基于Zynq平台的BFV全同态加密算法高效实现

杨亚涛[1,2]，曹景沛[2]，陈亮宇[1]，王伟[2]
（1.北京电子科技学院电子与通信工程系，北京 100070；2.西安电子科技大学通信工程学院，陕西 西安 710071）

摘　要：针对BFV全同态加密算法，在Zynq平台上设计了一种高效实现方案。该方案结合负包裹卷积与数论变换（NTT）算法，优化并加速了多项式乘法的过程。同时采用流水线设计思想和并行化硬件电路架构，加速BFV算法的RNS实现。系统采用AXI-DMA传输机制高效地实现了ARM和FPGA之间数据传输。在Zynq UltraScale+MPSoC ZCU102平台上测试，系统在200 MHz时钟频率下，执行一次同态加法的平均耗时为0.024 ms；执行一次同态乘法的平均耗时为5.779 ms，其中包括0.874 ms的密文传输时间。与SEAL库和OpenFHE库的实现相比，所提方案的同态加法实现了4.63倍和6.79倍的效率提升，同态乘法实现了4.43倍和2.95倍的效率提升，这为全同态加密算法的实际工程实现提供了重要参考。

关键词：全同态加密；BFV算法；剩余数系统；数论变换；硬件实现
中图分类号：TN92
文献标志码：A
DOI：10.11959/j.issn.1000−436x.2024160

Efficient implementation for BFV fully homomorphic encryption algorithm based on Zynq platform

YANG Yatao[1,2], CAO Jingpei[2], CHEN Liangyu[1], WANG Wei[2]
1. Department of Electronic and Communication Engineering, Beijing Electronic Science and Technology Institute, Beijing 100070, China
2. School of Telecommunication Engineering, Xidian University, Xi'an 710071, China

Abstract: An efficient implementation scheme for the BFV fully homomorphic encryption algorithm was proposed on the Zynq platform. This scheme effectively integrated the negative wrapped convolution with the number theoretic transform (NTT) algorithm, optimizing and accelerating the polynomial multiplication process. Furthermore, it adopted a pipeline design and parallel hardware architecture to enhance the RNS implementation of the BFV algorithm. The system efficiently implemented data transmission between the ARM processor and FPGA using the AXI-DMA transfer mechanism. Test results on the Zynq UltraScale+MPSoC ZCU102 platform show that the system performs a homomorphic addition in just 0.024 ms and a homomorphic multiplication in 5.779 ms at a 200 MHz clock frequency, which includes 0.874 ms for ciphertext transmission. Compared to the implementations of the SEAL and OpenFHE libraries, the proposed scheme achieves efficiency improvements of 4.63 and 6.79 times for homomorphic addition, and 4.43 and 2.95 times for homomorphic multiplication, providing an important reference for the practical engineering implementation of fully homomorphic encryption algorithms.

Keywords: fully homomorphic encryption, BFV algorithm, residual number system, number theoretic transform, hardware implementation

收稿日期：2024−05−10；修回日期：2024−08−17
基金项目：北京市自然科学基金资助项目（No.4232034）；中央高校基本科研业务费资金资助项目（No.3282024058, No.3282024052）
Foundation Items：Beijing Natural Science Foundation (No.4232034), The Fundamental Research Funds for the Central Universities (No.3282024058, No.3282024052)

0　引言

在传统的云计算领域中，云服务器为用户提供了丰富的存储资源和计算资源。但云端通常只能处理未加密的明文数据。因此，为了保护数据隐私性，数据在云端存储时多以加密形式存在，这就带来了一个挑战，即数据的计算和分析需要先对加密数据进行解密，但这一过程增加了数据泄露的风险。

全同态加密（FHE, fully homomorphic encryption）技术允许数据在加密状态同时进行计算，解密的结果与在未加密数据上执行的结果完全一致[1]，解决了隐私数据在云计算中的困境。全同态加密技术展现出巨大的应用潜力，但在实际应用中仍面临着效率低下的问题。目前，同态加密算法多依赖于软件算法库的实现，如SEAL库[2]、OpenFHE库等。这些基于库的实现在安全性和通用性方面表现良好，但在效率方面普遍不如硬件实现。

硬件实现的典型代表一一现场可编程门阵列（FPGA, field programmable gate array），在适应全同态加密算法方面受到其资源限制的挑战。这种复杂性主要体现在一些高度零碎化、多样性强的操作上[3]。目前，全同态加密算法在FPGA上实现主要集中在高效的数论变换（NTT, number theoretic transform）模块、多项式运算模块、同态乘法模块或加解密模块，而并非整个全同态加密算法。为了寻求有效的工程解决方案，许多研究者发现使用ARM+FPGA或CPU+FPGA等异构平台是实现完整全同态加密算法的新路径。本文的主要贡献如下。

1) 设计了一种基于Zynq平台的BFV（Brakerski-Fan-Vercauteren）全同态加密算法高效实现方案。通过整合负包裹卷积（NWC, negetive wrapped convolution）和数论变换算法，优化并加速了多项式乘法的过程；利用可变滑动窗口法改进了模约减步骤。此外，通过增加流水线的深度和采用并行化的硬件架构，设计了7组剩余多项式计算单元，每个计算单元内置2个NTT蝶形运算核心，提高了同态乘法的计算效率。在数据流传输方面，采用AXI-DMA传输机制优化信号传输的时序，实现ARM和FPGA之间的高效数据传输。并通过以太网TCP实现客户端与服务器之间的稳定通信。

2) 基于TCP从端到端实现了完整的BFV全同态加密算法。在Zynq UltraScale+MPSoC ZCU102平台上进行硬件测试，本文方案实现的同态加法运算平均耗时仅为0.024 ms；同态乘法运算的平均耗时为5.779 ms，这其中包括0.874 ms的密文传输时间。与SEAL库和OpenFHE库相比，这一测试结果在同态加法上分别实现了4.63倍和6.79倍的效率提升，在同态乘法上分别实现了4.43倍和2.95倍的效率提升。

1　相关研究

当前CPU中普遍缺乏硬件模算术单元[3]，导致基于模算术单元的同态加密方案在软件实现上性能受限。基于环上带错误学习（RLWE, ring learning with error）问题构建的同态加密方案，操作对象是具有大整数系数的多项式。相较基于错误学习（LWE, learning with error）问题构建的同态加密方案，RLWE方案更为安全、高效，但其涉及大规模的多项式算术运算，这些高昂的计算开销阻碍了同态加密技术在实际应用中的进程。

自Dijk等[4]首次提出全同态加密方案以来，诸多同态加密算法的研究不断涌现。Gentry等[5]在IMB X3500服务器上实现了全同态加密方案，但却引发了密钥存储的问题，且该方案实现的时间代价十分高昂，如运行一次Bootstrapping操作的执行时间高达30 min。近些年来，同态加密算法的硬件加速实现研究随着算法的优化持续发展推进，主要分为以下三类。

1) 基于FPGA的全同态加密算法加速器。Roy等[6]提出了用于多项式环全同态加密方案的硬件实现架构。该方案的目标算法为YASHE。作者以FPGA作为硬件平台，有效结合数论变换和负循环卷积[7]算法，设计了一种软硬件协同实现的方案。在后续的研究中，Roy等[8]将硬件加速方案架构放在BFV上，提出了一个基于FPGA的BFV方案全同态加密加速器的架构，该加速器应用了Halevi等[9]提出的算术优化技术。谢刚[10]在基于FPGA的BFV同态计算加速器的设计与实现的研究中，采用蒙哥马利模约减加速多项式乘法中的系数规约。其增加了算法实现的复杂度，需要对输入和输出进行额外的域转换处理，以及对于某些特定的模数，该算法的效率可能不是最优的。此外，由于文献[8]直接将密文存放在DDR中，而文献[10]中客户端和服务器之间采用的是串口协议进行密文与密钥的

传输，通信效率受限且与实际应用场景不符。

2) 基于图形处理单元（GPU, graphics processing unit）的全同态加密算法加速器。Badawi 等[11]提出了一种基于图形处理单元的同态加密实现方案。该方案采用计算统一设备架构（CUDA, compute unified device architecture）实现，通过规避多精度算术加速底层运算。此外，文献[12-15]也是基于 GPU 的加速器实现研究，但其加速效果并未达到预期，仅呈现出 2~4 个数量级的提升。这种局限性可能源于同态加密中涉及的大量非元素级数据流操作与 GPU 现有架构之间的不匹配，以及 GPU 片上存储容量有限导致的频繁数据迁移需求[16]。

3) 基于专用集成电路（ASIC, application-specific integrated circuit）的全同态加密算法加速器。为追求 FHE 技术计算效率的极致性，一些学者尝试为同态加密设计专用电路，也就是“芯片”。Samardzic 等[16]提出了一种名为“F1”的全同态加密硬件加速器。该加速器基于 ASIC 设计，实现了对软件 3~4 个数量级的加速。Samardzic 等[17]设计了一款名为 CraterLake 的 FHE 加速器，能够高效地处理无限乘法深度的 FHE 程序，其性能优于以往最好的 FHE 加速器约 11.2 倍。这些成果的加速效果显著，但相应的开发成本也是最高的。

综上所述，当前主流的研究趋势主要集中在 FPGA、GPU 和 ASIC 平台上。在具体应用中，根据实际需求对这些硬件平台的特性进行全面评估至关重要，应确保本文算法实现能够达到最佳的性能和效率。

2 基础知识

2.1 格密码

格密码是一种基于格论构建安全公钥密码系统的密码学方法，它由向量集合构成空间，具有一定的几何和代数性质[18]。

格密码的核心构建依赖于基于格的数学难题，其中与同态领域密切相关的是 LWE 问题和 RLWE 问题。在 LWE 问题中，Regev[19]将其量子规约到了格上的标准困难问题，该问题具有搜索问题和判定问题 2 个版本。这些问题旨在格中寻找特定向量或求解线性方程组，具有困难性。然而，基于 LWE 问题构建密码算法的效率问题成为使用的限制因素，例如，在 LWE 框架下生成一个伪随机数的代价是非常高昂的。为了克服此类问题，Lyubashevsky 等[20]通过引入多项式工具，就是目前人们所熟知的 RLWE。基于 RLWE 的加密方案相比于基于 LWE 的加密方案在效率、安全性和扩展性方面均有明显的改善。目前，许多实用的格密码方案都是基于 RLWE 问题构建的，如 BFV、BGV（Brakerski-Gentry-Vaikuntanathan）、CKKS（Cheon-Kim-Kim-Song）等。

2.2 BFV 算法

BFV 方案是由 Fan 等[21]提出的。该方案能通过 Bootstrapping 操作实现真正的“全同态”，但其时间成本高，且在资源有限的 FPGA 上更具困难。为兼顾效率和资源问题，本文采用的 BFV 为层次型的 RLWE 方案，通过预先设定乘法电路的最大深度，允许在该深度范围内执行同态乘法操作。

BFV 的 RLWE 方案中定义了一个特殊的多项式环 $R=\dfrac{\mathbb{Z}[x]}{x^n+1}$，其中 x^n+1 为多项式模，以确保将计算结果规约到多项式环上。BFV 算法的 5 个主要组成部分如下。

1) 密钥生成算法（BFV-KeyGen）：包括私钥生成算法、公钥生成算法和计算密钥生成算法。

① 私钥生成算法。从多项式空间 R_2 中选取私钥 s，其中将 R_2 定义为系数在 $\{-1,0,1\}$ 中独立且随机均匀选择的多项式集；输出私钥 $\mathrm{sk}=s$。

② 公钥生成算法。输入私钥 $\mathrm{sk}=s$，选取 $\mathrm{pk}_0=[-(as+e)]_q$，$\mathrm{pk}_1=a$，其中 $a\leftarrow R_q$，$e\leftarrow\chi$；输出公钥为

$$\mathrm{pk}=(\mathrm{pk}_0,\mathrm{pk}_1)=\left(\left[-(as+e)\right]_q,a\right) \tag{1}$$

③ 计算密钥生成算法。输入私钥 $\mathrm{sk}=s$；设 $l=\left\lfloor\log_T(q)\right\rfloor$，对于 $i\in[0,l]$，选取 $a_i\leftarrow R_q$，$e_i\leftarrow\chi$，$T=\left\lceil\sqrt{q}\right\rceil$；输出计算密钥为

$$\mathrm{rlk}=\left[\left(\left[-\left(a_is+e_i\right)+T^is^2\right]_q,a_i\right):i\in[0,l]\right] \tag{2}$$

2) 同态加密算法（BFV-Enc）：输入明文 $m\in R_t$，公钥 pk；记 $p_0=\mathrm{pk}[0]$，$p_1=\mathrm{pk}[1]$，$\Delta=\left\lfloor\dfrac{q}{t}\right\rfloor$，选取 $u\leftarrow R_2$，$e_1,e_2\leftarrow\chi$；输出密文为

$$\mathrm{ct}=\left(\left[p_0u+e_1+\Delta m\right]_q,\left[p_1u+e_2\right]_q\right) \tag{3}$$

3) 同态解密算法（BFV-Dec）：输入密文ct，私钥sk = s；令c_0 = ct[0]，c_1 = ct[1]；恢复明文为

$$m' = \left\lfloor \frac{t\left[c_0 + c_1 s\right]_q}{q} \right\rceil_t \tag{4}$$

4) 同态加法（BFV-Add）：输入密文$\mathrm{ct}_1 = (\mathrm{ct}_1[0], \mathrm{ct}_1[1])$，$\mathrm{ct}_2 = (\mathrm{ct}_2[0], \mathrm{ct}_2[1])$；输出同态加法结果为

$$\left(\left[\mathrm{ct}_1[0] + \mathrm{ct}_2[0]\right]_q, \left[\mathrm{ct}_1[1] + \mathrm{ct}_2[1]\right]_q\right) \tag{5}$$

5) 同态乘法（BFV-Mult）：输入密文$\mathrm{ct}_1 = (\mathrm{ct}_1[0], \mathrm{ct}_1[1])$，$\mathrm{ct}_2 = (\mathrm{ct}_2[0], \mathrm{ct}_2[1])$，计算密钥rlk = (rlk[0], rlk(1))；计算密文的基础乘法，输出3个密文分别为

$$c_0 = \left\lfloor \frac{t\left(\mathrm{ct}_1[0]\mathrm{ct}_2[0]\right)}{q} \right\rceil_q \tag{6}$$

$$c_1 = \left\lfloor \frac{t\left(\mathrm{ct}_1[0]\mathrm{ct}_2[1] + \mathrm{ct}_1[1]\mathrm{ct}_2[0]\right)}{q} \right\rceil_q \tag{7}$$

$$c_2 = \left\lfloor \frac{t\left(\mathrm{ct}_1[1]\mathrm{ct}_2[1]\right)}{q} \right\rceil_q \tag{8}$$

然后，进行重线性化，将c_2改为以T为基的形式，即 $c_2 = \sum_{i=0}^{l} {c_2}^{(i)} T^i c_2^{(i)} \bmod q$，其中 $l = \lfloor \log_T q \rfloor$，${c_2}^{(i)} \in R_T$；计算同态乘法。

$$c_0' = \left[c_0 + \sum_{i=0}^{l} \mathrm{rlk}[i][0] {c_2}^{(i)}\right]_q \tag{9}$$

$$c_1' = \left[c_1 + \sum_{i=0}^{l} \mathrm{rlk}[i][1] {c_2}^{(i)}\right]_q \tag{10}$$

最后，输出同态乘法结果$\left(c_0', c_1'\right)$。

通过上述算法，可以发现同态乘法是BFV整个方案中计算成本最高的环节。执行一次同态乘法需要执行4次多项式乘法和1次多项式加法，且同态乘法本质上又包含同态加法。因此，本文的重点在于同态乘法的加速。

2.3　NTT数论变换

同态加密算法的核心是在维持密文状态的同时能够对密文执行计算，这些计算在数学上表现为多项式操作。在多项式操作过程中，如果采用系数表示法进行多项式乘法，将会显著降低在密文空间中的运算效率。这种问题在处理大规模数据集或频繁进行多项式乘法时尤为突出。

快速傅里叶变换（FFT, fast Fourier transformation）是一种提高多项式乘法运算效率的方法。然而，FFT在涉及复数运算时不可避免地会带来精度损失[22]，这种损失会给计算结果带来误差。在同态加密领域，这种误差是希望能避免的，因为它会导致解密失败。NTT算法作为一种替代方法，在$\mathbb{Z}_P$上进行能有效地规避此问题。NTT算法与离散傅里叶变换（DFT, discrete Fourier transform）有类似之处，DFT及其逆变换分别为

$$X_k = \sum_{n=0}^{N-1} x_n \mathrm{e}^{-\frac{2\pi \mathrm{i}}{N} nk}, \quad k = 0,1,\cdots,N-1 \tag{11}$$

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \mathrm{e}^{\frac{2\pi \mathrm{i}}{N} kn}, \quad n = 0,1,\cdots,N-1 \tag{12}$$

FFT是一种实现DFT的高效算法，它的基本思想是分治。FFT在于单位根存有以下3个关键性质。

1) 周期性：n个单位根互不相同，且幂具有周期性。

2) 消去引理：对于任意的整数$n \geqslant 0, k \geqslant 0$及$d > 0$，则有

$$w_{dn}^{dk} = \mathrm{e}^{\frac{2dk\pi}{dn}\mathrm{i}} = \mathrm{e}^{\frac{2k\pi}{n}\mathrm{i}} = w_n^k \tag{13}$$

3) 折半引理：如果n为偶数，则n个n次单位复根的平方将构成$\frac{n}{2}$个不同的$\frac{n}{2}$次单位复根的集合。

$$\left(w_n^{\frac{k+n}{2}}\right)^2 = w_n^{2k+n} = w_n^{2k} = \left(w_n^k\right)^2 = w_{\frac{n}{2}}^k \tag{14}$$

定义g为素数模p的一个原根，令$g_n = g^{\frac{p-1}{n}}$，则称g_n为模p下的n次主单位根。$g^0, g^1, \cdots, g^{p-1}$和$0,2,\cdots,p-1$形成一个一一对应的简化剩余系，有

$$g_n^n \equiv 1 (\bmod p) \tag{15}$$

$$g_n^{\frac{n}{2}} \equiv -1 (\bmod p) \tag{16}$$

$$\left(g_n^{\frac{k+n}{2}}\right)^2 = g_n^{2k+n} \equiv g_{\frac{n}{2}}^k (\bmod p) \tag{17}$$

这说明原根具有与单位根相同的3个性质。因此，把复平面的单位根映射到有限域的原根上，只需将w_n替换为g_n，得到的数论变换及其逆变换（INTT, inverse number theoretic transform）分别为

$$X_k = \sum_{n=0}^{N-1} x_n g^{nk} (\mathrm{mod}\, p), \quad k = 0,1,\cdots,N-1 \tag{18}$$

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k g^{-kn} (\mathrm{mod}\, p), \quad n = 0,1,\cdots N-1 \tag{19}$$

2.4 RNS剩余数系统

剩余数系统（RNS, residue number system）是一种非传统的数制，它将整数表示为一组互素模数下的余数[23]。这种表示方式的独特之处是加法运算和乘法运算可以在各个模数下独立执行，大幅提高了并行计算的能力。

RNS需要选择k个互不相同且互素的正整数$m_1,m_2,\cdots,m_k$，它们的乘积确定了剩余数系统能够表示的数值范围。任意整数X在RNS下的表示形式为一组余数$(x_1,x_2,\cdots,x_k)$，其中，每个余数x_i都是整数X除以对应模数m_i后的余数，即$x_i \equiv X \bmod m_i$。

下面以多项式乘法为例，2个多项式$A(x)$和$B(x)$分别为$A(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$和$B(x) = b_0 + b_1 x + \cdots + b_{n-1}x^{n-1}$，则$C(x) = A(x)B(x)$的每个系数都可以在各模数$m_i$下独立地进行计算，然后利用中国余数定理（CRT, Chinese remainder theorem）将各模数下的计算结果合并，得到最终的系数。

CRT是一种能够将RNS中的余数表示形式转换回传统的整数表示形式的方法[24]。假设$M = \prod_{i=1}^{k} m_i$，且$M_i = \frac{M}{m_i}$，则可以通过式(20)从余数$(x_1,x_2,\cdots,x_k)$中恢复出原始整数X。

$$X = \sum_{i=1}^{k} x_i M_i M_i^{-1} \bmod M \tag{20}$$

其中，M_i^{-1}表示M_i在模数m_i下的逆元。

3 算法硬件电路架构设计

3.1 系统总体架构方案

本节设计了基于Zynq平台的BFV全同态加密算法高效实现的系统总体架构。将执行加密、密钥生成和解密的电脑端视为客户端（PC端），将执行同态运算的Zynq视为服务器，执行计算复杂度最高的操作——同态运算，两者之间采用TCP。本文把Zynq的FPGA部分称为PL端，ARM部分称为PS端。BFV全同态加密算法高效实现的系统架构如图1所示。

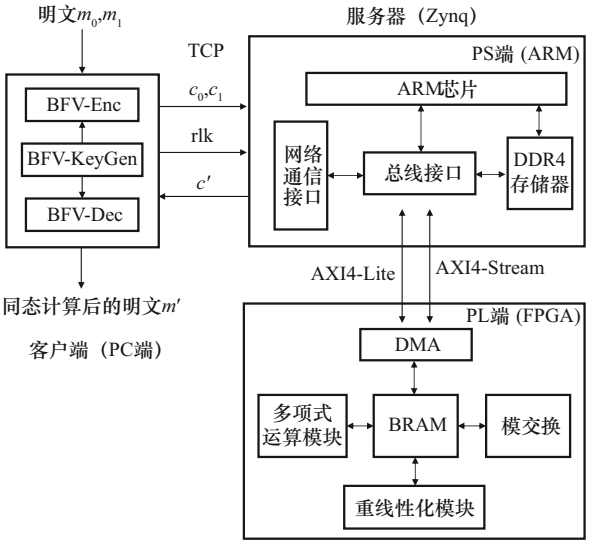

图1 BFV全同态加密算法高效实现的系统架构

在系统架构中，PS端担当“CPU”角色，负责接收和存储来自客户端（PC端）生成的计算密钥和密文数据；生成指令并通过AXI4-Lite控制PL端进行同态运算；PS端与PL端的密文和密钥数据通过AXI4-Stream协议传输。为提升传输效率，PS端的DDR4存储器与PL端之间采用直接存储器访问（DMA, direct memory access）方式进行通信。

3.2 参数设计

在开始硬件设计之前，首先要确定实现全同态加密方案所需的参数。本文在参数设计时借鉴了文献[25-28]的相关研究，在算法安全性和性能之间做出了合理的均衡。本文采用的BFV全同态加密算法参数设计如表1所示。

表1 BFV全同态加密算法参数设计

算法指标	参数值
模多项式环的大小n	4 096
模交换前的密文多项式系数模q/bit	180
模交换后的密文多项式系数模Q/bit	390
电路深度	4
安全参数/bit	80

在表1中，模多项式环的阶数为4 096，模交换前的密文多项式系数模q为180 bit。通常情况下，密文模数q越大，表示密文空间越大，其所能容纳的噪声越多，意味着系统能够执行更多次同态计算。模交换后的密文多项式系数模数称为Q，这是一个模数提升的操作，因此Q远大于密文模数q。

同态加密本质上包含噪声，这种噪声不仅保障了算法的安全性，也限制了执行同态乘法运算的次数。若噪声累计超过某个阈值，将导致解密失败。其中，所能允许的最大同态乘法次数，即“电路深度”取决于同态加密算法参数的设置。此系统中电路深度设置为4，安全参数为80 bit。

3.3　软硬件数据存储和传输方案设计

本节设计了异构计算平台下软硬件之间的数据传输方案，数据存储与传输过程如图2所示。为避免同态运算过程中对存储空间的重复访问，本文将所有密文数据存储在Zynq的PL端的BRAM，有效降低了时间成本。相比于密文数据，计算密钥所占的存储空间大约为其3倍。在同态加法运算中，这些计算密钥不会被使用，而在每次同态乘法运算中它们仅被使用一次。因此，选择将计算密钥存放在Zynq的PS端的DDR4。

本文方案采用AXI4（advanced extensible interface 4）协议中的AXI4-Lite和AXI4-Stream用于SoC内部的数据交换。AXI4-Lite用于简单的寄存器访问，如配置DMA寄存器和分发来自ARM的操作指令，它不支持突发传输。考虑到密文多项式系数的存储特性不需要复杂的地址控制，因此，这些系数被视为流数据并通过AXI4-Stream进行突发传输。

在ARM与FPGA之间，采用DMA机制传输密文和计算密钥的多项式系数。在传输启动前，必须使用AXI4-Lite协议配置DMA寄存器。之后DMA通过AXI4-Stream协议独立完成数据传输。此外，本文在ARM端基于轻量级IP协议栈Lwip设计了网络应用程序，为客户端与服务器之间提供网络通信服务。传输的数据包括密文数据c和计算密钥rlk。

4　多项式乘法运算方案设计

4.1　NTT算法

在多项式乘法运算中采用的NTT迭代算法[22]如算法1所示。

算法1　NTT迭代算法

输入　$n-1$次多项式$a(x)\in\mathbb{Z}_q[x]$，n次单位根$w_n\in\mathbb{Z}_q$

输出　多项式$A(x)\in\mathbb{Z}_q[x]=\mathrm{NTT}(a)$

1) $A\leftarrow \mathrm{BitReverse}(a)$

2) for m=2 to n by m=2m do

3)　$w_m\leftarrow w_n^{\frac{n}{m}}$

4)　$w\leftarrow 1$

5)　for j=0 to $\frac{m}{2}-1$ do

6)　　for k=0 to $n-1$ by m do

7)　　$t\leftarrow wA\left[k+j+\frac{m}{2}\right]$

8)　　$u\leftarrow A[k+j]$

9)　　$A[k+j]\leftarrow u+t$

10)　　$A\left[k+j+\frac{m}{2}\right]\leftarrow u-t$

11)　　end for

12)　　$w\leftarrow ww_m$

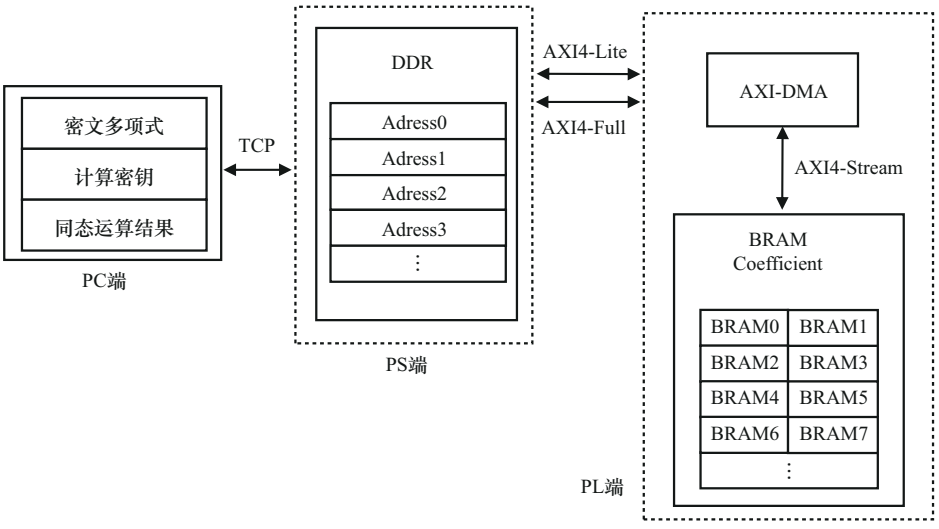

图2　数据存储与传输过程

13) end for

14) end for

该迭代算法首先需要对多项式进行预处理操作BitReverse(a)；然后进入迭代主循环，该循环从m=2开始翻倍，直到m=n；接着进入内部的蝶形循环Butterfly loop，主要由常数模乘、模加和模减运算组成。对于每个j值会对k值进行遍历，线性组合2个系数分别为$A[k+j] \leftarrow u+t$和$A\left[k+j+\frac{m}{2}\right] \leftarrow u-t$。其中，$t$和$u$为中间变量，用于存储当前正在进行蝶形循环的2个系数值；最后更新w的值，令$w=ww_n$。由于NTT的逆变换与正变换迭代过程类似，这里不再赘述。

由此可见，在使用NTT来计算多项式乘法时，实际上是用多项式的点值表示法代替系数表示法，利用点值的乘积从而简化多项式乘法中系数相乘的过程。但是，NTT要求选择足够的点值来确保乘法后的多项式可以被正确地表示和重构。具体来说，需要首先对原始多项式进行“补零”，然后进行2n点的NTT。

对于2个多项式，根据NTT算法，可以使用式(23)计算。

$$A(x)=a_{n-1}x^{n-1}+\cdots+a_1x+a_0 \tag{21}$$

$$B(x)=b_{n-1}x^{n-1}+\cdots+b_1x+b_0 \tag{22}$$

$$C(x)=\mathrm{INTT}_{2n}[\mathrm{NTT}_{2n}(A(x))\odot\mathrm{NTT}_{2n}(B(x))] \tag{23}$$

其中，$\odot$表示对应点值的乘积。

4.2 结合负包裹卷积的NTT优化

BFV全同态加密方案中的多项式乘法是建立在多项式环$\mathbb{Z}[x]$上的，此过程需要对乘法结果$C(x)$模多项式$f(x)$，考虑到BFV方案中模多项式$f(x)$为“x^n+1”的特殊形式，可借此采用负包裹卷积[7]的方法优化多项式环上的NTT算法。这样可以避免在执行NTT前对多项式$A(x)$和$B(x)$补零，以及对最后结果$C(x)$模多项式x^n+1的操作[27]，从而简化硬件电路设计。负包裹卷积的定义如下。

记ψ是$\mathbb{Z}_q$上的2n次单位原根，即$\psi^{2n}\equiv 1 \bmod q$，且对于所有$i<2n$均有$\psi^i \neq 1 \bmod q$。则有

$$A=(\psi a_0,\psi^3 a_1,\cdots,\psi^{2n-1}a_{n-1}) \tag{24}$$

$$B=(\psi b_0,\psi^3 b_1,\cdots,\psi^{2n-1}b_{n-1}) \tag{25}$$

记A和B的负包裹卷积为

$$C=(\psi^{-1},\psi^{-3},\cdots,\psi^{-(2n-1)})\odot\mathrm{INTT}_n[\mathrm{NTT}_n(A)\odot\mathrm{NTT}_n(B)] \tag{26}$$

结合负包裹卷积的NTT优化算法计算多项式乘法如算法2所示。

算法2 结合负包裹卷积的NTT优化算法

输入 $A(x),B(x)\in\dfrac{\mathbb{Z}_q[x]}{(x^n+1)}$，2$n$-th root of unity $\psi\in\mathbb{Z}_q$

输出 $C(x)=A(x)B(x)\in\dfrac{\mathbb{Z}[x]}{x^n+1}$,

$A'(x)\leftarrow \mathrm{NTT}_n[A(x)\odot(\psi,\psi^3,\cdots,\psi^{2n-1})]$,

$B'(x)\leftarrow \mathrm{NTT}_n[B(x)\odot(\psi,\psi^3,\cdots,\psi^{2n-1})]$,

$C'(x)\leftarrow A'(x)\odot B'(x)$,

$C(x)\leftarrow \mathrm{INTT}_n[C'(x)\odot(\psi^{-1},\psi^{-3},\cdots,\psi^{-(2n-1)})]$

在多项式乘法模块的硬件电路设计中，可以对ψ相关的固定参数$(\psi,\psi^3,\cdots,\psi^{2n-1})$和$(\psi^{-1},\psi^{-3},\cdots,\psi^{-(2n-1)})$进行预计算，以牺牲存储空间为代价换取算法执行效率的提升。

4.3 基于NTT并行计算加速方案设计

在RNS下，同态乘法的本质依旧是基于各个余数基的多项式系数之间的加法、减法和乘法。本节为同态乘法设计通用的剩余多项式计算单元，并通过实例化多个单元，实现NTT并行计算加速。

本文所采用的RNS剩余数系统将密文多项式系数模q（180 bit）分解为6组30 bit的互素小模数q_0~q_6。对于模交换过程中所需的更大模数Q（390 bit），以同样的30 bit互素小模数进行分解，需要13组互素小模数q_0~q_{12}。但这两步模数交换并非同时进行。因此，本文采取共享资源的策略，模数q和Q共用6个30 bit的互素小模数q_0~q_6，而为模数Q额外设置7个30 bit的互素小模数q_7~q_{12}。例如，第一个剩余多项式计算单元负责专门处理以q_0和q_6为模数的运算，第二个计算单元处理以q_1和q_7为模数的运算，依次类推，第七个计算单元只处理以q_{12}为模数的运算，该分配方式如图3所示。该策略可以有效避免硬件资源的浪费。

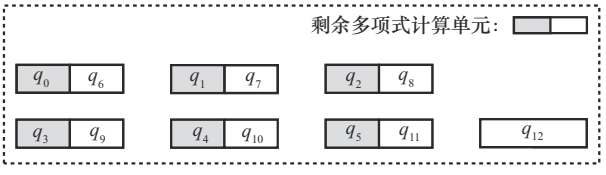

图3 剩余多项式计算单元分配方式

在确定剩余多项式计算单元的数量后，需要对每个计算单元分配适宜的存储资源。在FPGA端采用BRAM作为密文数据存储空间。由于FPGA中每块BRAM36K的地址空间都是连续的，可将密文多项式系数以数组的形式依次存入。又因为每块BRAM36K拥有36 KB的存储容量，由此设定每个多项式的系数位宽为30 bit，则每块BRAM36K最多可以存储1 024个30 bit的系数。存储一个4 096阶的密文多项式需要使用4块BRAM36K。

此外，每块BRAM36K均为双口BRAM，支持一个写端口和一个读端口，所以一个包含4块BRAM36K的密文多项式在每个时钟周期内可支持4个系数的读操作和写操作。由NTT迭代算法可知，执行一次NTT迭代算法中的Butterfly loop，需要2个多项式系数作为输入，并输出2个对应的多项式系数。因此，本文方案可以为每个剩余多项式计算单元设计2个NTT蝶形运算核心，这样在每个时钟周期内可以执行2次蝶形运算，以充分利用4块BRAM36K的读写能力。双NTT蝶形运算核心架构设计如图4所示。

尽管上述双NTT蝶形运算核心架构的设计实现了对BRAM资源的合理分配，但仍存在以下2个问题：一是在单个时钟周期内，2个NTT蝶形运算核心无法同时对同一块BRAM进行读取或写入；二是一个NTT蝶形运算核心不能从单个BRAM中同时读取2个系数。根据NTT算法，读取2个系数需耗费2个时钟周期，而本文想要在单个周期内完成。经过分析，这些问题的主要原因在于内存访问机制的冲突。文献[6,8,10,29]均有对内存访问冲突的研究，本文在蝶形运算核心与BRAM交互访问中采用文献[8,10]的做法。

图5展示了双NTT蝶形运算核心内存访问BRAM的优化方法。图5的最左侧部分是针对算法中$A[k+j]$和$A\left[k+j+\frac{m}{2}\right]$这2个系数不能同时读取的问题。该优化方法通过将两块位宽为30 bit、深度为1 024的BRAM36K组成一个“新的BRAM”（位宽为60 bit、深度为1 024），如BRAM1与BRAM3、BRAM2与BRAM4。每两块BRAM具有相同的地址和读写使能信号，这种数据存储方式可以解决不能同时访问2个系数的问题。然后，这2个“新的BRAM”分别由2个NTT计算核心控制（图中虚线方框），通过交替读写的方式避免访问冲突。其中，R_i和$R_i'(i=0,1,\cdots,1\,023)$分别表示第一个和第二个NTT蝶形运算核心里蝶形循环模块Butterfly loop的系数“读请求”，这种方法能够有效规避2个核心同时对BRAM进行读写的现象。访问模式与蝶形循环算法外层的参数m相关，此处以m=2 048为例。

4.4　基于滑动窗口模约减的优化

对于固定模数的求模运算，一种有效的做法是滑动窗口算法。根据窗口的可变性，它主要分为定长滑动窗口和可变滑动窗口。文献[8]在求模运算中采用了定长滑动窗口，虽然文献[10]采用了Montgomery约减对系数的规约起到一些速度上的优化，但却增加了电路面积和逻辑设计的复杂性。因此，本文决定采用性能相较于定长滑动窗口更优的可变滑动窗口。

可变滑动窗口使零窗口数量尽可能大，从而减少乘法阶段所需的乘法次数，其特殊点在于窗口提取的过程，提取方法如下。

记w_i为第i个窗口的值，$|w_i|$为每次提取的窗口长度，z表示窗口w后0的个数，则

$$E=((w_1 2^{|w_2|+z_1}+w_2)2^{|w_3|+z_2}+\cdots+w_t)2^{|w_{t+1}|+z_t} \tag{27}$$

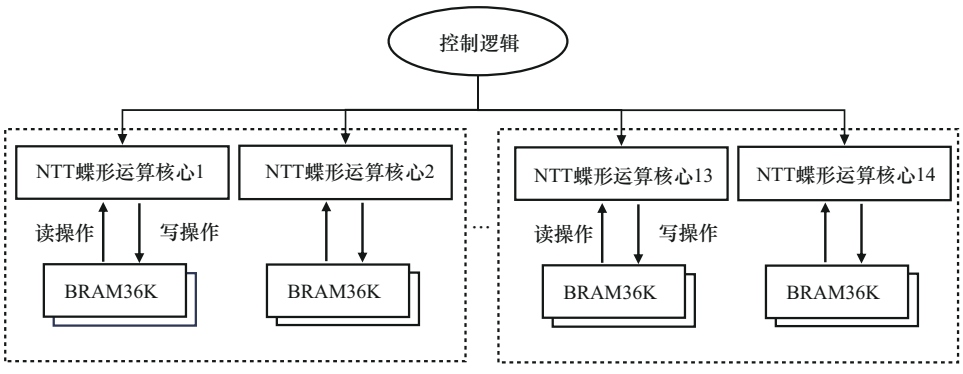

图4　双NTT蝶形运算核心架构设计

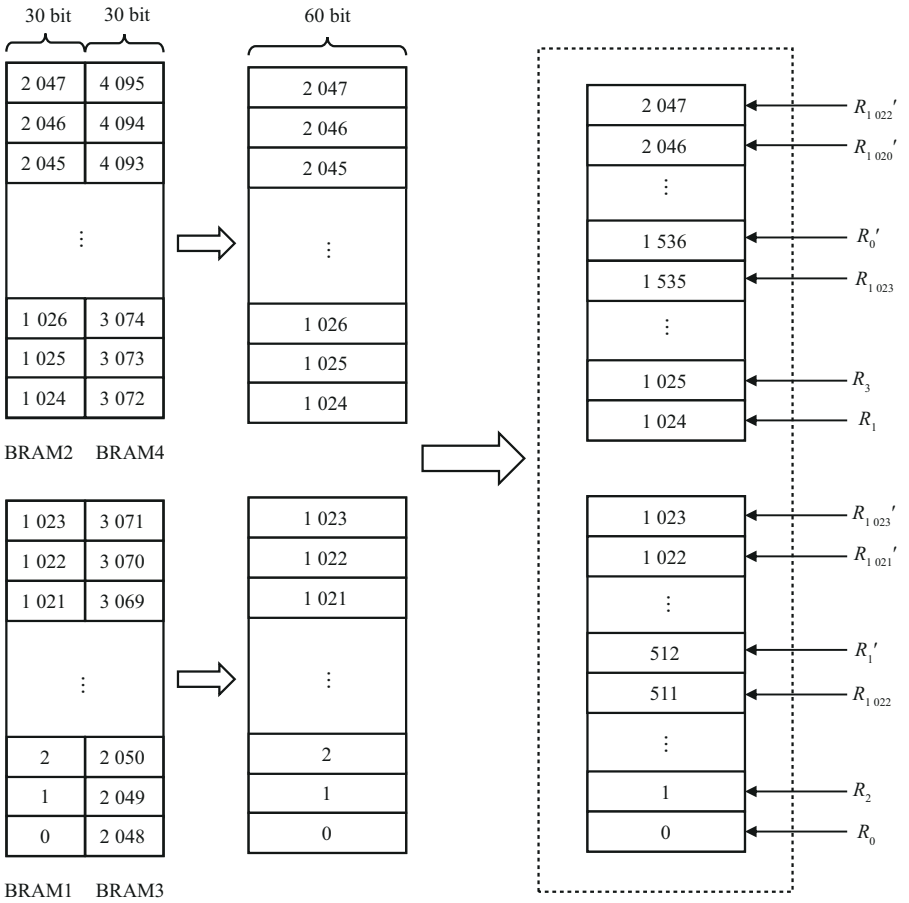

图5　双NTT蝶形运算核心内存访问BRAM优化方法（m=2 048）

5　模交换方案设计

在同态乘法中，涉及2个关键模交换的步骤：$\mathrm{Lift}(q \to Q)$和$\mathrm{Scale}(Q \to q)$。$\mathrm{Lift}(q \to Q)$步骤发生在多项式乘法之前，它将4个以模q表示的密文多项式转换到更大的模Q空间中；$\mathrm{Scale}(Q \to q)$步骤则发生在多项式乘法之后，将结果从模Q降回到模q。

记a为在环R_q上的多项式$f(x)$的一个系数，a_i为a的RNS表示，且该RNS的余数基为$q_i(i=0,1,\cdots,12)$。

5.1　基于RNS的Lift($q \to Q$)功能实现

$\mathrm{Lift}(q \to Q)$的计算步骤如下。首先，将RNS系统中的以余数基$q_0,q_1,q_2,q_3,q_4,q_5,q_6$表示的多项式转换为以余数基为$q$表示的形式。如式(28)所示。

$$a = \sum_{i=0}^{5} a_i \tilde{q}_i q_i^* \tag{28}$$

其中，$q_1^* = \dfrac{q}{q_i}$，$\tilde{q}_i$为q_i^*的模逆。

然后，将余数基为q的多项式转换为另外7个余数基q_6~q_{12}，最终得到以13个余数基q_0~q_{12}的RNS多项式表示。计算步骤如式(29)所示。

$$a_i = a \bmod q_i, i = 6,\cdots,12 \tag{29}$$

Halevi等[9]提出基于近似中国余数定理的优化方法，希望通过一些快速运算取代式(29)中耗时较高的求模操作。下面以余数基中的一个基q_6为例介绍求解过程，如式(30)所示。其他余数基q_i的求解过程与此类似。

$$a_6 = \sum_{i=0}^{5} (a_i \tilde{q}_i \bmod q_i) q_i^* - v' q_6 \tag{30}$$

其中，$a_i,q_i,\tilde{q}_i$均为二进制表示位宽为30 bit的整数；v'经化简后得到$v' = \left\lfloor \sum_{i=0}^{5} \left(\dfrac{a_i \tilde{q}_i \bmod q_i}{q_i} \right) \right\rceil$。

为契合FPGA并行计算的特性，在$\mathrm{Lift}(q \to Q)$硬件实现上，本文同样采用该优化方法，即用多个小整数的乘积来取代大整数的模运算。$\mathrm{Lift}(q \to Q)$由5个相应的硬件模块通过“流水线”的方式实现，如图6所示。

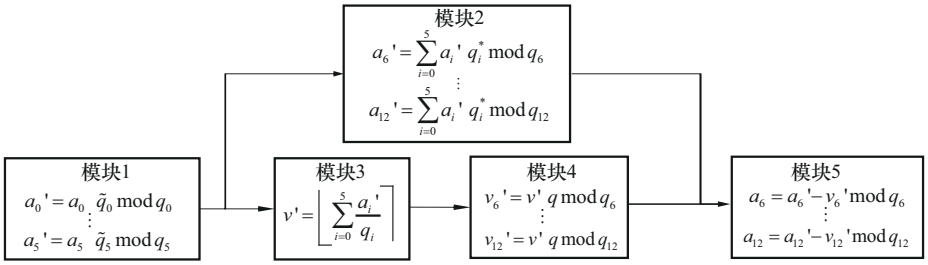

图6　Lift模块架构

5.2　基于RNS的Scale($Q \to q$)功能实现

为避免大整数取模运算，Scale($Q \to q$)实现方法与5.1节中Lift($q \to Q$)方法保持一致，如图7所示。

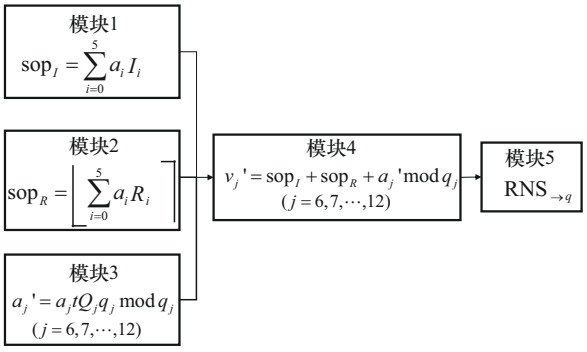

图7　Scale模块架构

Scale($Q \to q$)将余数基$q_0,q_1,\cdots q_{12}$转换为余数基$q_0,q_1,\cdots,q_5$，实现多项式系数从模数Q到模数q的转换。这一转换过程并非一步到位，而是先将模数Q转换成模数p的多项式形式，再利用Lift($q \to Q$)对模数p进行模数提升进而得到模数q的多项式。其中，模块1和模块2通过计算a_i与I_i、R_i乘积之和来代替大整数求模运算；模块3为模块4提供中间计算值a_j'；模块4输出模数p的7个余数基下的RNS表示并将结果提供给模块5；最终模块5再次调用Lift($q \to Q$)电路实现RNS模数p到模数q的转换。

6　实验测试

实验采用的硬件电路平台为Zynq UltraScale+ MPSoC ZCU102，该开发板搭载了Xilinx公司的XCZU9EG-2FFVB1156型号芯片。

此外，使用Vivado 2020.1和Vitis 2020.1作为开发平台。FPGA端基于Verilog语言，结合部分IP核（block memory generator、distributed memory generator、multiplier）实现复杂的功能模块；ARM端采用C语言；客户端软件层采用C++语言编写基于TCP通信协议的同态加密程序。

6.1　正确性测试

6.1.1　内存资源管理测试

本节对同态运算的内存资源管理模块进行测试。该模块由7个独立的Memory Block子模块组成，每个Memory Block都对应一个剩余多项式计算单元，其配置有9个深度为2 048、位宽为60 bit的存储单元。每个存储单元又进一步采用真双口RAM实现，以支撑高效的并行数据访问与处理。

图8展示了该内存资源管理模块读写功能的仿真测试，本文采用了4.3节中的BRAM访问优化策略以避免2个蝶形运算核心间的管理矛盾，并为BRAM的各地址依次写入数据。因限于篇幅，本节仅展示Memory Block 0（MB0）的测试结果，其他模块与此类似。

6.1.2　NTT/INTT功能测试

NTT和INTT算法是多项式乘法运算的关键组成部分。在基于NTT的并行计算加速中，本文为每个剩余多项式计算单元设计2个蝶形运算核心。功能仿真测试结果如图9所示。

测试结果表明，当INTT执行完毕后，done信号会被拉高一个周期，随后系统将会为密文多项式的系数分配存储资源，这会耗费一定数量的时钟周期。信号ntt_result和intt_result分别为NTT和INTT的计算结果，信号ntt_expected和intt_expected均为在该参数设置下的预期计算结果，它们不依赖Verilog程序，而是在Python环境下编译产生并存储在txt文件中，然后在testbench仿真文件中通过$readmemh指令进行读取。这2个信号用于对比验证Verilog程序中NTT/INTT算法的正确性，实验结果对比验证无误。

6.2　计算效率测试

本文设计的同态乘法硬件电路由多个子模块共

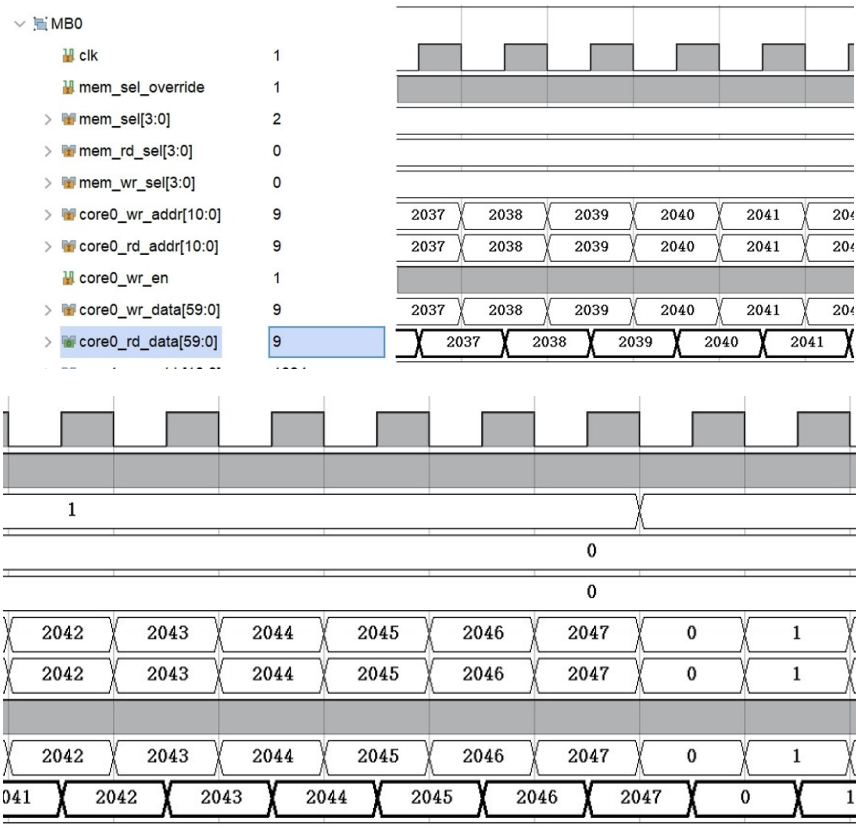

图8 Memory Block 0内存资源管理测试

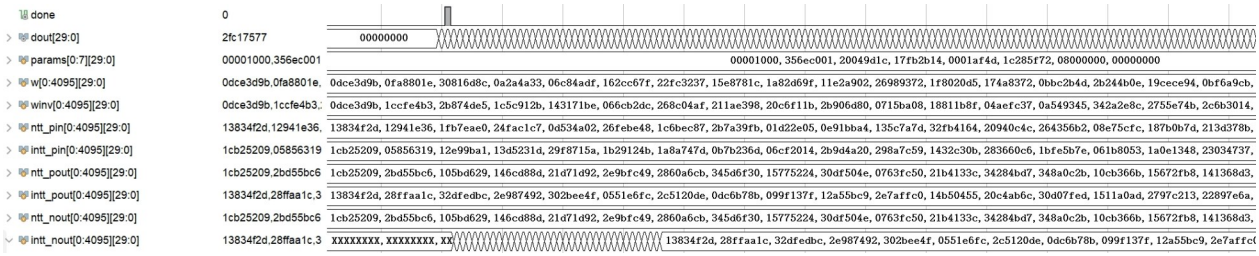

图9 基于双蝶形运算核心的NTT/INTT测试

同实现，包括NTT、INTT、多项式系数乘法、多项式系数加法、Lift$(q \to Q)$和Scale$(Q \to q)$。顶层电路已集成所有子模块，并采用额外的电路通过状态机实现对同态运算的流程控制。

系统根据接收到的指令调用相应子模块执行功能，从而实现整个同态乘法的过程。经测试，在200 MHz工作频率下执行一次同态乘法所需子模块的执行次数和耗费时间如表2所示。

密文和密钥在PS端与PL端之间的传输采用PL端的AXI-DMA软核实现，本文在文献[8]和文献[10]的基础上，对DMA的传输时序进行了优化，保证了数据流传输的连续性。

表2 执行一次同态乘法所需子模块的执行次数和耗费时间

子模块	执行次数/次	耗费时间/ms
NTT	14	1.358
INTT	8	0.872
多项式系数乘法	20	0.280
多项式系数加法	26	0.338
Lift$(q \to Q)$	4	0.332
Scale$(Q \to q)$	3	0.255
计算时间（不包括数据传输时间）	—	3.435

在数据流方面，密文和计算密钥均由客户端产生并通过TCP通信协议发送至服务器Zynq，被存储在PS端的DDR4中；而这一步骤在文献[8]和文献[10]

中并没有体现。经实际测试，计算密钥的传输时间约为 1.47 ms，完整执行一次同态乘法耗时为 4.905 ms。此外，还需考虑发送和接收密文多项式的时间成本，两者分别为0.581 ms和0.293 ms。综合以上各项时间开销，系统完成一次同态乘法总耗时为5.779 ms。表3分析了同态运算时间消耗。

表3　　同态运算时间消耗分析

功能	时间/ms
同态加法	0.024
同态乘法	4.905
发送密文	0.581
接收密文	0.293

与同态乘法相比，同态加法在实现上要容易许多，仅需调用2次多项式系数模加运算。在不考虑密文数据传输的情况下，实际执行一次同态加法的耗时仅为0.024 ms，执行一次同态乘法的耗时为4.905 ms。

在计算机平台（Intel Core i7-10870H CPU @ 2.20 GHz）上运行与本文方案参数相同的SEAL库和OpenFHE库的BFV实例，并与近年来时钟频率和参数接近的相关研究进行对比。结果如表4所示。

表4　　同态运算效率对比结果

工作	方案	同态加法/ms	同态乘法/ms
SEAL	BFV	0.111	25.651
OpenFHE	BFV	0.163	17.074
文献[8]	BFV	0.026	5
文献[26]	BFV	—	11.9
文献[27]	YASHE	0.19	6.75
本文方案	BFV	0.024	5.779

测试结果表明，本文方案相比于SEAL库和OpenFHE库在相同参数下的同态运算效率有显著提升。在不考虑密文传输时间的情况下，同态加法的效率提升分别达到4.63倍和6.79倍；在考虑密文传输时间的情况下，同态乘法的效率提升分别为4.43倍和2.95倍。实验结果表明，本文方案在同态乘法运算效率上略低于文献[8]，主要是因为文献[8]的密文直接存储在DDR中，并未考虑客户端与服务器之间的通信开销。此外，本文方案在取模运算中采用了可变滑动窗口法，使取模操作更加灵活且适当增加了流水线深度。通过在组合逻辑路径中插入多级寄存器来缩短关键路径，优化了时序并提升了电路的稳定性。

6.3　硬件资源利用分析

在完成功能和效率测试后，对硬件资源的利用情况进行分析，结果如表5所示。

表5　　硬件资源利用情况

资源	使用量	可用量	占比
LUT	56 995	274 080	20.80%
LUTRAM	3 641	144 000	2.50%
FF	34 054	548 160	6.21%
BRAM	407	912	44.63%
DSP	227	2 520	9.01%

从表5中可看出，各种硬件资源的利用情况并不均衡，占比最高的是BRAM，其次为LUT。原因是在计算过程中FPGA采用BRAM作为存储资源，而且NTT及同态乘法算法中的参数均通过预计算的方法提前处理并存入BRAM。实际上，本文是以BRAM资源为代价换取FPGA与DDR4之间较少的数据传输量，从而降低PS端与PL端之间的通信开销。

7　结束语

本文在Zynq平台上完整实现了BFV全同态加密算法。经过测试，在多项式环的大小 n=4 096，密文多项式系数模 q=180 bit且采用6组余数基的RNS配置下，系统执行一次同态加法的平均耗时为0.024 ms；执行一次同态乘法的平均耗时为5.779 ms，其中包括密文的发送与接收时间0.874 ms。与SEAL库和OpenFHE库在相同参数设置下的实现相比，本文方案的同态加法运算效率分别提升4.63倍和6.79倍，同态乘法的运算效率上分别实现了4.43倍和2.95倍的显著提升。未来的工作将对软硬件间的密文数据与密钥数据传输方案进行进一步优化。

参考文献：

[1] BADAWI A A, BATES J, BERGAMASCHI F, et al. OpenFHE: open-source fully homomorphic encryption library[C]//Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic

Cryptography. New York: ACM Press, 2022: 53-63.

[2] 杨亚涛, 赵阳, 张奇林, 等. 基于SEAL库的同态加权电子投票系统[J]. 计算机学报, 2020, 43(4): 711-723.
YANG Y T, ZHAO Y, ZHANG Q L, et al. Weighted electronic voting system with homomorphic encryption based on SEAL[J]. Chinese Journal of Computers, 2020, 43(4): 711-723.

[3] 欧光槟. 基于FPGA的同态加密计算加速硬件设计与实现[D]. 成都: 电子科技大学, 2023.
OU G B. Hardware design and implementation of homomorphic encryption calculation acceleration based on FPGA[D]. Chengdu: University of Electronic Science and Technology of China, 2023.

[4] DIJK M V, GENTRY C, HALEVI S, et al. Fully homomorphic encryption over the integers[C]//Proceedings of the 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2010: 24-43.

[5] GENTRY C, HALEVI S. Implementing gentry's fully-homomorphic encryption scheme[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2011: 129-148.

[6] ROY S S, VERCAUTEREN F, MENTENS N, et al. Compact ring-LWE cryptoprocessor[C]//International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2014: 371-391.

[7] LONGA P, NAEHRIG M. Speeding up the number theoretic transform for faster ideal lattice-based cryptography[C]//International Conference on Cryptology and Network Security. Berlin: Springer, 2016: 124-139.

[8] ROY S S, TURAN F, JARVINEN K, et al. FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data[C]//Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA). Piscataway: IEEE Press, 2019: 387-398.

[9] HALEVI S, POLYAKOV Y, SHOUP V. An improved RNS variant of the BFV homomorphic encryption scheme[C]//Cryptographers' Track at the RSA Conference. Berlin: Springer, 2019: 83-105.

[10] 谢刚. 基于FPGA的BFV同态计算加速器的设计与实现[D]. 哈尔滨: 哈尔滨工业大学, 2021.
XIE G. Design and implementation of BFV homomorphic computing accelerator based on FPGA[D]. Harbin: Harbin Institute of Technology, 2021.

[11] BADAWI A A, POLYAKOV Y, AUNG K M M, et al. Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme[J]. IEEE Transactions on Emerging Topics in Computing, 2021, 9(2): 941-956.

[12] WANG W, HU Y, CHEN L M, et al. Accelerating fully homomorphic encryption using GPU[C]//Proceedings of the 2012 IEEE Conference on High Performance Extreme Computing. Piscataway: IEEE Press, 2012: 1-5.

[13] WANG W, HU Y, CHEN L M, et al. Exploring the feasibility of fully homomorphic encryption[J]. IEEE Transactions on Computers, 2015, 64(3): 698-706.

[14] WANG W, CHEN Z L, HUANG X M. Accelerating leveled fully homomorphic encryption using GPU[C]//Proceedings of the 2014 IEEE International Symposium on Circuits and Systems (ISCAS). Piscataway: IEEE Press, 2014: 2800-2803.

[15] OH Y, PARK S C, NA J C, et al. GPU acceleration of Chinese remainder theorem for fully homomorphic encryption[C]//Proceedings of the 2023 International Conference on Electronics, Information, and Communication (ICEIC). Piscataway: IEEE Press, 2023: 1-4.

[16] SAMARDZIC N, FELDMANN A, KRASTEV A, et al. F1: a fast and programmable accelerator for fully homomorphic encryption[C]//Proceedings of the MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture. New York: ACM Press, 2021: 238-252.

[17] SAMARDZIC N, FELDMANN A, KRASTEV A, et al. CraterLake: a hardware accelerator for efficient unbounded computation on encrypted data[C]//Proceedings of the 49th Annual International Symposium on Computer Architecture. New York: ACM Press, 2022: 173-187.

[18] MICCIANCIO D, REGEV O. Lattice-based cryptography[M]. Berlin: Springer, 2009.

[19] REGEV O. On lattices, learning with errors, random linear codes, and cryptography[J]. Journal of the ACM, 2009, 56(6): 1-40.

[20] LYUBASHEVSKY V, PEIKERT C, REGEV O. On ideal lattices and learning with errors over rings[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2010: 1-23.

[21] FAN J F, VERCAUTEREN F. Somewhat practical fully homomorphic encryption[J]. IACR Cryptology Eprint Archive, 2012, 144: 1-19.

[22] MERT A C, ÖZTÜRK E, SAVAŞ E. Design and implementation of a fast and scalable NTT-based polynomial multiplier architecture[C]//Proceedings of the 2019 22nd Euromicro Conference on Digital System Design (DSD). Piscataway: IEEE Press, 2019: 253-260.

[23] CORMEN T H, LEISERSON C E, RIVEST R L, et al. Introduction to algorithms[M]. Massachusetts: The MIT Press, 2022.

[24] DING C, PEI D, SALOMAA A. Chinese remainder theorem: applications in computing, coding, cryptography[M]. Singapore: World Scientific, 1996.

[25] ROY S S, JÄRVINEN K, VLIEGEN J, et al. HEPCloud: an FPGA-based multicore processor for FV somewhat homomorphic function evaluation[J]. IEEE Transactions on Computers, 2018, 67(11): 1637-1650.

[26] MIGLIORE V, REAL M M, LAPOTRE V, et al. Hardware/software co-design of an accelerator for FV homomorphic encryption scheme using karatsuba algorithm[J]. IEEE Transactions on Computers, 2018, 67(3): 335-347.

[27] PÖPPELMANN T, NAEHRIG M, PUTNAM A, et al. Accelerating homomorphic evaluation on reconfigurable hardware[M]. Berlin: Springer, 2015.

[28] MERT A C, ÖZTÜRK E, SAVAŞ E. Design and implementation of encryption/decryption architectures for BFV homomorphic encryption scheme[J]. IEEE Transactions on Very Large Scale Integration Systems, 2020, 28(2): 353-362.

[29] AYSU A, PATTERSON C, SCHAUMONT P. Low-cost and area-efficient FPGA implementations of lattice-based cryptography[C]//Proceedings of the 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST). Piscataway: IEEE Press, 2013: 81-86.

[作者简介]

杨亚涛（1978– ），男，河南平顶山人，博士，北京电子科技学院教授、博士生导师，西安电子科技大学硕士生导师，主要研究方向为信息安全、全同态密码、抗量子密码、密码协议和算法。

曹景沛（2000– ），男，河南新乡人，西安电子科技大学硕士生，主要研究方向为信息安全、密码协议和算法。

陈亮宇（2002– ），男，江西赣州人，北京电子科技学院硕士生，主要研究方向为通信工程、信息安全。

王伟（1998– ），男，山西运城人，西安电子科技大学硕士生，主要研究方向为信息安全、密码协议和算法。