上，主要强调了两个方面：FPGA 的 IP(Intellec2tual Property ，知识产权)硬核和 IP 软核[10]。当前具有 IP 内核的系统级 FPGA 的开发主要体现在两个方面：一方面是 FPGA 厂商将 IP 硬核(指完成版图设计的功能单元模块)嵌入到 FPGA 器件中，另一方面是大力扩充优化的 IP 软核(指利用 HDL 语言设计并经过综合验证的功能单元模块)，用户可以直接利用这些预定义的、经过测试和验证的 IP 核资源，有效地完成复杂的片上系统设计。

(3) FPGA 和 ASIC 出现相互融合

虽然标准逻辑 ASIC 芯片尺寸小、功能强、功耗低，但其设计复杂，并且有批量要求。FPGA 价格较低廉，能在现场进行编程，但它们体积大、能力有限，而且功耗比 ASIC 大。正因如此，FPGA 和 ASIC 正在互相融合，取长补短。随着一些 ASIC 制造商提供具有可编程逻辑的标准单元，FPGA 制造商重新对标准逻辑单元发生兴趣。

(4) 动态可重构 FPGA

动态可重构 FPGA 是指在一定条件下芯片不仅具有在系统重新配置电路功能的特性，而且还具有在系统动态重构电路逻辑的能力[11]。对于数字时序逻辑系 统，动态可重构 FPGA 的意义在于其时序逻辑的发生不是通过调用芯片内不同区域、不同逻辑资源来组合而成，而是通过对 FPGA 进行局部的或全局的芯片逻辑的动态重构而实现的。动态可重构 FPGA 在器件编程结构上具有专门的特征，其内部逻辑块和内部连线的改变，可以通过读取不同的 SRAM 中的数据来直接实现这样的逻辑重构，时间往往在纳秒级，有助于实现 FPGA 系统逻辑功能的动态重构。

基于以上优点，在数字通信系统和无线通信系统中，FPGA 有着越来越广泛的应用。

1.4 本文的研究的内容及章节安排

第一章是绪论。简单的介绍了采样理论及采样方式；并概述了非均匀

采样的理论及其国内外发展现状。

第二章详细介绍了非均匀采样的基础理论。

第三章介绍了采用了 MATLAB 对均匀采样和非均匀采样的频谱分析的过程。

第四章是本文的重点部分。详细地介绍了整个随即时钟发生模块的硬件设计与实现，软件设计与实现。首先是整个硬件系统的设计框图；然后介绍了 FPGA 与 ADC 接口的硬件设计；最后对 FPGA 进行软件编制及功能的实现，并对 ADC 在采用随机时钟和均匀时钟时的功率消耗进行对比。

第五章是本文的结束语。总结了本课题的相关内容，以及今后的研究方向。

第二章 非均匀采样的分析

实际工程应用中，为了采用数字化的处理手段，必须将信号进行采样、量化，转换为适当的数字信号，然后借助于数字化处理方法进行处理。采样显然是至关重要的，因为模拟信号只要经采样、量化后，其一切特性就再也无法改变。前面已经对采样理论进行了介绍，在此就直接开始对非均匀采样的理论进行分析研究。

2.1 非均匀采样的可靠性分析

下面给出了是非均匀傅里叶变换的表达公式、推导及计算精度分析[12]

2.1.1 非均匀采样的频谱分析

对于信号$f(t)$满足下列条件：(1)绝对可积，即$\int_{-\infty}^{+\infty}|f(t)|dt<\infty$；(2)在任何有限区间内，$f(t)$只存在有限个数目的最大值和最小值；(3)在任何有限区间内，$f(t)$有有限个数目的不连续点，并且在每个不连续点都必须是有限值。则$f(t)$的傅里叶变换存在，即存在：

$$f(t)=\frac{1}{2\pi}\int_{-\infty}^{+\infty}F(j\omega)e^{j\omega t}d\omega \tag{2-1}$$

和

$$F(j\omega)=\int_{-\infty}^{+\infty}f(t)e^{-j\omega t}dt \tag{2-2}$$

当$f(t)$经过均匀采样后，得到离散序列$f(nT)$，其中 T 为采样周期。用$f(n)$代表$f(nT)$，则$f(n)$序列的离散时间傅里叶变换表示如下：

$$f(n)=\frac{1}{2\pi}\int_{2\pi}F(e^{j\omega})e^{j\omega t}d\omega \tag{2-3}$$

$$F(e^{j\omega})=\sum_{n=-\infty}^{+\infty}f[n]e^{-j\omega v} \tag{2-4}$$

根据 Shannon 采样定理，时域上的采样，将使信号频谱在频域上发生

搬移，若采样频率大于奈奎斯特频率，则不会发生频谱重叠。从而，

$$F_{\rho}\left(e^{j\omega}\right)=\frac{1}{T}\sum_{k=-\infty}^{+\infty}F\left(j\left(\omega-k\omega_{s}\right)\right) \tag{2-5}$$

其中，$F_{\rho}\left(e^{j\omega}\right)$为采样后得到的离散序列的频谱，T 为采样周期，$S(\omega)$为采样频率(角频率)。当采用非均匀采样时，得到的离散序列为 $f(tk)$，其中 tk 表示采样时刻。直接套用均匀采样的离散时间傅里叶变换，可以得到以下公式：

$$F\left(e^{j\omega}\right)=\sum_{k=-\infty}^{+\infty}f\left(t_{k}\right)e^{-j\omega t_{k}} \tag{2-6}$$

下面给出简洁的证明[13]

假设非均匀采样的各个采样点是随机的，且相互独立，其概率密度分布函数为 $p(t)$，采样点数为 N，则：

$$\begin{aligned}E\left\{\sum_{k=1}^{N}\int_{-\infty}^{+\infty}f\left(t_{k}\right)e^{-j\omega t_{k}}\right\}&=\sum_{k=1}^{N}E\left\{f\left(t_{k}\right)e^{-j\omega t_{k}}\right\}\\&=\sum_{k=1}^{N}\int_{-\infty}^{+\infty}f\left(t_{k}\right)e^{-j\omega t_{k}}p\left(t_{k}\right)dt_{k}\end{aligned} \tag{2-7}$$

如果 $p(t)$在信号持续时间[0,T_d]上服从均匀分布，即

$$p(t)=\begin{cases}1/T_{d}, t\in[0,T_{d}]\\0, other\end{cases}$$

则：

$$\begin{aligned}\sum_{k=1}^{N}\int_{-\infty}^{+\infty}f\left(t_{k}\right)e^{-j\omega t_{k}}p\left(t_{k}\right)dt_{k}&=\frac{1}{T_{d}}\sum_{k=1}^{N}\int_{0}^{T_{d}}f\left(t_{k}\right)e^{-j\omega t_{k}}dt_{k}\\&=\frac{N}{T_{d}}\int_{0}^{T_{d}}f\left(t\right)e^{-j\omega t_{k}}dt\\&=\frac{N}{T_{d}}F\left(j\omega\right)\\&=\frac{1}{\mu}F\left(j\omega\right)\end{aligned} \tag{2-8}$$

其中 $\mu=\frac{T_{d}}{N}$为平均采样时间。将式(2-8)代入(2-7)，并结合式(2-6)，可知

$$E\left\{F\left(e^{j\omega}\right)\right\}=\frac{1}{\mu}F\left(j\omega\right) \tag{2-9}$$

即非均匀离散傅里叶变换公式计算结果的期望是原始信号频谱。

2.1.2 不均匀采样精度分析

由于采样时刻的随机性，计算得到的信号频谱的期望是信号的真实频谱，因此必须考察频谱计算的精度[14]。非均匀离散傅里叶变换的方差推导如下：

令 $X_D(\omega)$ 代表由非均匀离散傅里叶变换计算得到的频谱，$X_c(\omega)$ 代表信号的实频谱，则

$$\begin{aligned}\mathrm{var}\left\{X_D(\omega)\right\}&=E\left\{\left[X_D(\omega)-E\left\{X_D(\omega)\right\}\right]^2\right\}\\&=E\left\{\left[X_D(\omega)\right]^2\right\}-\left[E\left\{X_D(\omega)\right\}\right]^2\end{aligned} \tag{2-10}$$

由于 $\{t_k\}$ 为相互独立、同分布的随机变量，概率密度函数为 $p(t)$，故

$$\begin{aligned}E\left\{\left[X_D(\omega)\right]^2\right\}&=E\left\{\left[\sum_{k=-\infty}^{+\infty}f(t_k)e^{-j\omega t_k}\right]^2\right\}\\&=E\left\{\left[\sum_{k=1}^{+\infty}f(t_k)e^{-j\omega t_k}\right]^2\right\}\\&=E\left\{\sum_{k=1}^{N}\sum_{l=1}^{N}f(t_k)e^{-j\omega t_k}.f(t_l)e^{-j\omega t_l}\right\}\\&=\sum_{k=1}^{N}\sum_{l=1}^{N}f(t_k)e^{-j\omega t_k}.f(t_l)e^{-j\omega t_l}\\&=\sum_{k=1}^{N}E\left\{f(t_k)e^{-j\omega t_k}.f(t_l)e^{-j\omega t_l}\right\}\\&+\sum_{k=1}^{N}\sum_{l=k}^{N}E\left\{f(t_k)e^{-j\omega t_k}.f(t_l)e^{-j\omega t_l}\right\}\end{aligned} \tag{2-11}$$

又因为

$$\begin{aligned}\sum_{k=1}^{N}E\left\{f(t_k)e^{-j\omega t_k}.f(t_l)e^{-j\omega t_l}\right\}&=\sum^{N}\int_0^T\left[f(t_k)e^{-j\omega t_k}\right]^2.p(t_k)dt_k\\&=N\int_0^T\left[f(t)e^{-j\omega t}\right]^2p(t)dt\end{aligned} \tag{2-12}$$

和

$$\begin{aligned}\sum_{k=1}^{N}\sum_{l\neq k}^{N}E\{f(t_k)e^{-j\omega t_k}\cdot f(t_l)e^{-j\omega t_l}\}&=N(N-1)E\{f(t_k)e^{-j\omega t_k}\}\cdot E\{f(t_l)e^{-j\omega t_l}\}\\&=N(N-1)\int_0^T f(t_k)e^{-j\omega t_k}p(t_k)dt_k\cdot\int_0^T f(t_l)e^{-j\omega t_l}p(t_l)dt_l\\&=N(N-1)\int_0^T f(t)e^{-j\omega t}p(t)dt\cdot\int_0^T f(t)e^{-j\omega t}p(t)dt\\&=N(N-1)X_c(\omega)\cdot X_c(\omega)\end{aligned}\tag{2-13}$$

将式(2-11)、(2-12)、(2-13)代入式(2-10)，可得：

$$\begin{aligned}\mathrm{var}\{X_D(\omega)\}&=E\{[X_D(\omega)]^2\}-[E\{X_D(\omega)\}]^2\\&=N\int_0^T[f(t)e^{-j\omega t}]^2p(t)dt+\frac{N(N-1)}{T^2}X_c(\omega)X_c(\omega)\\&-\frac{N^2}{T^2}X_c(\omega)X_c(\omega)\\&=N\int_0^T[f(t)e^{-j\omega t}]p(t)dt-\frac{N}{T^2}X_c(\omega)X_c(\omega)\end{aligned}$$

整理后，可得

$$\frac{T^2}{N^2}\mathrm{var}\{X_c(\omega)\}=\frac{T\int_0^T[f(t)e^{-j\omega t}]^2dt-X_c(\omega)X_c(\omega)}{N}$$

即

$$\mathrm{var}\{\mu X_c(\omega)\}=\frac{T\int_0^T[f(t)e^{-j\omega t}]^2dt-X_c(\omega)X_c(\omega)}{N}\tag{2-14}$$

根据此式可知对频谱计算精度的分析可得计算的信号频谱的期望是信号的真实频谱。

2.1.3 采样时钟抖动对非均匀离散傅里叶变换的影响

在实际采样过程中，采样时钟不可避免地存在一定的抖动。这种时钟抖动会对非均匀离散傅里叶变换的计算结果产生一定的影响，因此，必须对其进行理论上的推导[15]。设t_1，t_2，t_3，……t_N为一非均匀采样序列的采样时刻，τ_1，τ_2，τ_3…. τ_N代表相应的各个采样时刻的抖动，则此时的非均匀离散傅里叶变换如下式：

$$X_D(\omega)=\sum_{k=1}^{N}x(t_k)e^{-j\omega(t_k+\tau_k)}\tag{2-15}$$

如果假设τ_1，τ_2，τ_3.... τ_N满足以下两个条件：

a) 服从同分布 g(τ)，且相互独立；

b) 与t_1，t_2，t_3，.....t_N独立则采样时钟抖动对非均匀离散傅里叶变换的影响，可由下面两个数学表达式表示：$E\{X_D(\omega)\}=\frac{1}{\mu}X_c(\omega)\cdot G(\omega)$

$$\mathrm{var}\{\mu X_D(\omega)\}=\frac{TE_x-X_c(\omega)X_c(\omega)}{N}+\frac{TE_x}{N}\left[\frac{1-|G(\omega)|^2}{|G(\omega)|^2}\right] \tag{2-16}$$

其中，$E_x=\int_0^T\left[f(t)e^{-j\omega t}\right]^2dt$，$G(\omega)$是$g(\tau)$的傅里叶变换。

证明过程与理想非均匀离散傅里叶变换类似，这里只给出采样时钟抖动时的非均匀傅里叶变换的分析，下式就是分析的最终结果公式：

$$\begin{aligned}E\{X_D(\omega)\}&=\sum_{k=1}^{N}E\left\{x(t_k)e^{-j\omega(t_k+\tau_l)}\right\}\\&=\sum_{k=1}^{N}E\left\{x(t_k)e^{-j\omega(t_k+\tau_l)}\right\}E\left\{e^{-j\omega(t_k+\tau_l)}\right\}\\&=\sum_{k=1}^{N}\frac{1}{T}X_c(\omega)\int_{-\infty}^{+\infty}e^{-j\omega\tau_k}g(\tau_k)d\tau_k\\&=\frac{1}{\mu}X_c(\omega)G(\omega)\end{aligned} \tag{2-17}$$

2.2 非均匀采样中采样时刻的选择

采样时刻的选择无疑是非常重要的，它决定了采样后信号的性质。下面对两种非均匀采样时刻的选择方法进行分析[16]。

2.2.1 时钟抖动的均匀采样(JRS)

时钟抖动的均匀采样在工程实践中是普遍存在的，并且是不可避免的，例如 ADC 时钟频率存在一定偏差等[17]。有抖动的均匀采样时刻$\{t_k\}$,其数学表达式为：

$$t_k=kT+\tau_k,T>0$$

其中，T 表示均匀采样的采样周期，$\{\tau_k\}$为服从同分布的一组随机变量，

其均值是 0。设$\{\tau_k\}$的概率密度函数为$p(\tau_k)$则采样时刻t_k的概率密度函数为$p(t-(t_k-t_n))$因此，时钟抖动的均匀采样的采样点的分布如图 2-1 所示。

图2-1有抖动的均匀采样点概率分布

Fig.2-1 Jitter random sampling probability distribution

时钟抖动的均匀采样明显存在很大的缺点。如果在区间[$kT-0.5T$, $kT+0.5T$]上不是均匀分布，则显然，在 kT 点附近采样点数很多，其它地方采样点很少。如果τ_k在区间[$kT-0.5T$, $kT+0.5T$]上满足均匀分布，则会发生某些相邻采样点间距很小的情况。对第一种情况，它和均匀采样区别很小，无法利用非均匀采样的优点；对第二种情况，在实际实现中会非常困难，以致于无法实现，因为采样间距过小对 ADC 的要求很高。显然，这两种情况都不是我们所希望的。

2.2.2 加性非均匀采样(ARS)

在加性非均匀采样中，当前采样时刻是根据前一个采样时刻来选择的，其数学表达式为：$t_{k+1}=t_k+\tau_k$其中，τ_k为服从同分布的一组随机变量，其值恒为正[18]。

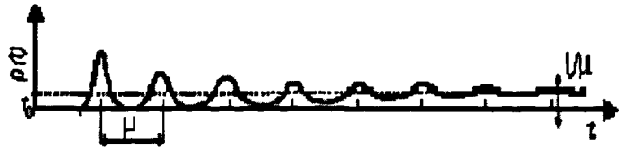


图2-2加性非均匀采样点的概率分布

Fig.2-2 Additive random sampling probability distribution

设τ_k的概率密度函数为$p_\tau(\tau_k)$其均值为 μ，由于$t_k=t_0+\tau_1+\tau_2+....+\tau_k$故$p_k(t)=p_{k-1}(t)*p_\tau(t)$根据中心极限定理，对于一组相互独立随机变量，当随机变量的个数大到一定程度的时候，它们的和服从正态分布，因此当 $k\to\infty$时，$p_k(t)$将趋向于正态分布。当 t 增加时，加性非均匀采样点的概率分

布 $p(t)=\sum_{k=1}^{\infty}p_k(t)$将趋向于平坦，其数值大小为 1/μ,如图 2-2 所示。

2.3 非均匀采样的抗频率混叠

由于采样时刻的分布不同于均匀采样，非均匀采样具有一个非常重要的特点就是可以消除频率混叠现象,我们用下面的例子进行形象化的阐述：假设给出一组采样数据，它代表了一个正弦信号(加粗的黑色)的均匀采样值，如图 2-3 所示[19]。

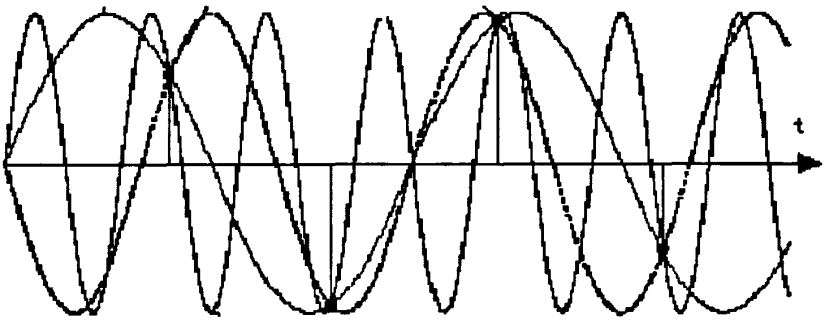


图2-3 混叠的产生

Fig.2-3 aliasing generation

观察图 2-3，就会清楚发现其他的频率的正弦信号和原始信号同一个采样点处的采样值相等(曲线交点处)。因此，如果要用这组采样值进行重建原始信号，显然得到的信号不是唯一的。也就是说，用小于奈奎斯特频率的采样频率进行采样，得到的采样值是无法恢复出原始信号，这与 Shannon 采样定理是相一致的。这种现象反映到频域上就是频率混叠。频率混叠现象就会引起信号的不确定。在没有其它先验知识的情况下，如何消除频率混叠现象是信号处理理论的一个重要研究课题。均匀采样理论中，在进行信号采样前，信号先通过一个低通滤波器以便把信号的频谱限制在一个特定的范围内，然后用高于信号最高频率两倍的采样频率进行采样，从而消除了频率混叠。虽然这种解决混叠问题的方法能够满足要求，但是这种方法滤除调了信号组成成分中超过某一频率的频率成分，很容易造成

失真，同时由于采样频率要高于信号最高频率的两倍，极大限制了数字信号处理理论使用的范围。如果能突破这个限制，将为数字信号处理理论开辟更为广泛的应用领域。所以摆在我们面前的一个问题就是在较低采样频率的情况下，消除频率混叠是否可能？非均匀采样给除了肯定的回答。为了更加直观地说明非均匀采样如何具有消除混叠的性能，我们先观察图2-4。

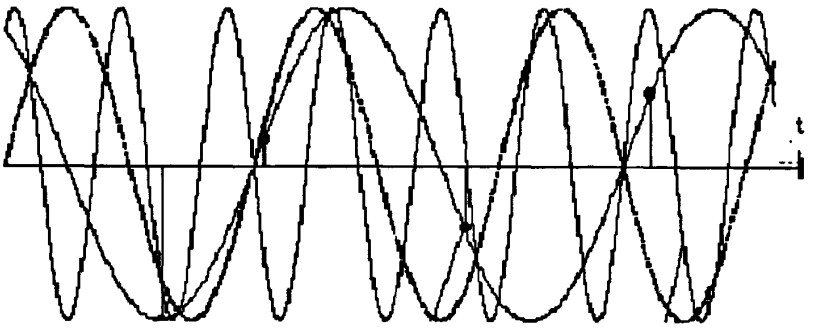


图2-4 消除混叠

Fig.2-4 avoid aliasing

图2-4中对原始的低频正弦信号进行了重新采样，采样点的个数保持不变，所不同的地方是采样点的间隔不再是相等的了。很容易从图中看出，由于采样点不再是均匀的，只有原始的低频正弦波可以通过采样点可以被拟合出来，从而也就消除了频率混叠。

2.4 本章小结

从以上理论分析，我们可以得出非均匀采样能有效对信号进行采样，并且能消除频率混叠的干扰，更重要的一点，不均匀采样降低了对采样频率的限制，对多频带信号处理更灵活。我们通过不均采样得到的离散信号样本，是否能进行信号重建恢复原信号，下一章节，我们将在MATLAB上进行仿真论证。

第三章 均匀采样和非均匀采样的频谱分析

前一章在理论上论证了不均匀采样的可行性，本章将针对 ARS、JRS 和均匀采样三种不同的采样时间序列，在多频带信号上，进行不均匀采样；同时使用 SVD 分解法重建原信号，并且对使用三种不同采样方式重建的信号跟原信号进行对比，对平均信噪比(SNR)和误差率进行对比分析。

3.1 基于 SVD 算法的多频带信号重建

假设多频带信号 x=x(t) 的频带宽度为 B，用 ARS 方式采样，T 为采样时间段 $t_n - t_1$，重建信号 $x'(t)$ 为：

$$x'(t) = \sum c_k \times e^{2j\pi f_k t} \tag{3-1}$$

其中(f_k)属于频带 B 中的任一频率并且对应相应的采样间隔，c_k 由最小均方误差来确定，

$$E_{is}^{\ 2} = |AC - x|^2 \ , \left(A_{ik} = e^{2j\pi f_k t_i}\right) \tag{3-2}$$

最优解由公式(3-3)得到：

$$A^H \times A \times C = A^H \times X_S \tag{3-3}$$

其中 A^H 为 A 的 hermit 矩阵[20]。

经过矩阵运算，我们能得到：$C = (A^H \times A)^{-1} \times A^H \times X_S$ (3-4)

$(A^H \times A)$ 是奇异矩阵，所以我们必须找到一个合适的方法来解出 C.

SVD (singular value decomposition) 是一种奇异矩阵求解的常用方法.。SVD 算法将矩阵分解成三个矩阵：

$$A \times S \times V^H \tag{3-5}$$

U 和 V 两个矩阵维度为 N and M，并且有：

$$U \times U^H = U^H \times U = I \ , \ V \times V^H = V^H \times V = I \ \text{(Identity matrix)} \tag{3-6}$$

S 为对角矩阵：S=diag($\sigma_1,\sigma_2 \cdots \sigma_P$)，且 p=min(M，N)，$\sigma_1 \geq \sigma_2 \cdots \geq \sigma_P \geq 0$。

设 $Z = V^H \times C$ ，$Y = S^H \times U^H \times X_S$，公式(3-3) 可以等价于：

$$S^H \times S \times Z = Y \tag{3-7}$$

然后我们可以解得：$Z_k = \frac{Y_k}{S_{kk}^2}$,k∈[1,...p]. (3-8)

最后我们可以得到相关系数 c_k：$C=V\times Z$ (3-9)

3.2 MATLAB 上信号频谱仿真和分析

设定要重建的信号为 5 路 QPSK 调制载波信号，如图 3-1 所示。符率为 R=4(sym/s)，升余弦滤波器滚降系数为 0.5，每路载波带宽为 8Hz[21]。由此得到整个多频带信号总带宽为 38Hz。这个信号功率谱密度如图 3-2 所示，中心频率为 175Hz。

我们采用两种方法得到采样信号，均匀采样和不均匀采样，对比用两种方法得到的采样重建原始信号误差率。均匀采样中我们用 100Hz 作为采样频率，不均匀采样中我们的平均采样频率设定为 100Hz。在 MATLAB 中我们分别用均匀采样，JRS 和 ARS 对 QPSK 信号进行采样，然后通过 SVD 算法对采样信号进行重建，并计算出均方误差。图 3-5,图 3-6,图 3-7 分别为 JRS，ARS，均匀采样三种方法采样后通过 SVD 算法恢复原信号的仿真图。

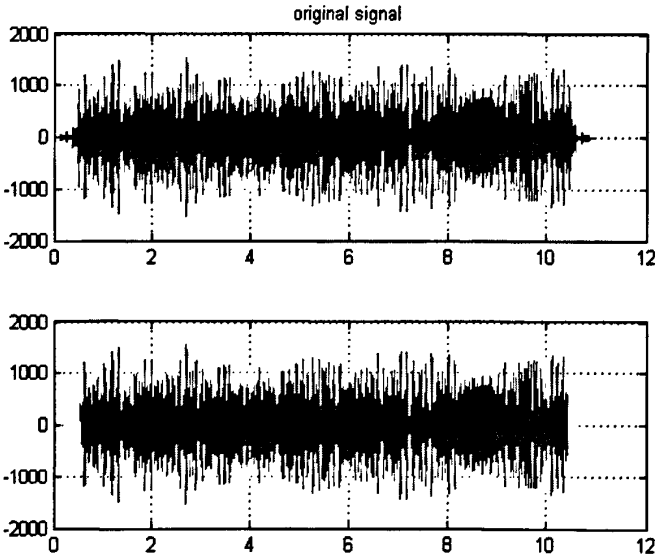


图3-1 QPSK信号频谱图

Fig.3-1 QPSK frequency spectrum

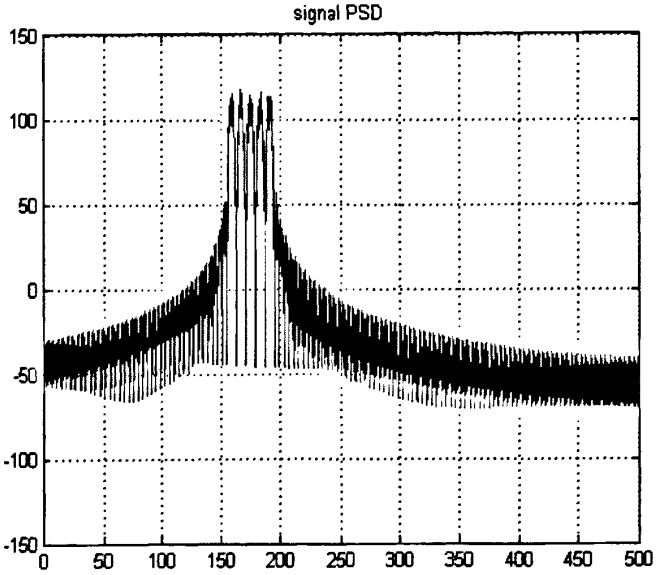


图3-2 QPSK信号功率谱密度

Fig.3-2 QPSK power spectral density

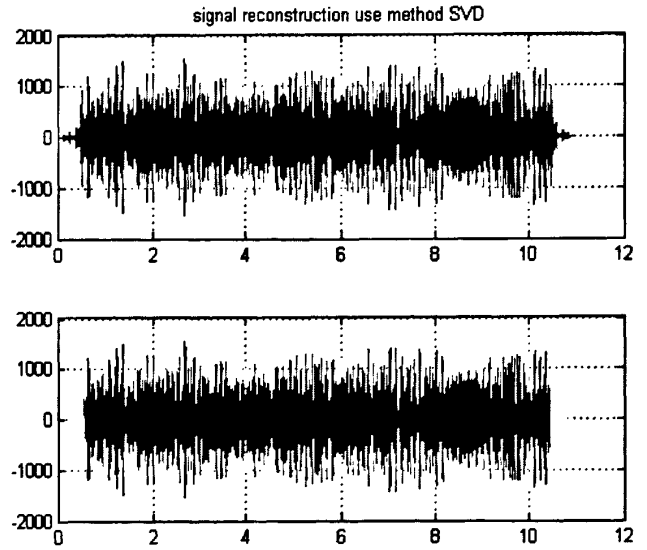


图3-3 JRS采样重建信号频谱图

Fig.3-3 Using JRS reconstruct the signal

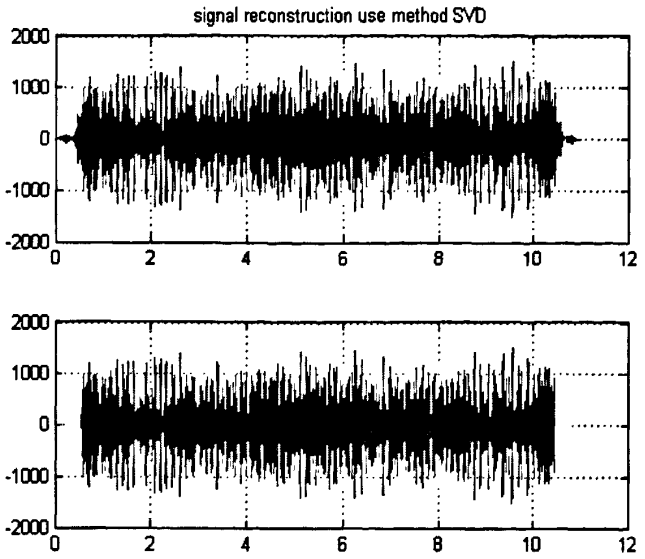


图3-4 ARS采样重建信号频谱

Fig.3-4 Using ARS reconstruct the signal

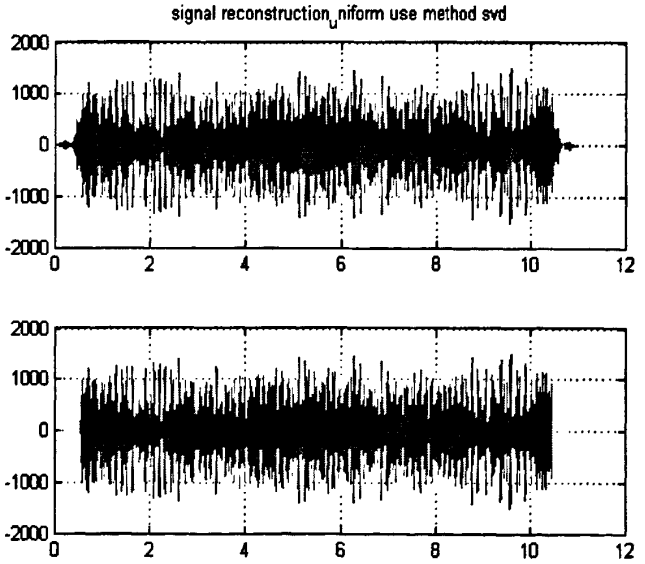


图3-5 均匀采样重建信号频谱

Fig.3-5 Using Uniform sampling reconstruct the signal

3.3 本章小结

在图 3-3，图 3-4，图 3-5 中，上面为原 QPSK 信号，下面为重建信号。通过 MATLAB 计算得到三种方式平均信噪比 SNR 和平均误差值为：

SNR(ARS)= 58.657809(dB), E(ARS)=0.0109.

SNR(JRS)= 59.662221(dB), E(JRS)=0.0076.

SNR(unif)= 67.479956 (dB) , E(unif)=0.0066.

从以上结果我们可以得出结论：非均匀采样与均匀采样的信噪比相差在可接受的范围之内，且通过不均匀采样重建的信号与原信号的误差值非常小，因此不均匀采样可以应用在多频带信号处理上，并且对采样频率的限制要远小于均匀采样。(仿真程序见附录 1)

第四章 数字前端功率消耗对比分析

由于现在传送的信号都具有高频率，宽频带的动态信号，而在以往的模拟前端中，已经无法支持这种高频宽带信号的处理，其工作频率也受到限制。因此数字前端在此背景下应运而生，开始广泛应用与数字通信系统中[22]。如图 4-1 所示，数字前端由 ADC 和数字信号处理 DSP(Digital signal processing)模块组成。DSP 中包括采样频率转换、目标频带选取和数字基带信号处理三个部分。

可以从图4-1中看出，数字前端可以将接受的多频带信号管道化，能够通过信道滤波器筛选出要处理的指定频率之间的信号，对筛选出来的信号通过ADC随机采样进行模数转换，然后进行数字信号处理。这样可以有效减少信号处理元件，加强接收器的适应性。在信号通过ADC时，通过不均匀采样可以有效降低采样频率，在重建信号时，可以选择目标频带的信号通过不均匀采样重建算法进行信号恢复。所以对ADC功率消耗的研究对于提高整个数字通信系统的工作效率，降低整体功率消耗有极为重大的意义。

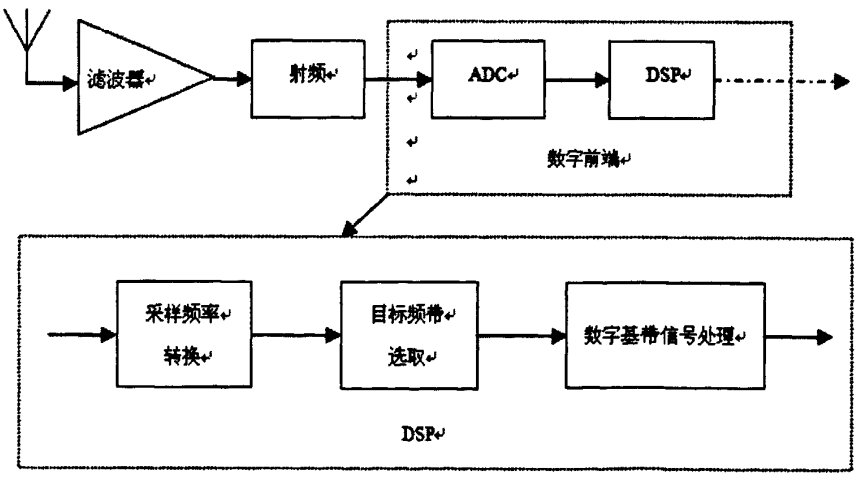


图4-1 数字前端

Fig.4-1 Digital-front-end

4.1 伪随机采样模块设计

在软件中一般都使用大数乘法器的递推公式来产生伪随机数序列[23]，如 C 语言中可以调用 rand( )函数产生随机数。随着 FPGA 在电子设备领域的逐渐发展应用，在很多高速设计和高速测试应用中，希望直接通过 FPGA 来产生随机序列，而实际中产生的不是绝对随机数，而是相对随机数，称为伪随机数，即具有一定周期可预见性的数。传统的大数乘法器不但不能产生高频率的时钟信号，而且占用 FPGA 大量逻辑单元，因此为了产生所需的随机时钟信号，下面将利用 FPGA 内的逻辑资源和线性移位寄存器(LFSR)设计一个简单、高效的伪随机采样时钟信号产生模块。本文所设计的随机时钟的产生如图 4-2 所示，主要包括三个部分：寄存器、线性反馈移位寄存器和多路复用器。

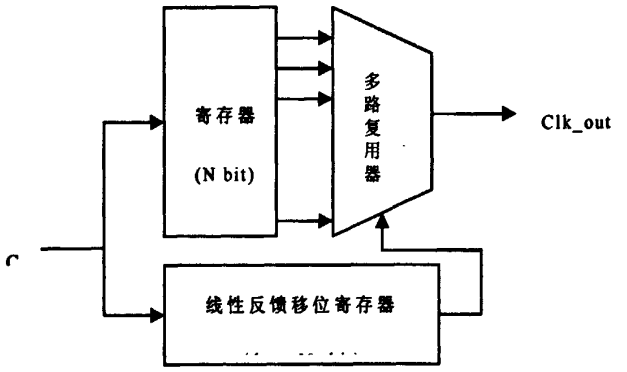


图4-2 伪随机采样电路结构图

Fig.4-2 Architecture of pseudorandom clock generator

本文所设计的随机时钟的产生如图 4-2 所示，主要包括三个部分：

(1) 寄存器：用来存放 1 或者 0。

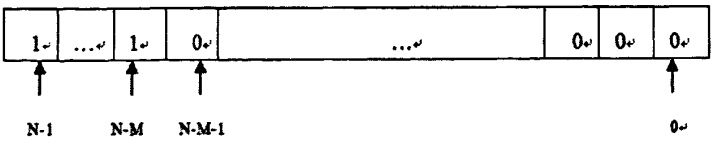


图4-3 寄存器结构

Fig.4-3 Architecture of register

寄存器结构如图 4-3 所示，定义一个寄存器有 N 位，在其中可以任意

放置 1 或 0 在寄存器的 N 个地址里。例如图 4-3 中，从 N-1 到 N-M 位，都放置 1，其他位均放置 0。如果产生的随机数能将寄存器每个地址的遍历，随机输出里面存放的数，则就可以定义这个随机时钟的采样频率为：

$$f_s = \frac{M}{N} \times f_0, \quad M \in Z^+, N \in Z^+, \tag{4-1}$$

其中 f_0 为寄存器工作的时钟频率。

(2) 线性反馈移位寄存器 (LFSR)：用来产生伪随机序列数(二进制序列)。

LFSR 能产生随机性和独立性都非常好的，在它一个周期内所输出的最长序列具有良好的随机特性[24]。虽然在数学上，它还是具有可预见性周期的序列，但是在实际应用中，可以将它看做是随机的。

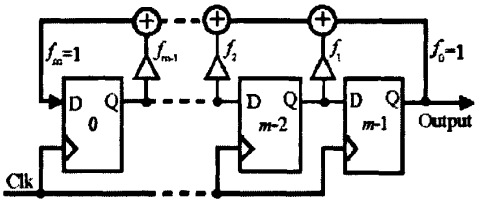


图4-4 LFSR结构图

Fig.4-4 Architecture of LFSR

如图 4-4 所示，当设置不同位置的 D 触发器的值为 0 或者 1 时(不能全为 0，否则会一直输出 0)，通过异或运算，形成适当的反馈方式，则 m 级的 LFSR 能产生序列周期达到最大值 2^m-1，这个伪随机序列就是最长序列。这个最长序列周期和初始状态无关，只与反馈方式有关。不同的反馈方式，有以下特征多项式：

$$p(x) = \sum_{i=0}^{m} f_i x^i = x_m + f_{m-1}x^{m-1} + ... + f_1 x + 1 \tag{4-2}$$

且得到最大 m 级 LFSR 输出最长序列的反馈方式，必须满足条件：

(i) $p(x) = x_m + f_{m-1}x^{m-1} + ... + f_1 x + 1$ 不可化约；

(ii) 不存在 $k < 2^m - 1$ 使得 $(x^k + 1)$ 能被 $p(x)$ 整除； (4-3)

(3) 多路复用器：选择输出 LFSR 产生的伪随机数(十进制数)对应的寄存器相应位的值，从而得到随机时钟信号。通过 LFSR 产生的伪随机序列，通过多路复用器来匹配对应的寄存器的地址。例如寄存器为 N 位，那么

LFSR的级数则为$(\log_2 N)$级。产生的最长随机序列会对应每个地址，在每个最长序列的周期内，平均采样频率由公式(4-1)定义。

4.2 随机时钟在Modelsim上的仿真分析

4.2.1 仿真环境介绍

FPGA的开发环境选用Xilinx的ISE 6.1，设计语言使用硬件描述语言Verilog HDL，仿真环境使用Modelsim5.7[25]。

ISE是集成综合环境的简称，是Xilinx提供的一套工具集，可以完成整个FPGA/CPLD的开发过程[26]。

ISE的主要特点如下：它是一个集成开发环境，集成了众多著名的FPGA/CPLD设计工具，可以显著提高工程师的工作效率。ISE的界面风格简洁流畅，易学易用。ISE秉承了Xilinx设计软件的强大设计辅助功能。在编写代码时，可以使用编写向导生成文件头和模块框架，也可以使用语言模板帮助编写代码。在图形输入时，可以使用ECS的辅助项帮助设计原理图。此外，ISE的Generator和LogiBLOX工具可以方便地生成IP Core与高效模块为用户所用，大大减少了设计者的工作量，提高了设计效率与质量。ISE有丰富的在线帮助信息。

Verilog HDL是目前应用最广泛的硬件描述语言，于1995年成为IEEE标准，可以用于从算法级、门级到开关级的多种抽象层次数字系统设计[27]。Verilog HDL语言具有简洁、高效、易学易用、功能强等特点，与C语言有许多相似之处，并继承和借鉴了C语言的多种操作符和语法结构。由于Verilog HDL巨大的优越性，尤其是在ASIC设计领域更是处于主流地位，在美国、日本等国家，Verilog HDL语言一直是使用最为广泛的硬件描述语言，其使用人数大大超过其它语言的使用人数，在国内，Verilog HDL语言的应用群体也在不断扩大。Verilog HDL硬件描述语言的主要特点如下：

(1) 能形式化地表示电路的结构和行为。

(2) 借助高级语言的结构和语句，例如条件语句、赋值语句和循环语句等，既简化了电路的描述，又方便了设计人员的学习和使用。

(3) 能够在多个层次上对所设计的系统加以描述，设计的规模可以是

任意的，语言不对设计规模施加任何限制。

(4) Verilog HDL 具有混合建模能力，即在一个设计中，各个模块可以在不同设计层次上建模和描述。

(5) 基本逻辑门，例如 and、or 和 nand 等都内置在语言中；开关级结构模型，例如 pmos 和 nmos 等也被内置在语言中，用户可以直接调用。

(6) 用户定义原语(UDP)具有很大的灵活性，用户定义的原语既可以是组合逻辑原语，也可以是时序逻辑原语。

在 FPGA/CPLD 的设计开发过程中，系统的设计仿真是至关重要的[28]。通常，一个设计的仿真将占整个开发的大部分时间。设计仿真按设计步骤可分为功能仿真和时序仿真两部分：功能仿真是在设计输入之后且综合之前进行；时序仿真在综合布局布线之后进行，能够得到目标器件的详细的时序信息。Modelsim 是 Modelsim Technology 公司提供的 HDL 硬件描述语言仿真软件，可以实现 VHDL、Verilog HDL 以及 VHDL—Verilog HDL 混合设计的仿真[29]。除此之外， Modelsim 还能够与 C 语言一起对 HDL 设计文件实现协同仿真。同时，相对于大多数的 HDL 仿真软件来说，Modelsim 在仿真速度上具有明显优势。这些特点使 Modelsim 越来越受到 EDA 设计者、尤其是 FPGA/CPLD 设计者的青睐。本硬件设计中 FPGA 的功能仿真和时序仿真如图 4-5 所示。

4.2.2 伪随机采样硬件选择和电路仿真

本实验中使用 Xilinx Spartan-3 XC3S200FT256 FPGA 电路板产生随机时钟，该电路板工作时钟频率为 50MHz，支持的 LFSR 最长位数为 128 bit，重复周期超过 5000 年，这在实际应用中可以看成是随机性很好的伪随机序列。这里定义寄存器中参数 N=32，M=7，LFSR 产生 5 位二进制的伪随机序列，在 Xilinx ISE 软件中仿真结果如图 4-5 所示。

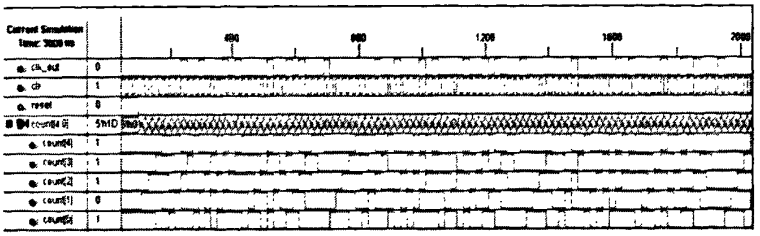


图4-5 伪随机采样电路仿真(N=32，M=7)

Fig.4-5 Simulation of random sampling(N=32, M=7)

图 4-5 中 count[4:0]为 LFSR 产生的 5 位二进制伪随机序列，clk 为工作时钟频率，clk_out 为输出随机时钟。在寄存器高 7 位全部放置 1，即[31:25]位均为 1。当 LFSR 产生的值为 31 到 25 之间的数时，clk_out 就会输出 1。在每个最长序列中，每次输出的高电平可以看成是 ARS 时间序列，$\{\tau_k\}$ 服从[0,32]上的均匀分布。如果要增加输出时钟的随机性，可以通过增加 LFSR 产生的最长序列的长度(增加 N 的值)，和增加相应的 M 的值，则可以产生相同采样频率但随机性更高的随机时钟。(程序见附录 2)

4.3 随机时钟的硬件实现和数字前端的功率消耗比较

4.3.1 AD9225 接口功能特点

FPGA 电路板和 ADC 的连接方式如图 4-6 所示。数字前端中 ADC 使用 AD9225[30]。AD9225 是 ADI 公司生产的单片、单电源供电、12 位精度、25Msps 高速模数转换器,片内集成高性能的采样保持放大器和参考电压源。AD9225 采用带有误差校正逻辑的四级差分流水结构,以保证在 25Msps 采样率下获得精确的 12 位数据。除了最后一级,每一级都有一个低分辨率的闪速 A/D 与一个残差放大器(MDAC)相连[31]。此放大器用来放大重建 DAC 的输出和下一级闪速 A/D 的输入差,每一级的最后一位作为冗余位,以校验数字误差,其结构如图 4-7 所示。

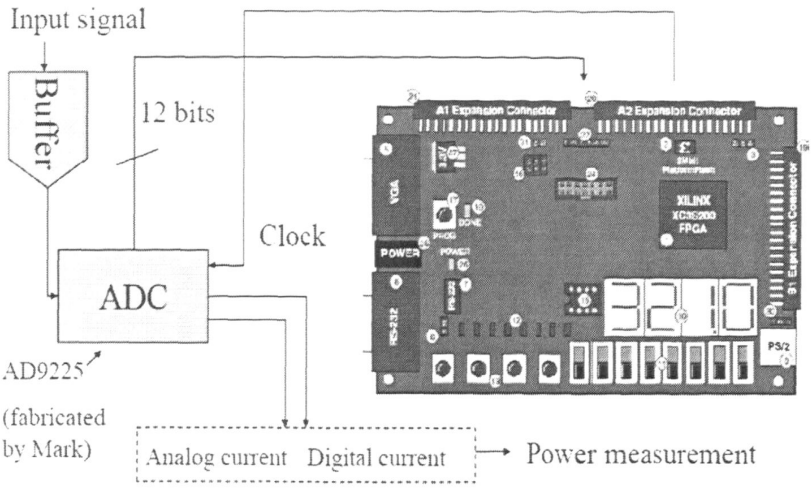


图4-6 FPGA和ADC连接图

Fig.4-6 The platform of FPGA and ADC

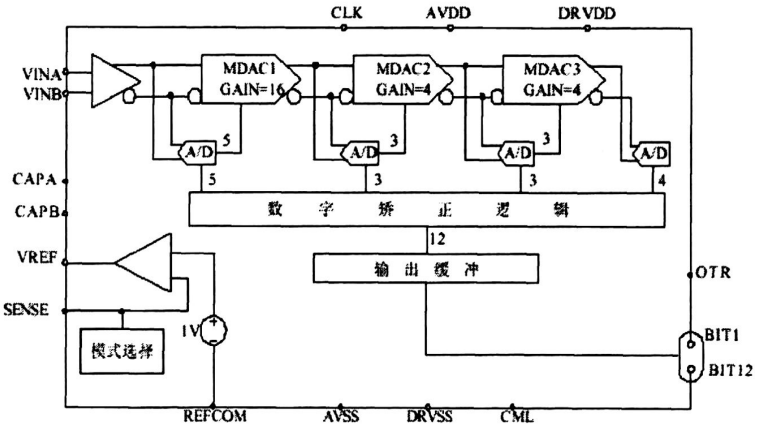


图4-7 AD9225结构图

Fig.4-7 Architecture of AD9225

AD9225 的输入和输出

(1)时钟输入：AD9225 采用单一的时钟信号来控制内部所有的转换，A/D 采样是在时钟的上升沿完成。在 25Msps 的转换速率下,采样时钟的占空比应保持在 45%~55%之间;随着转换速率的降低,占空比也可以随之降低。在低电平期间,输入 SHA 处于采样状态;高电平期间,输入 SHA 处于保持状态。图 4-7 为其时序图[32]。图 4-8 中：

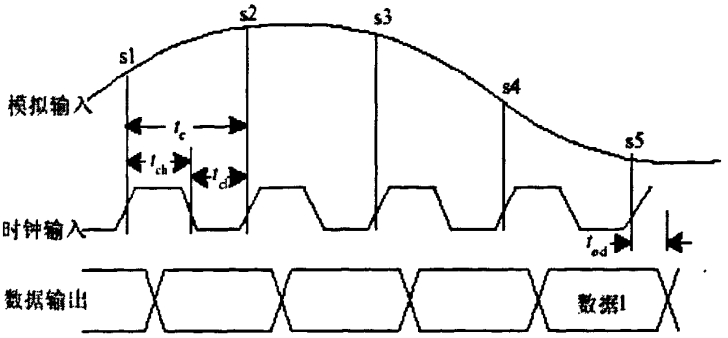


图4-8 AD9225时序图

Fig.4-8 Time sequence of AD9225

tch——高电平持续时间,最小值为 18 ns;

tcl ——低电平持续时间,最小值为 18 ns;

tod——数据延迟时间,最小值为 13 ns。

从时序图可以看出：转换器每个时钟周期(上升沿)捕获一个采样值,三个周期以后才可以输出转换结果。这是由于 AD9225 采用的四级流水结构,虽然可以获得较高的分辨率,但却是以牺牲流水延迟为代价的[33]。

(2)模拟输入：AD9225 的模拟输入引脚是 VINA、VINB,其绝对输入电压范围由电源电压决定：其中, AVSS 正常情况下为 0 V,AVDD 正常情况下为+5 V。AD9225 有高度灵活的输入结构,可以方便地和单端或差分输入信号进行连接。采用单端输入时,VINA 可通过直流或交流方式与输入信号耦合,VINB 要偏置到合适的电压;采用差分输入时,VINA 和 VINB 要由输入信号同时驱动[34]。

(3) 数字输出：AD9225 采用直接二进制码输出 12 位的转换数据,并有一位溢出指示位(OTR),连同最高有效位可以用来确定数据是否溢出。图 4-9 为溢出和正常状态的逻辑判断图[35]。

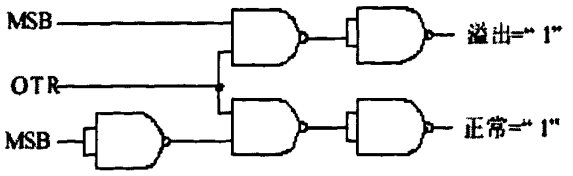


图4-9 溢出和正常状态的逻辑判断图

Fig.4-9 The logic criterion of the overflow and normal status

4.3.2 随机采样与均匀采样的功率消耗分析

在本次实验中，AD9225 输入信号频率为 500KHz，幅度为 2.6V 的正弦信号，通过伪随机采样电路模块，可以产生一系列不同频率的随机时钟和均匀时钟，来实现随机采样和均匀采样，并对同一频率的不同采样方式，测出 ADC 的数字端和模拟端电流的数值。结果如表 4-1 和图 4-10 所示。

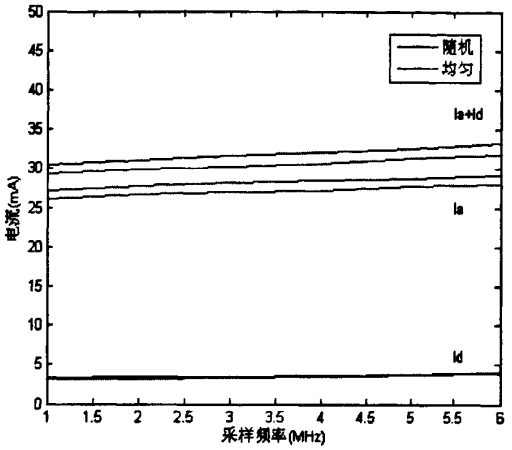


图4-10 ADC数字端和模拟端电流大小

Fig.4-10 The current in digital and analog of ADC

表4-1 不同时钟频率下ADC数字端和模拟端电流(mA)

Table.4-1 The current with different frequencies between digital and analog of ADC

条件	模拟端 I_A	数字端 I_D	I_A+I_D
f_1=6MHz,N=512,M=61,随机时钟	27.95	3.88	31.83
f_1=6MHz,N=512,M=61,均匀时钟	29.23	3.98	33.21
f_2=5MHz,N=512,M=51,随机时钟	27.69	3.67	31.36
f_2=5MHz,N=512,M=51,均匀时钟	28.72	3.78	32.50
f_3=4MHz,N=512,M=41,随机时钟	27.18	3.47	30.65
f_3=4MHz,N=512,M=41,均匀时钟	28.46	3.57	32.03
f_4=3MHz,N=512,M=31,随机时钟	26.93	3.265	30.195
f_4=3MHz,N=512,M=31,均匀时钟	28.21	3.47	31.68
f_5=2MHz,N=512,M=20,随机时钟	26.67	3.16	29.83
f_5=2MHz,N=512,M=20,均匀时钟	27.69	3.37	31.06

f_6=1MHz,N=512,M=10,随机时钟	26.15	3.11	29.26
f_6=1MHz,N=512,M=10,均匀时钟	27.18	3.265	30.445
f_7=500kHz,N=512,N=5,随机时钟	25.64	3.07	28.71
f_7=500kHz,N=512,N=5,均匀时钟	---	----	

从表 4-1 中可以看出，在采样频率相同时，随机采样的电流值要低于均匀采样的电流值；并且随着采样频率的降低，电流值也在降低。当采样频率与信号频率一样时，随机采样有电流值，而均匀采样不能明确读出电流值，说明随机采样对采样频率的限制没有均匀采样大。图 4-8 中，Ia 为模拟端电流值，Id 为数字端电流值。从图中可以明显看出随着采样频率的降低，ADC 数字端和模拟端的电流也随着降低。在同一采样频率时，ADC 中使用随机时钟的电流都低于均匀时钟的值。

4.3.2 ADC 功率的消耗与时钟随机性的关系

为了验证随机性的大小也能决定 ADC 功率的消耗，在伪随机采样电路模块里，分别设置两组 N 和 M 的值。第一组，N=8，M=1；第二组，N=512，M=64，输入信号同上。测得数据如表 4-2 所示。

表4-2 随机性不同的时钟ADC中电流值(mA)

Table.4-2 The current of ADC between different randomized clock

电流	模拟端 I_A	数字端 I_D	I_A+I_D
N=8,M=1	28.980	4.01	32.990
N=512,M=64	28.031	3.92	31.951

从表 4-2 可以得出，利用公式(4-1)，可以计算出采样频率：

$$f_{av}=\frac{M}{N}\times f_0=\frac{1}{8}\times 50MHz=\frac{64}{512}\times 50MHz=6.25MHz$$

在同一采样频率 6.25MHz 下，随机性大的第二组 N 和 M，即 N=512，M=64 时，测得的电流值要小于第一组的值。

4.3.3 与其它方法的性能对比

文献[36][37]提供了另一种随机时钟产生模块，该模块需要利用格雷码计数器产生数个频率相同但相位不同的时钟信号，然后通过 LFSR 产生的伪随机序列选择输出对应相位的时钟信号。文献[36]使用格雷码计数器产生 8 个同频率不同相位的时钟信号，然后将这 8 路时钟混合，通过 LFSR 和多路复用器的随机选择生成随机时钟信号，其随机时钟频率为 $f_{av}=\frac{1}{8}\times 50MHz=6.25MHz$。测得同条件下 ADC 的模拟端和数字端的电流，如表 4-3 所示。

从表 4-3 中的数值可以看出，同样的时钟频率 6.25MHz，采用文献[36][37]中的模块时的电流值与采用本文中的模块在 N=8，M=1 的电流值基本相同。并且该方法局限性大，如要产生随机性大的时钟信号，则需生成很多个同频率不同相位的时钟信号，这样不仅增加了 FPGA 片内资源的消耗，同样增加了 FPGA 本身功率的消耗。相比之下，本文提供的方法灵活性大，能够生成多种频率，随机性也不同的时钟信号，且实现起来简单高效。

表4-3 文献[36]中测量的随机时钟ADC的电流值(mA)

Table.4-3 The current of ADC with the method in[36]

电流	模拟端 I_A	数字端 I_D	I_A+I_D
随机时钟	28.718	4.27	32.988
均匀时钟	29.750	5.31	35.060

4.4 本章小结

综合上述可知，使用随机采样，能放宽采样定理对采样频率的限制，能降低采样频率，在同一采样频率下，不均匀采样能有效降低 ADC 的功率消耗，且随机性越大的随机时钟，功率消耗越低。因此在数字前端中采用随机时钟来进行信号处理能有效降低 ADC 的动态功率消耗。

总 结

非均匀采样理论是数字信号处理领域中一个全新的发展方向，是世界各国科研人员都非常关注的一项课题。同时，非均匀采样理论和技术是一个非常庞大而复杂的研究领域，涉及到的基础学科有测试策略、非线性理论、概率论与随机过程、空间变换理论、信号分析和处理以及多种工程技术学科。由于计算机技术的迅速发展以及人们对信息处理技术愈来愈高的要求，使得人们对非均匀采样理论和技术的研究往纵深方向发展。

本文中我们主要探讨了非均匀采样的基本理论及应用，结合实际工程实践，给出了实现非均匀采样的近似方法——非均匀周期采样，并研制出一套基于 FPGA 的非均匀采样系统。

本文提出了一种新的随机时钟产生方法，能有效利用 FPGA 片内资源和其所支持的线性反馈移位寄存器，使其能够简单高效的生成不同频率和不同随机程度的时钟信号，通过仿真和实验结果表明，采用随机时钟进行采样，不仅能放宽对采样频率的要求，降低采样频率，能有效降低 ADC 的动态功率消耗。而且通过实验发现使用随机性越大的随机时钟，ADC 功率消耗越低，大大提高了数字信号处理设备的工作性能。在利用非均匀采样优点的同时，我们也发现了其具有的一些缺陷，如频谱检测中的频谱噪声和信号恢复很困难等。因此，下一步的研究工作应主要集中于以下几个方面：

(1) 在非均匀采样频谱检测中含有一定程度的频谱噪声，这种现象极大地限制了非均匀采样理论的发展和应用范围。鉴于非均匀采样理论的巨大应用前景，必须找到处理频谱噪声的方法，这样能更有助于提高数字信号处理的精确度，同时降低数字信号处理设备的功率消耗。

(2) 找到非均匀采样频谱计算的快速算法。正如 FFT 极大地推动了传统数字信号处理理论的应用和发展一样，非均匀采样理论能够得到广泛使用和飞速发展的前提是其必须存在频谱计算的快速算法。但是，非均匀采样与均匀采样相比，有着很大的不同，如不满足均匀采样中的时移关系等等，因此，均匀采样理论中的频谱计算快速算法不能应用于非均匀采样，必须重新寻找非均匀采样频谱计算的快速算法。

(3) 如何由非均匀采样的离散信号重构模拟信号。信号重建是信号处理领域中一个

重要方面，无论采样什么处理方法，都需要在处理后进行信号的恢复。目前，对于非均匀采样得到的离散信号，还没有一个可以普遍使用的信号恢复方法。

(4) 非均匀采样信号在时域及频域下的能量对应关系。

上述四个方面中，前四个方面是当前人们都很感兴趣的理论研究方向，它们之间的关系不是独立的，而是相互联系、相互促进的，对这些问题进行广泛、深入和细致的研究，无论在理论上还是在实际应用上都具有重大而深远的意义。

参考文献

[1] A.J. Jerri. The Shannon Sampling Theorem - Its Various Extensions and Applications : Atutorial Review. Proceedings of the IEEE[J]. 1977, Vol. 65, no. 11, pp. 1565-1596.

[2] A.J. Jerri. The Shannon Sampling Theorem - Its Various Extensions and Applications : Atutorial Review. Proceedings of the IEEE[J]. 1977, Vol. 65, no. 11, pp. 1565-1596.

[3] Chinnery D and Keutzer K Closing the Gap Between ASIC and Custom Tools and Techniques for High—Performance ASIC Design[M]. Netherland: Kluwer Academic Publishers, 2002: 157-158.

[4] Sliman Kadi M, Brasen D, and Saucier G. A fast—FPGA prototyping system that uses inexpensive high perform ance FPIC. Proc. 2nd Annual W orkshop on FPGAs[J], Berkeley, 1994: 147—156.

[5] Birkner J and Chua H T. Programmable array logic circuit. U. S. Patent, 4124899, 1978.

[6] Kuon I and Rose J. Measuring the gap between FPGAs and ASICs[C]. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 2007, 26(2): 203-215.

[7] Hamdy E and McCollum J. Dielectric based antifuse for logic and memory IC[C]. International Electron Devices Meeting Technical Digest, San Francisco, 1988: 786—789.

[8] Birkner J and Chan A. A very high speed field programmable gate array using metalto-metal antifuse programmable elements[J]. Microelectronics Journal, 1992, 23(7): 561—568.

[9] Altera Corporation . Hardcopy series handbook . http : //www.Altera.co-in.cn/literature/hb/hrd/hc—handbook.pdf, 2008, 9.

[10] Brown S and Rose J. FPGA and CPLD architectures[J]. A tutorial. IEEE Design and Test of Computers, 1996, 12(2): 42-57.

[11] Guterman D C and Rimawi L H. An electrically alterable nonvolatile memory cell using a floating-gate structure[J]. IEEE Transactions on Electron Devices, 1997, 26(4): 576-586.

[12] Marvasti. Nonuniform Sampling: Theory and Practice[M]. Kluwer Academic/Plenum Publishers, 2001.

[13] Y. Rahmat-Samii, R.L.T. Cheung. Nonuniform Sampling Technique for Antenna Applications. IEEE Transactions on Antennas and Propagation[J]. March 1987, Vol. AP-35, no.3.

[14] O.A.Z. Leneman. Random Sampling of Random Processes. Impulse Processes. Information and Control[J]. 1966, Vol. 9, no.4, pp. 347-363.

[15] H.S. Shapiro, R.A. Silverman. Alias-Free Sampling of Random Noise. Journal Society for Industrial and Applied Mathematics[J]. 1960, Vol. 8, no. 2, pp. 225-248.

[16] I. Bilinskis, A. Mikelsons. Randomized Signal Processing[M]. Prentice Hall. 1992.

[17] R.J. Martin. Irregularly Sampled Signals: Theories and Techniques for Analysis[D]. PhD.Thesis. University College, 1998.

[18] D. M. Bland and A. Tarczynski. The effect of sampling jitter in a digitized signal[C]. In Proceedings of the 1997 IEEE International Symposium on Circuits.

[19] J.J.Wojtiuk, Randomized Sampling for Radio Design[D], Ph.D.Thesis, University of South Australia, School of Electrical and Information Engineering, 2000.

[20] Haïfa FARES, Manel BEN-ROMDHANE, Chiheb REBAI, Non Uniform Sampled Signal Reconstruction for Software Defined Radio Applications[C], in IEEE 2008 International Conference on Signals, Circuits and Systems.

[21] Jeffrey J.Wojtiuk and Richard J.Martin. Random Sampling Enables Flexible Design for Multiband Carrier Signals[J]. IEEE TRANSACTIONS ON SIGNAL PROCESSING, VOL. 49, NO. 10, OCTOBER 2001

[22] V.K. Dao, Q.D. Bui, C.S. Park, A multi-band 900 MHz/1.8 GHz/5.2 GHz LNA for reconfigurable radio[J], IEEE Radio Frequency Integrated Circuits Symposium (2007) 69–72.

[23] 吕辉，何晶，王刚.伪随机序列中本原多项式生成算法[J].计算机工程，2004，30(16)：34-46.

[24] 束礼宝，宋克柱，王砚方.伪随机数发生器的 FPGA 实现与研究[J].电路与系统学报，2003.6(8),03.121-124.

[25] 王建，白瑞林.基于 FPGA 的智能控制器设计及测试方法研究[J].微计算机信息，2005，(36)：128-130.

[26] 曾繁泰，陈美金，沈卫红.EDA 工程方法学[M].北京：清华大学出版社,2003.

[27] 吉国凡，赵智昊，杨嵩.基于 ATE 的 FPGA 测试方法[J].电子测试，2007，(12)：43-46.

[28] 贾秋玲.基于 Matlab7.x/Simulink/Stateflow 系统仿真分析与设计[M].西安：西北工业大学出版社，2006.

[29] 刘宏杰.可编程逻辑器件的 VHDL 语言优化设计方法[J].测控技术,2001,20(6)：32-34.

[30] Bings, J.P., Gadlage , M.J., Clark, S.D. Total dose results for the AD9225RH analog-to-digital converter[J].Radiation Effects Data Workshop, IEEE 2003. pp.38–42.

[31] D. Macq, P.G.A. Jespers, A 10-bit pipelined switched-current A/D converter[J], IEEE Journal of Solid-State Circuits 29 (8) (1994) 967–971.

[32] A. Zanchi, F. Tsay, A 16 bit 65-MS/s 3.3-V pipeline ADC core in SiGe BiCMOS with 78-dB SNR and 180-fs Jitter[J], IEEE Journal of Solid-State Circuits 40 (6) (2005) 1225–1237.

[33] Yun-Shiang Shu, Bang-Sup Song, A 15-bit linear 20-MS/s pipelined ADC digitally calibrated with signal-dependent dithering[J], IEEE Journal of Solid- State Circuits 43 (2) (2008) 342–350.

[34] Jian Li, Xiaoyang Zeng, Lei Xie, Jun Chen, Jianyun Zhang, Yawei Guo, A 1.8-V 22-mW 10-bit 30-MS/s pipelined CMOS ADC for low-power subsampling applications[J], IEEE Journal of Solid-State Circuits 43 (2) (2008) 321–329.

[35] Junhua Shen, P.R. Kinget, A 0.5-V 8-bit 10-MS/s pipelined ADC in 90-nm CMOS[J], IEEE Journal of Solid-State Circuits 43 (4) (2008) 787–795.

[36] Chiheb Rebai, Manel Ben-Romdhane&al , Pseudorandom signal sampler for relaxed design of multistandard radio receiver[J], Microelectronics Journal 40 (2009), pp.991-999.

[37] Chiheb Rebai, Manel Ben-Romdhane&al, Pseudorandom signal sampler for relaxed design of multistandard radio receiver[J], Microelectronics Journal 40 (2009), pp.991-999.

攻读硕士学位期间发表的学术论文

邓潇宇，戴青云，钟润阳.数字前端功耗降低方法的仿真研究.计算机仿真(已录用)

致　谢

本文是在戴青云教授的悉心指导下完成的。导师敏锐的洞察力、渊博的学识、严谨的治学态度、高瞻远瞩的开拓目光令我敬佩至深、受益匪浅。在本课题研究过程以及论文撰写阶段，导师提出了很多独到的见解以及宝贵的建议，并从各方面为我提供研究和学习的条件。值此论文付梓之际，向恩师致以衷心的感谢和崇高的敬意！

另外，我还要特别感谢同学对我实验以及论文写作的指导，他们为我完成这篇论文提供了巨大的帮助。还要感谢，王锐煌、郭浩威、王贤伟、李旭明和曹璐同学对我的无私帮助，使我得以顺利完成论文。

最后，还要对在法国一起共同完成这项课题的法国朋友表示感谢，他们的科研方法、编程思想让我深受启发。

特别感谢父母及亲人对我一直以来的支持和爱护！

衷心感谢所有给予我帮助和关心的老师、同学和朋友们！

邓潇宇

2011年5月8日于广州

附录 A

随机时钟在 FPGA 上的实现

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter_combination is
Generic(
    num : integer range 0 to 1022 := 10; -- numerateur du rapport de decimation
    den : integer range 0 to 2048 := 1024 -- denominateur du rapport de decimation
);
port(
          clk,reset   :    in std_logic;
        data_in   :    in std_logic_vector(11 downto 0);
          clk_out     :    out std_logic;
            data_out :    out std_logic_vector(11 downto 0);
            time_out :    out integer
      );
end counter_combination;
architecture Behavioral of   counter_combination is
signal count                 :std_logic_vector (9 downto 0);
signal linear_feedback :std_logic;
signal var_clk_out       :std_logic := '0';
signal data_var            :std_logic_vector(11 downto 0);
signal time_var            : integer:='0';
--constant enable           :std_logic='1';
```

```
begin
--linear_feedback <= not(count(7) xor count(5)xor count(4)xor count(3));
linear_feedback <= not(count(9) xor count(6));
process(clk,reset) is
VARIABLE urne : std_logic_vector(den-1 DOWNTO 0) := (others => '0');
VARIABLE num_case : integer range 0 TO den-1;
VARIABLE urne_0 : std_logic := '0';
VARIABLE T          : integer := '0';
begin
                    if (reset = '1') then
                        data_var <="000000000000";
            urne(den-1 DOWNTO den-1-20*num) := (others => '0');
            urne(den-1-num*20-1 DOWNTO den-1-num*20-1-num) := (others => '1');
            urne(den-1-num*20-1-num-1 DOWNTO 0) := (others => '0');
                        count <= (0=>'1',others=>'0');
                        elsif (clk'event and clk='1') then
                count <= (count(count'high - 1 downto 0)&linear_feedback);
                        num_case := conv_integer(unsigned(count)) ;
                        urne_0 := urne(num_case);
                        T=T+1;
                         if urne_0 = '1' THEN
          var_clk_out <= '1';
                                data_var <=data_in;
                                 time_var <= T;
            else
              var_clk_out <= '0';
              end if;
                    end if;

clk_out <= var_clk_out;
```

```
data_out<=data_var;
time_out<=time_var;
end process;
end Behavioral;
```

附录 B

随机时钟的发生程序

```
-------------------------------------------------------
---Pseudorandom signal sampler .vhd
---pss_top
---4th,June
-------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

Entity PSS_circuit IS
port(
        clk,reset :in std_logic;
        Pss_out :out std_logic);
END pss_circuit;

ARCHITECTURE struct OF PSS_circuit IS
-------------------------------------------------------------------
-------------------------------------------------------------------
COMPONENT counter_combination
            port( clk:      in std_logic;
                        reset:   in std_logic;
                        Ph0:     out std_logic;
                        Ph1:     out std_logic;
                        Ph2:     out std_logic;
                        Ph3:     out std_logic;
                        Ph4:     out std_logic;
```

```
                Ph5:    out std_logic;
                Ph6:    out std_logic;
                Ph7:    out std_logic;
                clk_8: out std_logic);
end COMPONENT;
-----------------------------------------------------------------
COMPONENT   lfsr
        port(   clk:    in std_logic;
                reset: in std_logic;
                enable: in std_logic;
                count_lfsr: out std_logic_vector( 2 downto 0 ));
end COMPONENT;
-----------------------------------------------------------------
COMPONENT selector
        port(       clk,reset : in std_logic;
                    select_num : std_logic_vector( 2 downto 0 );
                    input_ph0    :in std_logic;
                    input_ph1    :in std_logic;
                input_ph2    :in std_logic;
                input_ph3    :in std_logic;
                input_ph4    :in std_logic;
                input_ph5    :in std_logic;
                input_ph6    :in std_logic;
                input_ph7    :in std_logic;
                    output_ph    :out std_logic);
end COMPONENT;
-----------------------------------------------------------------
signal clk_8_output : std_logic;
signal ph_0,ph_1,ph_2,ph_3,ph_4,ph_5,ph_6,ph_7 : std_logic;
signal count_lfsr_int : std_logic_vector( 2 downto 0 );
```

```
signal reset_not : std_logic;
constant enable: std_logic :='1';
Begin
--------------------------------------------------------------------------
reset_not <= not reset;
counter_combination_inst :     counter_combination
port MAP (
                                clk =>clk,
                                reset => reset_not,
                                Ph0 =>ph_0,
                                Ph1 =>ph_1,
                                Ph2 =>ph_2,
                                Ph3 =>ph_3,
                                Ph4 =>ph_4,
                               Ph5 =>ph_5,
                               Ph6 =>ph_6,
                               Ph7 =>ph_7,
                                clk_8 => clk_8_output);
--------------------------------------------------------------------------
--------------------------------------------------------------------------
LFSR_inst    : lfsr
port MAP (
              enable => enable,
              clk =>clk_8_output,
           reset => reset_not,
           count_lfsr => count_lfsr_int);
--------------------------------------------------------------------------
--------------------------------------------------------------------------
selector_inst :    selector
port MAP    (
```

```
                clk => clk,   --clk_8_output,
                reset => reset_not,
              input_ph0 => ph_0,
                input_ph1 =>   ph_1,
                input_ph2   => ph_2,
                input_ph3 => ph_3,
                input_ph4 => ph_4,
                input_ph5   =>   ph_5,
                input_ph6   => ph_6,
                input_ph7   => ph_7,
                select_num => count_lfsr_int,
                output_ph => Pss_out);
-------------------------------------------------------------------------------
END struct;
-------------------------------------------------------
-----counter and combining function
-----generate 8 different phases clock with same frequency
-------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity counter_combination is
----generic(nb_bit : integer:=3
----------state     : integer range 0 to 7 );

port(
        clk,reset   :    in std_logic;
        clk_8       :    out std_logic;
        Ph0         :    out std_logic;
        Ph1         :    out std_logic;
        Ph2         :    out std_logic;
```

```
        Ph3     :   out std_logic;
        Ph4     :   out std_logic;
        Ph5     :   out std_logic;
        Ph6     :   out std_logic;
        Ph7     :   out std_logic);
end counter_combination;
architecture Behavioral of  counter_combination is
signal ph0_int: std_logic;
signal ph1_int: std_logic;
signal ph2_int: std_logic;
signal ph3_int: std_logic;
signal ph4_int: std_logic;
signal ph5_int: std_logic;
signal ph6_int: std_logic;
signal ph7_int: std_logic;
signal counter_state: integer :=0;
signal clk_8_int: std_logic;
begin
process(clk,reset) is
    begin
                if  clk'event and clk='1' then
                    if reset='1' then counter_state<=0;
                    else
          if (counter_state<7) then
                     counter_state<=counter_state+1;
                     else counter_state<=0;
                     end if;
                     case counter_state is
 when  0=>ph0_int<='1';ph1_int  <='0';ph2_int  <='0';ph3_int<='0';ph4_int  <='0';ph5_int
<='0';ph6_int<='0';ph7_int<='0';
```

```
when 1 =>ph0_int<='1';ph1_int <='1';ph2_int <='0';ph3_int<='0';ph4_int <='0';ph5_int
<='0';ph6_int<='0';ph7_int<='0';
when 2 =>ph0_int<='1';ph1_int <='1';ph2_int <='1';ph3_int<='0';ph4_int <='0';ph5_int
<='0';ph6_int<='0';ph7_int<='0';
when 3 =>ph0_int<='1';ph1_int <='1';ph2_int <='1';ph3_int<='1';ph4_int <='0';ph5_int
<='0';ph6_int<='0';ph7_int<='0';
when 4 =>ph0_int<='0';ph1_int <='1';ph2_int <='1';ph3_int<='1';ph4_int <='1';ph5_int
<='0';ph6_int<='0';ph7_int<='0';
when 5 =>ph0_int<='0';ph1_int <='0';ph2_int <='1';ph3_int<='1';ph4_int <='1';ph5_int
<='1';ph6_int<='0';ph7_int<='0';
when 6 =>ph0_int<='0';ph1_int <='0';ph2_int <='0';ph3_int<='1';ph4_int <='1';ph5_int
<='1';ph6_int<='1';ph7_int<='0';
when 7 =>ph0_int<='0';ph1_int <='0';ph2_int <='0';ph3_int<='0';ph4_int <='1';ph5_int
<='1';ph6_int<='1';ph7_int<='1';
when others =>ph0_int<='0';ph1_int <='0';ph2_int <='0';ph3_int<='0';ph4_int
<='0';ph5_int<='0';ph6_int<='0';ph7_int<='0';
                                end case;
                                 if (counter_state<4) then clk_8_int <='1';
                                 else clk_8_int <='0';
                                 end if;
                        end if;
                    end if;
end process;
Ph0 <= ph0_int;
Ph1 <= ph1_int;
Ph2 <= ph2_int;
Ph3 <= ph3_int;
Ph4 <= ph4_int;
Ph5 <= ph5_int;
Ph6 <= ph6_int;
```

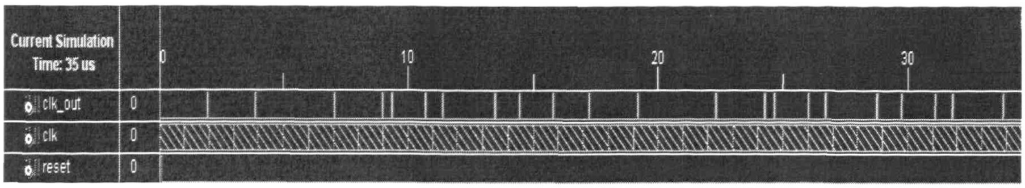
```
Ph7 <= ph7_int;
clk_8 <=clk_8_int;
end Behavioral;
----------------------------------------------
  ---- Design Name : lfsr
  ---- Function      : Linear feedback shift register
 ----------------------------------------------
   library ieee;
       use ieee.std_logic_1164.all;
  entity lfsr is
     port (
       count_lfsr    :out std_logic_vector (2 downto 0);
       enable :in   std_logic;                          -- Enable counting
        clk      :in   std_logic;                          -- Input rlock
        reset   :in   std_logic                          -- Input reset
     );
  end entity;
  architecture rtl of lfsr is
       signal count                :std_logic_vector (2 downto 0);
       signal linear_feedback :std_logic;
begin
    linear_feedback <= not(count(2) xor count(1));
    process (clk, reset) begin
         if (reset = '1') then
              count <= (0=>'1',others=>'0');
          elsif (clk'event and clk='1') then
               if (enable = '1') then
                    count <= (count(1)&count(0)&linear_feedback);
                end if;
          end if;
```

```
    end process;
    count_lfsr <= count;
 end rtl;
-------------------------------------------------
---Name: selective Combiner
---function: select the phase among eight different phases
-------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity selector is
        port (
                clk,reset       :in std_logic;
                select_num      :in std_logic_vector(2 downto 0);
                input_ph0       :in std_logic;
                input_ph1       :in std_logic;
                input_ph2       :in std_logic;
                input_ph3       :in std_logic;
                input_ph4       :in std_logic;
                input_ph5       :in std_logic;
                input_ph6       :in std_logic;
                input_ph7       :in std_logic;
                output_ph       :out std_logic);
end selector;
architecture Behavioral of selector is
signal    output_ph_int       : std_logic;
begin
        process (clk,reset)
            begin
                    if (clk'event and clk='1') then
                        if reset='1' then output_ph_int <= input_ph0;--_int;
```

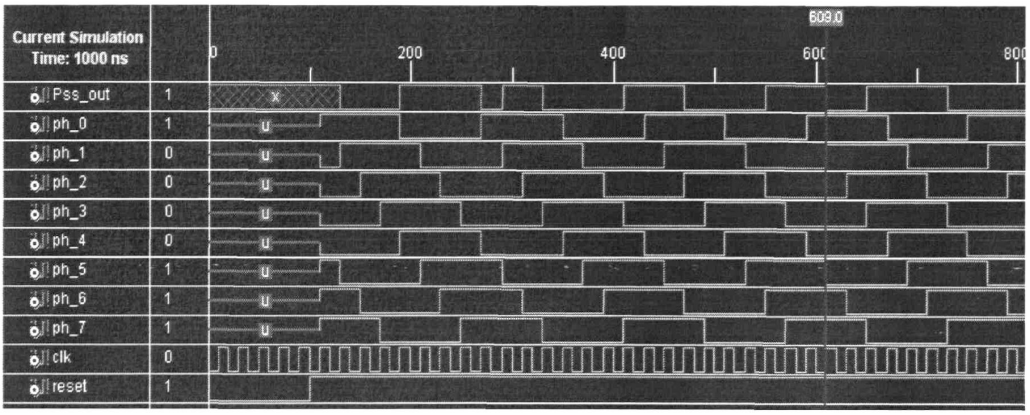
```
                elsif   select_num  ="001"   then   output_ph_int  <=
input_ph1;--_int;
                elsif   select_num  ="010"   then   output_ph_int  <=
input_ph2;--_int;
                elsif   select_num  ="011"   then   output_ph_int  <=
input_ph3;--_int;
                elsif   select_num  ="100"   then   output_ph_int  <=
input_ph4;--_int;
                elsif   select_num  ="101"   then   output_ph_int  <=
input_ph5;--_int;
                elsif   select_num  ="110"   then   output_ph_int  <=
input_ph6;--_int;
                elsif   select_num  ="111"   then   output_ph_int  <=
input_ph7;--_int;
                else    output_ph_int <= input_ph0;--_int;
            end if;
        end if;
end process;
output_ph <= output_ph_int;
end Behavioral;
```

附录 C

仿真结果图



附录 A 仿真结果图



附录 B 仿真结果图