一种基于 FPGA 的高斯随机数生成器的设计与实现

谷晓忱 张民选
(国防科学技术大学计算机学院 PDL 重点实验室 长沙 410073)

摘 要 基于 FPGA 的高斯随机数生成器需要满足可重构、高吞吐率和高硬件资源使用效率等要求. 文中提出了一种易于硬件实现的状态转换逻辑结构, 并给出了均匀分布随机数周期和输出位宽的配置方法和配置原则. 文中详细分析了应用"最值分析法"和"静态误差分析法"求解 Box Muller 算法实现过程中各操作数位宽的具体过程. 硬件实现结果在 Xilinx Vertex 5 上的工作速度为 491 MHz, 吞吐率为 9.82×10^8 samples/second, 硬件资源使用效率为 2.085×10^6 samples/second/slice. 文中作者使用 DIEHARD 测试集、χ^2 和 K-S 方法对产生的随机数质量进行了检测, 文中给出了结果.

关键词 现场可编程门阵列; 硬件加速器; 高斯随机数产生; 均匀分布随机数产生; 可重构计算
中图法分类号 TP302 **DOI** 号: 10.3724/SP.J.1016.2011.00165

Design and Implementation of a FPGA Based Gaussian Random Number Generator

GU Xiao-Chen ZHANG Min-Xuan
(PDL, School of Computer, National University of Defense Technology, Changsha 410073)

Abstract A good FPGA based Gaussian Random Number Generator has to be reconfigurable for different applications, while running fast and acquiring a high throughput per slice. In this paper, a hardware structure of the Transform Module is proposed. The configuration process and principle of the period and output bit-width of uniform random numbers are introduced. For the generation of Gaussian random numbers, methods for bit-width optimization of the fixed point operands are introduced in details. Implementation results on Xilinx Vertex 5 FPGA show that the proposed hardware design could acquire a frequency as high as 491MHz, while the corresponding throughput is 9.82×10^8 samples/second and the throughput per slice is 2.085×10^6 samples/second/slice. DIEHARD test suit, χ^2 and K-S method are used to test the quality of the generated random numbers here.

Keywords FPGA; hardware accelerator; Gaussian random number generation; uniform random number generation; reconfigurable computing

1 引 言

随着 FPGA 性能的不断提高, 基于 FPGA 的计算加速已经逐渐成为提高计算速度和计算效率的重要手段之一. 近年来的很多研究成果[1-2] 表明, 现阶段适合在 FPGA 上进行计算加速的应用都具有计算密集型的特点, 例如蒙特卡罗计算和图像处理等. 在这些应用中, 大都存在随机采样的过程, 因此, 随机数生成器是这些应用中不可或缺的组件. 高斯随机数是应用最为广泛的一类随机数, 所以, 研究基于 FPGA 的高斯随机数生成器具有非常重要的实际意义.

收稿日期: 2010-07-13; 最终修改稿收到日期: 2010-08-30. 本课题得到国家"八六三"高技术研究发展计划项目基金(2009AA01Z124, 2009AA01Z104, 2009AA01Z102)资助. 谷晓忱, 男, 1980 年生, 博士, 主要研究方向为数模混合集成电路设计、射频集成电路设计和可重构计算. E-mail: specialsjtu@163.com. 张民选, 男, 1954 年生, 教授, 博士生导师, 主要研究领域为高性能计算机体系结构、微电子技术.

近年来,有很多关于在 FPGA 上实现高斯随机数生成器的研究[3-5]. Lee 等人[5] 应用 Box Muller 方法在 Xilinx V4 FPGA 上产生高斯随机数,并对设计过程进行了分析; Cheung 等人[4] 应用 Inversion 方法设计高斯随机数生成器,并在 Xilinx V4 和 V2 两个型号的 FPGA 上进行了实现; Thomas 等人[3] 应用"Central limit"原理设计的高斯随机数生成器(在 Xilinx V5 上实现)可以达到 397MHz 的工作频率. 这些研究都在不同方面取得了相应的成果,为后续研究提供了重要的参考. 但是,他们的研究仍然存在很多缺陷,其中最重要的一点就是:已有的研究只关注对高斯随机数产生算法的设计实现过程,基本不关心均匀分布随机数的产生过程.

除了 Wallace 等少数方法以外,大多数高斯随机数产生算法都是基于"转换"的思想得到高斯随机数的,即高斯随机数是由均匀分布随机数通过某种算法转换得到的. 因此,这些算法的实现前提是需要有一组满足质量要求的均匀分布随机数. 上述对高斯随机数生成器的研究中,大都将均匀分布随机数生成器作为已知条件看待,没有将均匀分布随机数生成器的设计作为高斯随机数产生的一部分来进行深入的研究.

均匀分布随机数生成器对高斯随机数产生过程的影响主要体现在以下 3 个方面:

(1) 均匀分布随机数的周期决定了高斯随机数的周期,基于 FPGA 的计算加速应用中,需要保证随机数的周期具有可配置属性,这点主要体现在均匀分布随机数的设计中;

(2) 均匀分布随机数的位宽决定了可以产生的高斯随机数的范围,即某些小概率事件的产生需要依靠对均匀随机数位宽的正确配置来实现;

(3) 均匀分布随机数生成器的工作速度不能低于高斯随机数产生算法的硬件工作速度,为了不降低最终产生高斯随机数的吞吐率,均匀分布随机数生成器需要具有结构简单、工作速度快的特点.

本文的研究将均匀分布随机数的产生过程考虑到高斯随机数生成器的设计过程中,综合考虑均匀分布随机数生成器对高斯随机数产生过程的影响,有针对性地解决了上述 3 个方面的问题. 本文通过对 Combined Tausworthe 均匀分布随机数生成器的改进,提出了一种基于矩阵思想设计转换逻辑 $\boldsymbol{A}^s$ 的硬件结构,提高了均匀分布随机数的产生速度. 通过对均匀分布随机数生成器的周期和输出位宽进行配置,使得高斯随机数的周期和产生范围具有可配置的属性,可以满足不同应用环境的需求.

2 随机数产生算法

2.1 均匀分布随机数产生算法 Combined Tausworthe[6]

本文应用 Combined Tausworthe 算法产生均匀分布随机数. Combined Tausworthe 随机数生成器由若干个 Tausworthe 随机数生成器组成,每个 Tausworthe 随机数生成器产生的随机数可以表示成如下形式

$$u_{n,j}=\sum_{i=1}^{L}x_{j,ns+i-1}\times 2^{-i} \tag{1}$$

其中,L 和 s 都是非零正整数,s 是跳变步长,L 是随机数的输出位宽. $x_{j,ns+i-1}$ 是一个随机序列,其递归关系式为

$$x_{j,n}=a_1x_{j,n-1}\oplus a_2x_{j,n-2}\oplus\cdots\oplus a_kx_{j,n-k} \tag{2}$$

其中,a_i 是特征多项式 $P(z)=z^k-a_1z^{k-1}-\cdots-a_k$ 的系数. 当 $P(z)$ 为本原多项式时,输出序列的周期取得最大值: 2^k-1. $P(z)$ 通常取三项式: $P(z)=z^k-z^q-1$.

由 J 个 Tausworthe 随机数生成器构成的 Combined Tausworthe 随机数生成器所产生的(0, 1)区间内的均匀分布随机数可以表示为

$$u_n=\sum_{i=1}^{L}(x_{1,ns_1+i-1}\oplus x_{2,ns_2+i-1}\oplus\cdots\oplus x_{J,ns_J+i-1})\times 2^{-i} \tag{3}$$

其中,x_{j,ns_1+i-1} 是第 j 个 Tausworthe 随机数生成器的输出.

2.2 高斯随机数产生方法 Box-Muller[7]

Box-Muller 方法可以将均匀分布随机数转换为高斯分布随机数,它依据的原理是:两组相互独立的高斯分布随机数的平方和服从指数分布. 其转换过程可以表示为

$$\alpha=\sqrt{-2\times\ln(u_1)}\sin(2\pi u_2),$$
$$\beta=\sqrt{-2\times\ln(u_1)}\cos(2\pi u_2) \tag{4}$$

其中,u_1 和 u_2 是两路均匀分布随机数,α 和 β 是得到的高斯分布随机数.

3 硬件结构

3.1 整体架构

图 1 所示为高斯随机数生成器的整体架构. 该架构分为两个部分:虚线框内为 Box Muller 算法实现部分;虚线框以外为均匀分布随机数产生部分. Box Muller 算法实现部分需要使用到指数函数、对

数函数以及三角函数运算模块, 这些模块可以采用查表和函数最佳一致逼近等方法设计实现, 也可以采用 Xilinx 公司等提供的 IP 核实现. 由于 Box Muller 算法需要使用两路独立的均匀分布随机数, 所以图 1 所示架构中需要使用两个 Combined Tausworthe 均匀分布随机数生成器.

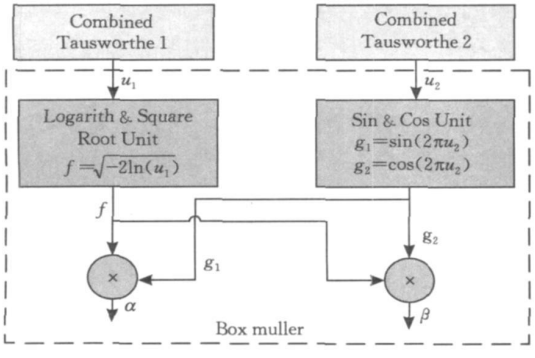

图 1　高斯随机数生成器整体架构

图 2 所示为 Combined Tausworthe 随机数生成器的结构. 由该结构产生的随机数满足式(3)的约束关系. 最终的输出 u_n 是 J 个 Tausworthe 随机数生成器的输出的异或结果. 每个 Tausworthe 随机数生成器由一个 L 位的寄存器和转换逻辑 $\boldsymbol{A}^s$ 构成. 寄存器中的每一位数据都满足式(2)的约束关系. 转换逻辑 $\boldsymbol{A}^s$ 根据这一约束关系以及式(1)的组合关系求出下一时刻的输出 $u^{n+1,j}$.

在高斯随机数生成器的设计过程中, Box Muller 算法实现部分的难点在于如何确定各操作数的位宽, 因为各操作数的位宽决定了最终产生的高斯随机数的正确性和精度; 而均匀分布随机数产生部分的难点在于转换逻辑 $\boldsymbol{A}^s$ 的设计, 因为转换逻辑需要保证在每个时钟周期内产生一个新的随机数 $u_{n,j}$, 所以 $\boldsymbol{A}^s$ 的硬件复杂程度和工作速度就决定了整个 Combined Tausworthe 随机数生成器的性能.

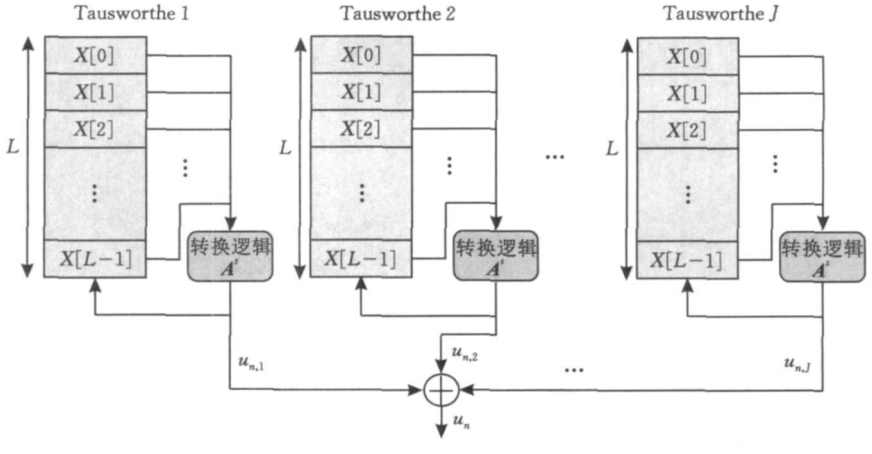

图 2　Combined Tausworthe 随机数生成器的结构

3.2　转换逻辑 $\boldsymbol{A}^s$ 的设计

一般情况下, 式(2)所示的递推关系大都使用线性反馈移位寄存器来实现. 此时, 每个时钟周期只能产生一位有效的数据信息 $x_{j,n}$, 而根据式(1)的约束关系, 产生一个 L 位的随机数 u_{nj} 需要使用 s 位新的数据信息, 也就是说, 需要 s 个时钟周期才能产生一个随机数 $u_{n,j}$. 因此, 为了在每个时钟周期内可以产生一个 L 位的随机数 u_{nj}, 就需要改变线性反馈移位寄存器的反馈逻辑, 使其可以在一个时钟周期内产生 s 位有效数据信息.

式(2)的递推关系可以用矩阵形式表示为

$$X_{j,n+1}=\boldsymbol{A}\times X_{j,n} \tag{5}$$

其中, $X_{j,n}$ 是当前时刻的 L 位状态信息, $X_{j,n+1}$ 是下一时刻的 L 位状态信息, $\boldsymbol{A}$ 是状态转换矩阵, 其形式为

$$\boldsymbol{A}=\begin{pmatrix}\boldsymbol{C}_{1\times L}\\ \boldsymbol{I}_{(L-1)\times(L-1)} \quad \boldsymbol{0}_{(L-1)\times 1}\end{pmatrix} \tag{6}$$

其中, $\boldsymbol{C}_{1\times L}$ 是特征多项式 $P(z)$ 的系数向量, $\boldsymbol{I}_{(L-1)\times(L-1)}$ 是一个单位矩阵, $\boldsymbol{0}_{(L-1)\times 1}$ 是一个零向量.

根据式(6)的状态转换关系, 包含 s 位新的有效数据位的状态信息可以表示为

$$\begin{aligned}X_{j,n+s}&=\boldsymbol{A}\times X_{j,n+s-1}\\&=\boldsymbol{A}\times\left(\boldsymbol{A}\times X_{j,n+s-2}\right)=\cdots=\boldsymbol{A}^s\times X_{j,n}\end{aligned} \tag{7}$$

即

$$u_{n+1,j}=\boldsymbol{A}^s\times u_{n,j} \tag{8}$$

也就是说, 如果使用转换矩阵 $\boldsymbol{A}^s$ 对当前时刻的输出 $u_{n,j}$ 进行状态转换, 就可以直接得到下一时刻的输出 $u_{n+1,j}$. 关于转换矩阵 $\boldsymbol{A}^s$ 的求解过程, 本文作者在文献[8]中已经有详细说明, 这里不再赘述.

根据式(9)的状态转换关系,可以得到图3所示的硬件结构. 转换矩阵 A^s 中每个"1"都对应一个异或门. $P(z)$ 通常取三项式: $P(z)=z^k-z^q-1$. 因此, $w^{n+1,j}$ 中新产生的 s 位数据信息都是由 $u^{n,j}$ 中的某两位数据异或得到的. A^s 的硬件结构中只包含 s 个异或逻辑门,同时这 s 个异或逻辑门都是并联关系,因此,该结构可以达到较高的工作速度.

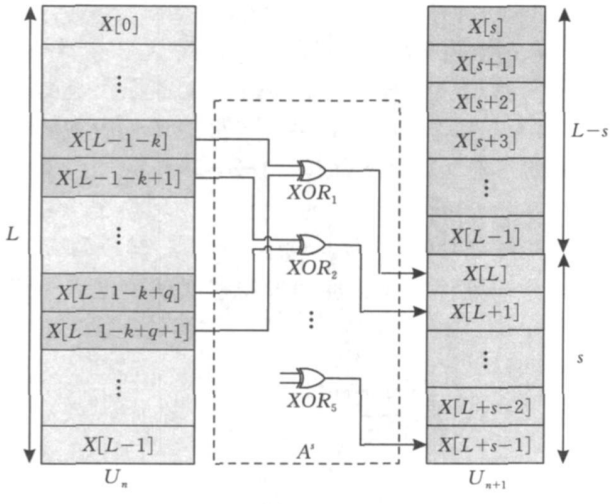

图3 转换逻辑 A^s 的内部结构

3.3 周期的配置

由 J 个 Tausworthe 生成器构成的 Combined Tausworthe 随机数生成器所产生的随机数的周期为[6]

$$T=(2^{k_1}-1)\times(2^{k_2}-1)\times\cdots\times(2^{k_J}-1) \quad (9)$$

其中, k_j 为 $P_j(z)$ 的级数. 可见,当 k_j 和 J 值增加时,周期 T 也随之增加. 此处,均匀分布随机数的周期 T 也就是最终产生的高斯分布随机数的周期,因此,可以根据实际需要改变 k_j 和 J 的值,从而达到配置周期的目的.

周期的配置原则: 在实际的应用中,不同的需求对随机数周期的要求不同. 例如在蒙特卡罗计算中,当随机数周期是实际使用的随机数数量的2次方或者3次方以上时就能保证结果的正确性[9]. 过短的周期将导致计算结果的错误,但是过长的周期则意味着随机数生成器的实现需要消耗更多的硬件资源. 所以,在设计高斯随机数生成器时,应该根据实际需要选取 J 和 k_j,在保证计算结果正确的前提下,达到消耗最少硬件资源的目的.

3.4 均匀随机数位宽的配置

输出位宽 L 的可配置性表现在参数组合 (L, k_j, q_j, s_j) 的选取上. 设计 Combined Tausworthe 随机数生成器时,可以根据设计需求选取输出位宽 L,然后按照文献[6]中给出的搜索算法得到满足"ME(Maximally Equidistributed)条件"的参数组合 (k_j, q_j, s_j). 需要指出的是,并不是所有的 L 都能找到满足 ME 条件的 (k_j, q_j, s_j) 参数组合. 当没有参数组合可以满足 ME 条件时,文献[6]中也给出了一个折中的办法: 假设 $l>L$,且 (l, k_j, q_j, s_j) 对应的 CTRNG 满足 ME 条件,那么可以选取该 Combined Tausworthe 随机数生成器输出中的 L 位作为新的输出,构成 L 位输出的 RNG. 所以,在设计 Combined Tausworthe 随机数生成器时,总可以根据实际需要来选取合适的输出位宽 L,实现硬件设计的可配置性.

4 操作数位宽选取与误差分析

根据图1所示架构进行硬件设计时,另一个设计难点就是如何确定 Box Muller 算法实现部分各操作数的位宽. 图1硬件结构中的所有操作数都采用定点数的表式方式. 每一个定点数都可以分成3个组成部分: 符号位、整数部分和小数部分. 本文中,使用 I_x 表示操作数 x 的整数部分位宽,使用 F_x 表示操作数 x 的小数部分位宽. 以下内容将分别给出确定 I_x 和 F_x 的具体方法.

4.1 整数部分位宽选取

本文使用"最值分析法"确定各操作数的整数部分位宽,即根据 Combined Tausworthe 生成器的输出 u_1 和 u_2 以及 Box Muller 算法中的运算关系,求解每个操作数可能取得的最大(小)值,进而确定每个操作数的整数部分位宽. 这里以求解 I_α 为例进行说明.

根据式(4)以及图1所示结构可知:

$$\alpha=f\times g_1 \quad (10)$$

因为,

$$|g_1|=|\sin(2\pi u_2)|\le 1 \quad (11)$$

所以可以得到如下关系,

$$|\alpha|=|f\times g_1|\le|f| \quad (12)$$

如图1所示,

$$f=\sqrt{-2\ln(u_1)} \quad (13)$$

所以,

$$\max(f)=\sqrt{-2\ln(\min(u_1))} \quad (14)$$

u_1 是 $(0, 1)$ 区间内的小数,因为 Combined Tausworthe 生成器的输出位宽为 L,所以可得

$$\min(u_1)=2^{-L} \quad (15)$$

即

$$\max(f) = \sqrt{-2\ln(2^{-L})} \tag{16}$$

将式(16)的结论代入式(12)可以得到如下结论,

$$|\alpha| \le \sqrt{-2\ln(2^{-L})} \tag{17}$$

$|\alpha|$ 的最大值与 I_α 的关系为

$$\max(|\alpha|) = 2^{I_\alpha} - 1 \tag{18}$$

因此, 为了避免计算过程中 α 出现溢出现象, I_α 的取值应该为

$$I_\alpha = \left\lceil \log_2 \sqrt{-2\ln(2^{-L})} + 1 \right\rceil \tag{19}$$

根据式(19)就可以算出 I_α 的值. 例如, 当 $L = 32$ 时,

$$I_\alpha = \left\lceil \log_2 \sqrt{-2\ln(2^{-32})} + 1 \right\rceil = \lceil \log_2^{7.66} \rceil = 3 \tag{20}$$

上述即为利用"最值分析法"确定操作数 α 的整数部分位宽 I_α 的求解过程. 根据式(10)~式(19)的推导过程, 可以确定图 1 硬件结构中所有操作数的整数部分位宽. 应用该方法, 每个操作数的整数部分位宽都可以与 Combined Tausworthe 生成器的输出 u_1 和 u_2 建立数值运算关系. 这也表明了 u_1 和 u_2 的位宽对最终产生的高斯随机数的重要影响, 说明了 3.4 小节中对均匀随机数位宽的配置是非常重要的.

4.2　小数部分位宽选取

整数部分位宽 I_x 决定了各操作数的取值范围, 而小数部分的位宽 F_x 决定了各操作数的计算精度. F_x 的取值越大, 对应的计算精度就越高, 但同时对应的硬件实现结果所消耗的资源也就越多. 因此合理选取 F_x 的原则就是: 在满足计算精度要求的前提下, 通过选取各操作数的小数部分位宽 F_x, 使得实现结果达到消耗最少硬件资源的目的.

本文采用"静态误差分析法"[5]来求解各操作数的小数部分位宽 F_x, 即应用误差传递原理, 推导出各操作数的量化误差对最终输出结果的影响, 而后应用组合最优化问题的求解方法, 得到各操作数的最优小数位宽组合. 下面将以求解 F_f、F_{g_1} 和 F_{g_2} 的过程为例进行说明.

在下面的分析过程中, 将 f、g_1 和 g_2 的求解模块当做一个 IP 核看待, 其计算精度分别可以达到 2^{-F_f}、$2^{-F_{g_1}}$ 和 $2^{-F_{g_2}}$. 根据计算精度的要求, α 和 β 的最终结果需要精确到 $2^{-F_{\alpha,\beta}}$.

经过量化后操作数可以表示为

$$\tilde{x} = x + E_x \tag{21}$$

其中, $\tilde{x}$ 为该操作数的真实值, x 为该操作数的量化值, E_x 为量化误差. 如果采用截断方式对操作数进行量化处理, 那么量化误差 E_x 在数值上等于操作数的计算精度 2^{-F_x}. 当进行乘法操作时, 误差的传递关系可以表示为

$$\begin{aligned}\tilde{x} \times \tilde{y} &= (x + E_x) \times (y + E_y) \\ &= x \times y + x \times E_y + y \times E_x + E_x \times E_y\end{aligned} \tag{22}$$

即

$$E_{\tilde{x} \times \tilde{y}} = x \times E_y + y \times E_x + E_x \times E_y \tag{23}$$

因此, 根据图 1 所示的硬件结构关系, α 和 β 的计算误差可以表示为

$$\begin{aligned}E_\alpha &= f \times E_{g_1} + g_1 \times E_f + E_{g_1} \times E_f, \\ E_\beta &= f \times E_{g_2} + g_2 \times E_f + E_{g_2} \times E_f\end{aligned} \tag{24}$$

硬件实现中, α、β 以及 g_1、g_2 分别取相同的位宽, 将各操作数的计算精度代入式(24)中可以得到如下关系

$$2^{-F_{\alpha,\beta}} \ge f_{\max} \times 2^{-F_{g_{1,2}}} + g_{1,2_{\max}} \times 2^{-F_f} + 2^{-F_{g_{1,2}}} \times 2^{-F_f} \tag{25}$$

其中, $f_{\max}$ 和 $g_{1,2_{\max}}$ 可以根据 4.1 小节的"最值分析法"求得. 因此, 根据式(11)的不等式就可以计算出 F_f、F_{g_1} 和 F_{g_2} 的取值范围.

根据 F_f、F_{g_1} 和 F_{g_2} 的取值范围, 可以有多种取值组合. 位宽的增加, 意味着对应模块消耗硬件资源的增加. 根据前面提出的 F_x 选取原则, 应该选取硬件资源消耗最少的取值组合. 对于复杂的组合最优化问题, 可以采用遗传算法或者模拟退火算法等最优化算法寻找最优解. 对于式(25)这样相对简单的问题, 可以采用穷举法获得最优组合.

例如, 若 α(β)的精度要求为 2^{-16}, $u_1(u_2)$ 的位宽 L 等于 32. 根据式(25)的约束关系可以得到如下不等式

$$2^{-16} \ge f_{\max} \times 2^{-F_{g_{1,2}}} + g_{1,2_{\max}} \times 2^{-F_f} + 2^{-F_{g_{1,2}}} \times 2^{-F_f} \tag{26}$$

根据"最值分析法"可以计算出 $f_{\max} \approx 6.66$, $g_{1,2_{\max}} = 1$. 而后通过穷举所有可能的最优组合, 可以得到 $F_f = 17$, $F_{g_{1,2}} = 20$.

应用上述"静态误差分析法"以及 F_x 的选取原则, 可以确定图 1 所示硬件结构中所有操作数的小数部分位宽.

5　硬件实现与结果分析

本文的设计使用 Verilog 语言对硬件逻辑进行描述, 综合工具选用 Xilinx 公司的 ISE 11.5.

5.1　均匀分布随机数生成器

本文实现了 4 个 Combined Tausworthe 随机

数生成器的硬件设计,它们具有不同的周期和输出位宽,其配置参数如表 1 所示.

表 1[6] **Combined Tausworthe** 随机数生成器的配置参数

RNG	配置参数	
Combined Tausworthe 1	(k_1, k_2, k_3)	(31, 29, 28)
	(q_1, q_2, q_3)	(12, 4, 17)
	(s_1, s_2, s_3)	(13, 23, 3)
Combined Tausworthe 2	(k_1, k_2, k_3, k_4)	(31, 29, 28, 25)
	(q_1, q_2, q_3, q_4)	(6, 2, 13, 3)
	(s_1, s_2, s_3, s_4)	(18, 2, 7, 13)
Combined Tausworthe 3	(k_1, k_2, k_3)	(63, 58, 55)
	(q_1, q_2, q_3)	(5, 19, 24)
	(s_1, s_2, s_3)	(24, 13, 7)
Combined Tausworthe 4	(k_1, k_2, k_3, k_4)	(63, 58, 55, 47)
	(q_1, q_2, q_3, q_4)	(31, 19, 24, 21)
	(s_1, s_2, s_3, s_4)	(18, 28, 7, 8)

这 4 个随机数生成器在 Xilinx Vertex 6 FPGA 上的硬件实现结果如表 2 所示. 位宽 L 的值可以根据文献[6]中提供的参数确定. 如表 2 所示,本文分别实现了两个 32 位和两个 64 位的 Combined Tausworthe 随机数生成器. 根据 3.4 小节给出的位宽配置原则,表 2 所示的 4 个均匀随机数生成器可以被配置成输出位宽不大于 32 位和不大于 64 位的任意输出位宽的均匀分布随机数生成器.

表 2 **Combined Tausworthe** 随机数生成器的硬件实现结果

RNG	L	J	T	Slices	Frequency/MHz
Combined Tausworthe 1	32	3	2^{88}	30	1113
Combined Tausworthe 2	32	4	2^{113}	34	1113
Combined Tausworthe 3	64	3	2^{176}	52	1113
Combined Tausworthe 4	64	4	2^{223}	64	1113

根据式(9)可知,周期的大小由 k_j 和 J 的值确定. 结合表 1 的设计参数,可以得到表 2 中所列出的各 Combined Tausworthe 随机数生成器的周期. 文献[6]中提供了多种不同的 k_j 和 J 的配置组合,所以在不同的应用环境中,根据 3.3 小节给出的周期配置原则,可以根据实际需求对 Combined Tausworthe 随机数生成器的周期进行重新配置.

从表 2 的硬件实现结果可以看出,所设计的 4 个 Combined Tausworthe 随机数生成器消耗的硬件资源都非常少,而且都获得了很高的工作频率. 这主要是因为 3.2 小节中提出的转换逻辑 $\boldsymbol{A}^s$ 具有非常简单而且易于硬件实现的结构. 根据图 3 所示的结构,$\boldsymbol{A}^s$ 中仅包含 s 个二输入异或门,而且它们都是并联关系. 因此,无论 L 与 s 的值怎样变化,$\boldsymbol{A}^s$ 的硬件实现结果都只有一级异或逻辑. 也是由于这个原因,表 2 中所示的 4 个 Combined Tausworthe 随机数生成器才会有相同的工作频率.

5.2 高斯分布随机数生成器

本文使用表 1 中所列出的 Combined Tausworthe1 作为均匀分布随机数生成器,应用 Box Muller 算法,在 Xilinx FPGA 上实现了高斯随机数生成器的硬件设计. 硬件设计中采用定点方式表示各操作数,最终输出(α 和 β)需要精确到 2^{-16}. 设计中采用 4.1 和 4.2 小节给出的"最值分析法"和"静态误差分析法"确定各操作数的位宽. 硬件设计过程中对复杂的运算部件(如乘法等)进行了流水化处理. 表 3 是高斯随机数生成器的硬件实现结果. 为了便于跟其它参考设计进行比对,本文分别给出了该硬件设计在 Vertex 4 和 Vertex 5 上的综合结果,该综合结果在 Modelsim 中进行了功能验证.

表 3 几种高斯随机数生成器的实现结果比较

Design	Method	Device	Slices	Block RAMs	DSPs	Frequency/MHz	Samples/cycle	Throughput (million samples/sec)	Throughput/slice (million samples/sec/slice)
本文	Box Muller	V4	993	0	8	418	2	836	0.842
		V5	471	0	8	491	2	982	2.085
		CPU(3GHz)	N/A	N/A	N/A	N/A	2	2.1	N/A
Lee[5]	Box Muller	V4	1452	3	12	375	2	750	0.517
Cheung[4]	Inversion	V4	579	1	4	370	1	370	0.639
Thomas[3]	Central limit	V5	774	4	0	397	1	397	0.5

注:"Device"一栏中的 V4 和 V5 指的是 Xilinx Vertex 4 和 Vertex 5 FPGA.

本文主要从以下两个方面对表 3 的硬件实现结果进行分析:

(1) 硬件实现方式与软件实现方式的性能比较. 本文用 C 语言对图 1 的硬件结构进行了编程,并将软件运行于主频为 3GHz 的 Intel 处理器上. 从表 3 列出的软/硬件运行结果可见,硬件实现方式产生随机数的吞吐率是软件实现方式产生随机数吞吐率的 400 多倍. 因为硬件运算可以保证每个时钟周期都产生一个输出结果,而软件运算需要经过若干条指令的运行才能得到一个最终结果,因此使用硬件运算可以获得很好的加速效果. 文献[5]中甚至获得了千余倍的加速比. 虽然软件运行方式也可以

通过各种手段提高其运算速度, 但是硬件运算还是可以保证获得 1 至 2 个数量级的加速比[5]. 所以, 使用硬件实现方式可以更好地满足高吞吐率的要求.

(2) 本文实现结果与其它硬件实现结果的性能比较. 表 3 中列出了已报道的其它几种高斯随机数生成器的硬件实现结果. 本文主要从硬件资源使用情况(slices、BRAMs 和 DSPs)、吞吐率(Frequency、Throughput) 和硬件资源使用效率(Throughput/slice) 3 个方面进行比对.

硬件资源的使用情况跟所使用的算法有关. Box Muller 算法中包含较多的算术运算(如乘法等) 过程, 因此从表 3 给出的结果可见, 本文和 Lee 等人设计的 Box Muller 随机数生成器比使用 Inversion和 Central Limit 方法设计的随机数生成器要消耗更多的 DSP 资源. 但是, 本文的设计所消耗的硬件资源要少于 Lee 等人的设计, 尤其是逻辑运算部分(slices 部分), 本文的设计所消耗的 slice 资源仅是 Lee 等人的设计的 2/3. Lee 等人的设计也是采用 Combined Tausworthe 方法生成均匀分布随机数, 但是他们的硬件设计采用的是文献[6]中给出的算法. 这种算法适合于 PC 机上的指令运算, 当进行硬件设计时, 该算法就需要消耗比较多的硬件资源.

吞吐率是衡量随机数生成器性能的重要指标之一. 它主要取决于随机数生成器的工作速度(时钟频率). 但是, 由于 Box Muller 算法可以在每拍产生两个有效输出, 所以, 它在算法上具有独特的优势. 从表 3 中列出的数据可以看出, 采用 Box Muller 算法所获得的吞吐率要高于其它算法. 因为其它算法的工作频率需要达到 Box Muller 算法的两倍以上才能具有高于 Box Muller 算法的吞吐率, 而在同等的硬件实现环境中, 这是很难实现的. 同样, 本文的硬件设计所获得的吞吐率要高于 Lee 等人的设计. 这同样是因为在设计 Combined Tausworthe 随机数生成器时本文采用了不同于 Lee 等人的设计思路. 文献[6]中给出的实现算法需要三次移位操作、两次异或操作和一次与操作才能得到下一时刻的输出 $u_{n+1,j}$, 在硬件设计时, 其逻辑级数要远大于 3.2 小节中所给出的转换逻辑 $\boldsymbol{A}^s$ 的逻辑级数. 因此, 其工作速度要低于本文的设计.

硬件资源使用效率是衡量基于 FPGA 的随机数生成器时所使用的一个特殊的指标. 它使用平均到每个 slice 上的吞吐率来衡量硬件实现结果对资源的使用效率. 由于 FPGA 中包含有 slice、BRAM 和 DSP 等多种硬件资源, 所以使用哪种类型的硬件资源作为衡量标准一直是一个比较难以达成一致的问题. 根据众多参考文献的记录[3-5,8], 一般是将对 DSP 和 BRAM 等硬件资源的使用看做是对运算部件和存储部件的一种特殊实现方式, 而使用 slice 资源(逻辑部件) 作为衡量硬件资源使用效率的参考标准, 即用单位 slice 资源上的吞吐率来衡量设计随机数生成器时对硬件资源的使用效率. 由于本文的设计对 Combined Tausworthe 随机数生成器的实现结构和 Box Muller 算法实现部分各操作数位宽进行了合理的优化, 因此可以在消耗比较少的 slice 资源的前提下获得较高的工作频率. 从表 3 所给出的数据可以看出, 本文的设计获得了明显高于其它几种实现结果的硬件资源使用效率.

5.3　随机数质量检测

对随机数的质量检测可以分成两部分进行: 一是检测由 Combined Tausworthe 方法产生的均匀随机数的质量; 二是检测最终产生的高斯随机数的质量.

本文使用著名的随机数质量检测软件 DIEHARD① 对 Combined Tausworthe 方法产生的均匀随机数的质量进行了测试. 测试结果显示: Combined Tausworthe 方法产生的均匀随机数可以通过 DIEHARD 测试集[10] 中所有测试程序的检测.

图 4 所示为本文设计的高斯随机数生成器所产生的 100000 个随机数的概率密度函数图, 可见其实测结果与理论结果基本保持一致. 同时, 本文使用 χ^2 和 K-S 两种测试方式对这 100000 个随机数的质量进行检测[11]. 根据计算出的 p 值可以得出结论: 本文设计的高斯随机数生成器所产生的随机数符合 (0, 1) 正态分布.

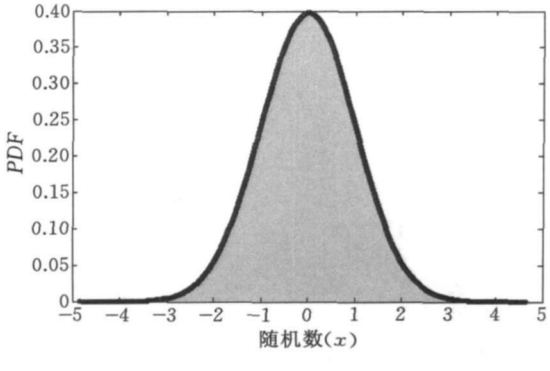

图 4　100000 个随机数的概率密度函数图

① http://stat.fsu.edu/~geo/diehard.html

6 总 结

本文首先应用矩阵的形式分析了 Combined Tausworthe 随机数生成器中的核心部件——状态转换逻辑 A^s 的特征,提出了一种只需要使用 s 个二输入异或门来实现 A^s 的硬件结构. 而后,给出了 Combined Tausworthe 随机数生成器中周期和输出位宽的配置方法和配置原则. 在 Box Muller 算法的硬件实现部分,本文给出了应用"最值分析法"和"静态误差分析法"确定各操作数位宽的具体分析过程,并计算出了均匀分布随机数的位宽 L 与高斯分布随机数输出范围的关系. 在 Xilinx Vertex 6 FPGA 上的硬件实现结果表明:应用本文提出的转换逻辑 A^s 所设计实现的 Combined Tausworthe 随机数生成器可以在消耗较少硬件资源的前提下获得较高的工作速度. 通过与其它几种高斯随机数生成器的硬件实现结果进行比对,本文设计的高斯随机数生成器可以实现更高的工作速度和获得更优的硬件资源使用效率. 本文的硬件设计所产生的高斯随机数可以通过 χ^2 和 K-S 测试方法的检验.

参 考 文 献

[1] Jason C et al. Accelerating Monte Carlo based SSTA using FPGA//Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays. Monterey, California, USA, 2010: 111-114

[2] Luu J et al. FPGA-based Monte Carlo computation of light absorption for photodynamic cancer therapy//Proceedings of the 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines. Napa, CA, 2009: 157-164

[3] Thomas D B, Howes L, Luk W. A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation//Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays. Monterey, California, USA, 2009: 63-72

[4] Cheung R C C et al. Hardware generation of arbitrary random number distributions from uniform distributions via the inversion method. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2007, 15(8): 952-962

[5] Lee D U et al. A Hardware Gaussian noise generator using the box-muller method and its error analysis. IEEE Transactions on Computers, 2006, 55(6): 659-671

[6] Pierre L E. Maximally equidistributed combined tausworthe. Mathematics of Computation, 1996, 65(213): 203-213

[7] Box G E P, Muller M E. A note on the generation of random normal deviates. Annals of Mathematical Statistics, 1958, 29(2): 610-611

[8] Gu X, Zhang M. Uniform random number generator using leap-ahead LFSR architecture//Proceedings of the International Conference on Computer and Communications Security (ICCCS 2009). Hong Kong, China, 2009: 150-154

[9] Dalal I L, Stefan D. A hardware framework for the fast generation of multiple long-period random number streams//Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays (FPGA'2008). Monterey, California, USA, 2008: 245-254

[10] Pierre L E, Richard S. TestU01: A C library for empirical testing of random number generators. ACM Transactions on Mathematical Software(TOMS), 2007, 33(4): Atrtical 22

[11] Agostino R B D, Stephens M A. Goodness-of-Fit Techniques. New York: Marcel Dekker Inc, 1986

GU Xiao-Chen, born in 1980, Ph.D.. His research interests include mixed signal IC design, RFIC design and reconfigurable computing.

ZHANG Min-Xuan, born in 1954, professor, Ph. D. supervisor. His current research interests include high performance computer architecture and micro-electronics.

Background

This paper is supported by National High Technology Research and Development Program (863 Program) of China (2009AA01Z124, 2009AA01Z104, 2009AA01Z102).

The research work of this paper is part of the project that is being carried on by Institute for Integrated Micro and Nano Systems (IMNS) of Edinburgh University. The aim of the project is to explore the performance of FPGA based accelerators which have Monte Carlo based applications running on them. Monte Carlo method is computation intensive and could be well accelerated on FPGAs. There is a FPGA based

reconfigurable supercomputer called MAXWELL designed by Edinburgh University. The research works are mostly implemented on this platform. The research group has done some works about financial computation and gene analysis and published more than 20 papers in international conferences and journals which are cited by SCI and EI.

The authors' work is to do some research on FPGA based financial Monte Carlo acceleration. Financial Monte Carlo computation is widely used in risk analysis, price estimation and so on. Real-time computation is important because the sooner the price is calculated, the more valuable it is. Black Scholes and GARCH algorithms have been implemented on single and multiple FPGAs and very high speedups have been acquired.

This paper proposes a FPGA based Gaussian random number generator which is a core module in most Monte Carlo applications. Gaussian random numbers are widely used in scientific computations. The FPGA based Gaussian random number generatior has to be reconfigurable for different applications, while running fast and acquiring a high throughput per slice. The proposed hardware structure in this paper could well satisfy these requirements.