用于全同态加密的数论变换乘法蝶形运算优化及实现

华斯亮* 张惠国 王书昶

(常熟理工学院 苏州 215500)

摘 要：全同态加密(FHE)可以真正从根本上解决云计算时将数据及其操作委托给第三方时的数据安全问题。针对全同态加密中占较大比例的大整数乘法运算优化需求，该文提出一种数论变换乘法蝶形运算的操作数合并算法，利用取模操作的快速算法，分别可将基16和基32运算单元的操作数减少到43.8%和39.1%。在此基础上，设计并实现了数论变换基32运算单元的硬件设计架构，在SMIC 90 nm工艺下的综合结果显示，电路的最高工作频率为600 MHz，面积1.714 mm^2。实验结果表明，该优化算法提升了数论变换乘法蝶形运算的计算效率。

关键词：全同态加密；大整数乘法；数论变换；蝶形运算

中图分类号：TN918.91; TN492 **文献标识码**：A **文章编号**：1009-5896(2021)05-1381-08

DOI: 10.11999/JEIT200174

Optimization and Implementation of Number Theoretical Transform Multiplier Butterfly Operation for Fully Homomorphic Encryption

HUA Siliang ZHANG Huiguo WANG Shuchang

(*Changshu Institute of Technology*, *Suzhou* 215500, *China*)

Abstract: Fully Homomorphic Encryption (FHE) allows data to be encrypted and out-sourced to commercial cloud environments for processing, while encrypted which diminishes privacy concerns. For the optimization requirements of large integer multiplication operations in fully homomorphic encryption, an operand merge algorithm of a Number Theory Transform (NTT) multiplier butterfly operation unit is proposed. By using a fast algorithm of modulo operation, the operands of the Radix-16 and Radix-32 units are reduced to 43.8% and 39.1%. The hardware architecture of the NTT Radix-32 unit is designed and implemented. The proposed design is synthesized using 90 nm process technology. The results show that the maximum frequency of the circuit is 600 MHz with die area 1.714 mm^2. The results also show that the optimization algorithm improves the computational efficiency of NTT multiplier butterfly operation.

Key words: Fully Homomorphic Encryption (FHE); Large integer multiplier; Number Theoretical Transform (NTT); Butterfly operation

1 引言

云计算为我们提供了一个巨大的信息处理平台，各种信息存储于网络，传输于网络，处理于网络，人们对数据安全和隐私保护的要求越来越高。如何在保护数据安全和用户隐私的前提下完成安全计算，是云计算亟待解决的一个实际问题[1]。同态加密技术使人们可以在加密的数据中直接进行诸如检索、比较等操作，得出正确的结果，而在整个处理过程中无需对数据进行解密。其意义在于，真正从根本上解决将数据及其操作委托给第三方时的保密问题[2]。

全同态加密(Fully Homomorphic Encryption, FHE)可以执行任意次同态加法和同态乘法操作。许多密码学家都将全同态加密思想置于与公钥加密思想比肩的重要地位[3]。有研究报告预测，FHE在基因组学、健康、国家安全和教育等领域具有广阔的潜在应用前景[4]。自2009年Gentry[5]提出第1个FHE方案至今，全同态加密取得了长足的发展，但仍存在诸多的不足，特别是在计算效率、安全性以及全同态加密的应用等方面。对于一个维度仅为2048的格基FHE，需要785006 bit(约10^{235500})的乘法。通过分析FHE算法的密码库中，每个函数的分析结果，计算的大多数执行时间是大整数乘法(包括平方运算)，消耗了FHE总执行时间的53%～

收稿日期：2020-03-17；改回日期：2020-10-21；网络出版：2020-11-19
*通信作者： 华斯亮 huasiliang@cslg.edu.cn
基金项目：江苏省自然科学基金(BK20191027)
Foundation Item: The Natural Science Foundation of Jiangsu Province (BK20191027)

62%[6]。利用专用硬件加速大整数乘法，可以较大程度地提高FHE的计算效率，推进FHE的市场化进程。

大整数乘法除了传统的长乘法，还有Karatsuba算法、Toom-Cook算法和Schönhage–Strassen算法。以上算法的复杂度依次降低，实现复杂度依次增加[7,8]。对于大于$2^{2^{17}}$(约10^{40000})的数字乘法，Schönhage–Strassen算法开始优于Karatsuba和Toom-Cook乘法[9]。这一特性使得Schönhage–Strassen算法适用于FHE中的乘法。使用数论变换(Number Theoretical Transform, NTT)而不是离散傅里叶变换(Discrete Fourier Transform, DFT)，可以通过使用整数算术而不是浮点算术来避免舍入误差问题[10]。本文中，数论变换乘法特指Schönhage–Strassen算法中使用数论变换的乘法。数论变换和逆数论变换(Inverse Number Theoretical Transform, INTT)作为NTT乘法中的运算核心，占据了NTT乘法中90%以上的运算量和运算时间，优化NTT的速度、面积和功耗，对于NTT乘法的整体性能，具有关键性的影响。

为了加速FHE的运算，Wang等人[11]提出用于全同态加密的大整数乘法架构和实现。Feng等人[12]提出利用中国余数定理的双模数论变换结构，提升了数论变换的效率。施佺等人[13]和谢星等人[14]提出了两级数论变换的结构，提高转换并行度。

本文从数论变换的快速算法出发，研究操作数移位后的“零填充”的空位特征，开展了数论变换乘法蝶形运算的优化工作。本文首先简单介绍数论变换和库利–图基快速变换分解，然后详细给出蝶形运算操作数合并算法和取模操作的快速算法，在此基础上，提出数论变换基32运算单元的硬件设计架构并设计实现，最后给出了仿真验证的结果。

2 数论变换和库利–图基快速变换分解

NTT是传统的DFT在有限域上的一般化。数论变换用旋转因子$W^{\frac{p-1}{N}} \pmod p$来等价DFT运算中的$\mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}}$，其中$W$是模素数$p$的原根，由于$p$是素数，根据狄利克雷定理，其原根$W$一定存在。$N$点NTT及其逆过程INTT的计算式为

$$X_k = \sum_{n=0}^{N-1} x_n (W_N)^{nk} \pmod p \tag{1}$$

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k (W_N)^{-nk} \pmod p \tag{2}$$

其中，$0 \le k \le N-1$, W_N是第N个单位根。将x_n和X_k堆叠成向量$\boldsymbol{x} = (x_0, x_1, \cdots, x_{N-1})^{\mathrm{T}}$和$\boldsymbol{X} = (X_0, X_1, \cdots, X_{N-1})^{\mathrm{T}}$可得到矩阵向量积的等价形式

$$\boldsymbol{X} = \mathbf{NTT}_N \boldsymbol{x} \pmod p, \quad \mathbf{NTT}_N = \left[W_N^{nk}\right]_{0 \le n,k < N} \tag{3}$$

$$\boldsymbol{x} = \mathbf{INTT}_N \boldsymbol{X} \pmod p, \quad \mathbf{INTT}_N = N^{-1}\left[W_N^{-nk}\right]_{0 \le n,k < N} \tag{4}$$

与DFT和IDFT的关系类似，NTT与INTT极为相似，其优化方法和硬件结构可以合并讨论。因为NTT是DFT在有限域上的一般化，对于大点数的数论变换，也可以参考快速傅里叶变换算法中库利–图基快速傅里叶变换算法的思想。对于当前的FHE应用，使用12 Mbit的乘法加密是足够的。在该长度乘法下，需要1048576点的数论变换实现。根据应用要求和结构不同，大点数数论变换常见从基4到基64的分解。蝶形运算由不同的基运算单元实现。基越大数论变换乘法需要级联的级数越少，延时也越小，但每一级的运算量会有$O\left(N^2\right)$的增加。对于$N = 1048576$点的数论变换，可以按照5级基16或4级基32分解。本文优化了基32运算单元，使得基32运算单元的运算量和效率适合硬件系统的实现。将式(1)中的n和k的索引重写，并应用单位根的特性，可以得到

$$\begin{aligned} & X\left(32768k_1 + 1024k_2 + 32k_3 + k_4\right) \\ & = \sum_{n_1=0}^{31} W_{32}^{n_1 k_1} \left(\sum_{n_2=0}^{31} W_{32}^{n_2 k_2} \left(\sum_{n_3=0}^{31} W_{32}^{n_3 k_3} \right.\right. \\ & \quad \left.\left. \cdot \left(\sum_{n_4=0}^{31} x(n) W_{32}^{n_4 k_4} \right) W_{1024}^{n_3 k_4} \right) W_{32768}^{n_2 k_4} \right) \\ & \quad \cdot W_{1048576}^{n_1 k_4} \pmod p \end{aligned} \tag{5}$$

其中，$n_1, n_2, n_3, n_4, k_1, k_2, k_3, k_4$为[0, 32)的整数。

从式(5)的分解可以看出，一个1048576点的数论变换可以被分解成4级基32运算单元和旋转因子乘法的运算。其中旋转因子可以事先计算好并存于ROM中，需要使用时直接读取即可。基32蝶形运算的计算量占数论变换乘法的90%以上，它的优化对数论变换的效率至关重要。

3 蝶形运算的操作数合并算法

3.1 基32运算单元

本文中，蝶形运算由基32运算单元实现，基32运算单元的计算如下

$$X_k = \mathbf{NTT}_{32} \boldsymbol{x} \pmod p = \sum_{n=0}^{31} x_n W_{32}^{nk} \pmod p \tag{6}$$

其中，$0 \le k < 32$, $0 \le n < 32$, p是素数，W_{32}是第32个单位根。

在素数p的选择上，通常选择Solinas素数$p = 2^{64} - 2^{32} + 1$[11]。因为该素数支持高效的取模操

作： $2^{192} \bmod p = 1$， $2^{96} \bmod p = -1$， $2^{64} \bmod p = 2^{32} - 1$。利用该素数计算得到的单位根$W_{32} = 2^6$是2的幂次方的特性，可以将式(6)的乘加运算，方便地转换为移位和模加运算，降低数论变换的计算复杂性。由此，基32运算单元的计算可以写成

$$X_k = \mathbf{NTT}_{32}\boldsymbol{x} \,(\bmod p) = \sum_{n=0}^{31} x_n 2^{6\times nk(\bmod 192)} \,(\bmod p)$$

$$\mathbf{NTT}_{32} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2^6 & 2^{12} & \cdots & 2^{186} \\ 1 & 2^{12} & 2^{24} & \cdots & 2^{180} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2^{186} & 2^{180} & \cdots & 2^6 \end{bmatrix} \tag{7}$$

从式(7)X_k的计算公式可以看出，如果将x_n用0填充的方式，扩展成32个192 bit的操作数(OPerand, OP)，基32运算就转换成了32个操作数乘以相应的旋转因子的总和，也即x_n向左循环移位$6nk \,(\bmod 192)$比特的总和，其中n和k对应式(7)中的n和k。

给定32个操作数，使用进位保存加法器(Carry Save Adder, CSA)树执行求和运算将至少需要8级，这将导致较大面积要求。32个操作数相加后的结果，通过执行模p操作得到最终结果。因为p的位数是64位，所以最终结果为64位。

由于每个操作数中都有128位0，因此可以利用零元素，通过合并兼容的操作数来减少有效操作数的数量，从而可能会降低CSA树的级数，减少复杂度和面积。例如，OP11是x_{11}向左移位66 bit，其余位填充0后得到的。OP0和OP11中的有效数据位是不重叠的，可以把OP0和OP11组合产生一个新的操作数。本文探索了每个X_k的操作数集合中的固有特征，并提出了一种有效的操作数归约算法，以最大限度地减少蝶形运算中的有效操作数。

3.2 操作数合并算法的提出

从式(7)观察到x_n是以6 bit为基本单位，向左循环移位的，为了增加合并操作数的机会，将每个x_n以6 bit为一个基本单位，分成11个字，称为$x_{n,\,m}$, $0 \le m < 11$。一个192 bit的OP中包含有32个字。x_n可以表示为$x_n = \sum_{m=0}^{10} x_{n,m} 2^{6m}$，其中$m$表示第$m$个字，$x_{n,\,m}$的数据宽度是6 bit，则式(7)可以得到

$$X_k = \sum_{n=0}^{31}\sum_{m=0}^{10} x_{n,m} 2^{6\times(m+nk)(\bmod 192)} \,(\bmod p) \tag{8}$$

使用$x_{n,m}$的好处是，将$x_{n,m}$作为基32运算单元的基本单元，可以增加合并操作数的自由度，从而找到一个减少操作数的优化策略。例如对于零填充算法，OP0和OP1分别包含了x_0和x_1，两操作数之间是有重叠的，即两操作数是无法合并的。但是，如果把$x_{n,m}$当成新的操作数，$x_{0,0}$和$x_{1,0}$就是可以合并的。值得注意的是，x_n的分割和$x_{n,m}$的合并能够很容易地用硬件实现，几乎没有硬件开销。

通过仔细分析式(8)不同的k值下，$6\times(m+nk)\,(\bmod 192)$的周期，可以将式(8)分为3种情况讨论，分别是$k=0$, k是奇数，k是偶数且$k\ne 0$。为了重组式(8)，定义$k = 2^i(2j+1)$，其中$0 \le i < 5$, $0 \le j < 16/2^j$, i和j是整数。以下分别讨论这3种情况。

对于第1种，假设$k=0$的特殊情况，可以通过将$k=0$代入式(8)，并考虑零扩展，其结果为

$$X_0 = \sum_{n=0}^{32-1} \underbrace{\left(\sum_{m=0}^{16-1} x_{n,m} \times 2^{6\times m(\bmod 96)}\right)}_{\text{64 bit OP 扩展成 96 bit OP}} \,(\bmod p) \tag{9}$$

对于第2种，假设k是奇数的情况，则$i=0$且$k=2j+1$，式(8)可重新组织为

$$\begin{aligned} X_k &= \sum_{n=0}^{31}\sum_{m=0}^{10} x_{n,m} 2^{6\times(m+n(2j+1))(\bmod 192)} \,(\bmod p) \\ &= \sum_{m=0}^{10} \underbrace{\left(\sum_{n=0}^{31} x_{n,m} \times 2^{6\times(m+n(2j+1))(\bmod 192)}\right)}_{\text{一个新的192 bit OP}} \\ &\quad \cdot \,(\bmod p) \end{aligned} \tag{10}$$

对于第3种，假设k是偶数且$k\ne 0$的情况，需要进一步将m和n分解，以合并操作数。定义$m = 2^i \times \beta + \alpha$, $n = \delta \times \left(\dfrac{32}{2^i}\right) + \gamma$，其结果为

$$\begin{aligned} X_{i,j} &= \sum_{\delta=0}^{2^i-1}\sum_{\gamma=0}^{\frac{32}{2^i}-1}\sum_{\beta=0}^{\left\lceil\frac{11}{2^i}\right\rceil-1}\sum_{\alpha=0}^{2^i-1} x_{\delta\times\left(\frac{32}{2^i}\right)+\gamma,\ 2^i\times\beta+\alpha} \times 2^{6\times\left(\left(2^i\times\beta+\alpha\right)+\left(\delta\times\left(\frac{32}{2^i}\right)+\gamma\right)\times 2^i(2j+1)\right)(\bmod 192)} \,(\bmod p) \\ &= \sum_{\delta=0}^{2^i-1}\sum_{\beta=0}^{\left\lceil\frac{11}{2^i}\right\rceil-1} \underbrace{\left(\sum_{\gamma=0}^{\frac{32}{2^i}-1}\sum_{\alpha=0}^{2^i-1} x_{\delta\times\left(\frac{32}{2^i}\right)+\gamma,\ 2^i\times\beta+\alpha} \times 2^{6\times\left(\left(2^i\times\beta+\alpha\right)+\gamma\times 2^i(2j+1)\right)(\bmod 192)}\right)}_{\text{一个新的192 bit OP}} \,(\bmod p) \end{aligned} \tag{11}$$

其中，$\lceil\cdot\rceil$表示向上取整。实际上，式(10)是式(11)当$\alpha=0$且$\beta=0$时的特殊情况。因此，在操作数合并算法的伪代码中分为$k=0$和$k\neq 0$两种情况，如表1所示。

在下节中，本文将直观地在硬件上展示系统如何有效地执行操作数合并过程。

3.3 操作数的分割和合并

根据以上的算法结果，如图1所示，将每个64 bit输入数据x_n，最高2 bit填充0，分割成11个字，每个字包含6 bit。对于X_0的特殊情况，操作数实际上是对齐的输入数据。换句话说，每个操作数都是从输入数据的11个连续字中派生得出的。由于新的96 bit操作数由16个字组成，前5个字被分配为0。

奇数输出X_1所需要的合并后的操作数，如图2所示。合并后共有11个操作数，每个操作数由32个不同的输入数据$x_{n,m}$, $0\le n\le 32$，使用相同的字索引m, $0\le m<11$合并而成。其余奇数输出的操作数合并方法与X_1类似，其区别在于向左循环移位时的位数不同。使用了操作数合并算法，192 bit的操作数从32个减少到了11个。

对于偶数输出X_2，以2个字为周期合并操作数，合并后的结果如图3所示。这里有两组操作数，每组包括6个合并后的操作数。第1组包含了OP0至OP5；第2组包括OP6至OP11。每个新的192 bit操作数由32个字组成，它们来自16个不同的输入数据，每个输入数据提供两个连续的字。其余除X_0, X_8, X_{16}和X_{24}以外的偶数输出的操作数合并方法与X_2类似，其区别在于向左循环移位时的位数不同，以及合并字的周期不同。在此情况下，192 bit操作数的数量从32减少到12。

对于偶数输出X_8，以8个字为周期合并操作数。合并后有4组操作数，每组包括4个合并后的操作数。每个新的192 bit操作数由32个字组成，它们来自4个不同的输入数据，每个输入数据提供4个连续的字。X_{16}和X_{24}的操作数合并方法与X_8类似，其区别在于向左循环移位时的位数不同，以及合并字的周期不同。在此情况下，192 bit操作数的数量从32减少到16。

基32运算单元的操作数合并前后的数量对比，如表2所示。基32运算单元在合并前需要1024个操作数，在合并后仅需要400个操作数。

表 1 操作数合并算法

输入：原始输入数据x
输出：合并后的操作数OP

{//For X_1 to X_{31} **NTT** output}
for $i=0$ to 4 do
  for $j=0$ to $16/2^i-1$ do
    $k\leftarrow 2^i\times(2j+1)$ {//For OP $[k]$ of k-th **NTT** output}
    for $\delta=0$ to 2^i-1 do
    for $\beta=0$ to CEILING$(11/2^i-1)$ do
      $n\leftarrow \delta\times$CEILING$(11/2^i+\beta)$ {//For OP $[k]$ $[n]$}
      OP $[k]$ $[n]\leftarrow 192'$b0
      for $\gamma=0$ to $32/2^i-1$ do
        for $\alpha=0$ to 2^i-1 do
          $m\leftarrow 2^i\times\beta+\alpha$
          if $m\ge 11$ then
            PASS
          else
            OP $[k]$ $[n]$ $[6(m+\gamma k)+5\,(\mathrm{mod}\,192):$
            $6(m+\gamma k)\,(\mathrm{mod}\,192)]\leftarrow x\,[6m+5:6m]$
          end if
        end for
      end for
    end for
  end for
end for
{//For X_0 **NTT** output}
for $n=0$ to 31 do
  OP $[0]$ $[n]\leftarrow 96'$b0
  OP $[0]$ $[n]$ $[63:0]\leftarrow x\,[n]$
end for

4 取模操作的快速算法

在数论变换的实现中，经常需要执行一系列模

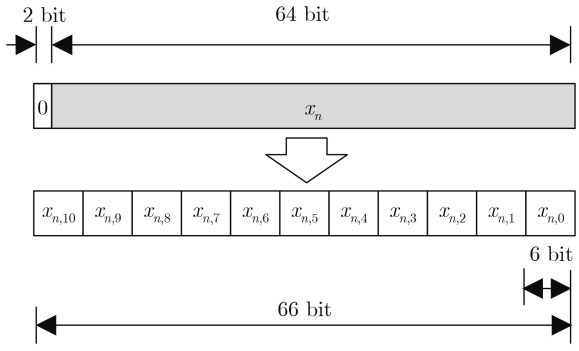


图 1 输入数据x_n的分割

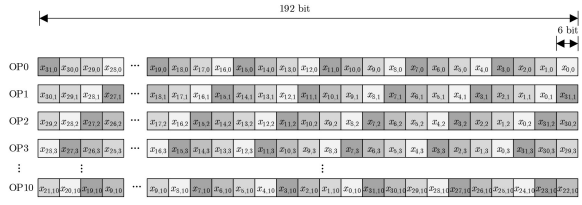


图 2 合并后X_1的操作数

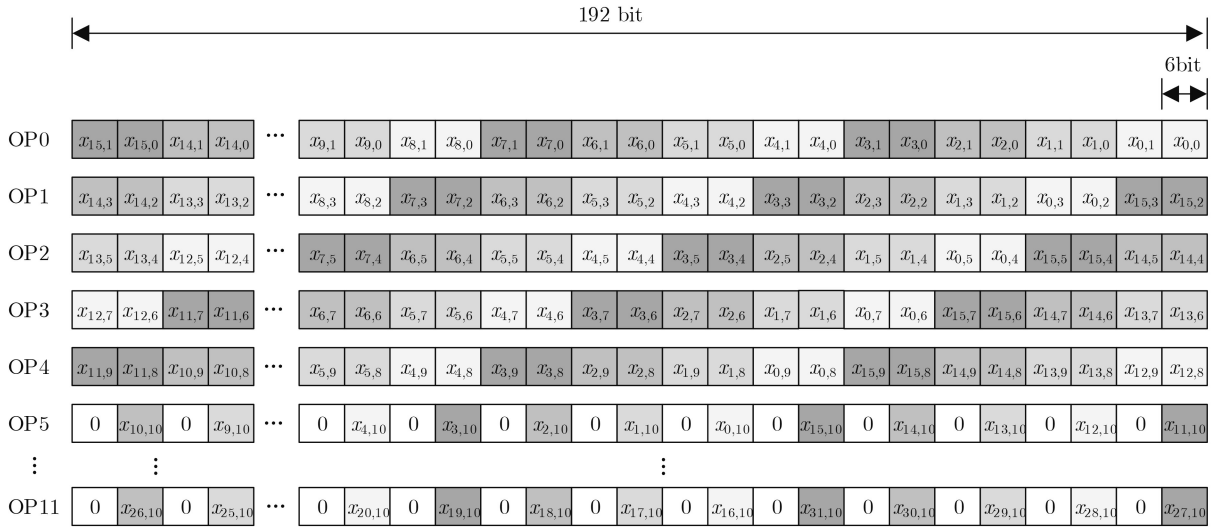


图 3 合并后X_2的操作数

表 2　基32运算单元的操作数

数论变换输出	个数	合并前操作数数量	合并后操作数数量
X_0	1	32	32
奇数输出	16	32	11
X_8, X_{16}, X_{24}	3	32	16
除X_0, X_8, X_{16}, X_{24}以外的偶数输出	12	32	12
合计	32	1024	400

运算，如加、减和乘以2的幂。如果将其实现为64位宽的操作，则需要对中间结果或最终结果取模p。尽管直接计算取模p的过程非常快，但仍然需要大量硬件。然而，如果利用Solinas素数$2^{192} \bmod p = 1$的特性，将每个64位值扩展到192位，并以192位宽的值在流水线上运算，则可以避免在每次运算后执行模p运算，以减少面积。以下给出示例。

(1) 加法：对于算式$x + y$，有两种情况，有进位和无进位。如有进位，$x + y = \text{trunc}\,(x + y) + 2^{192} \equiv \text{trunc}\,(x + y) + 1 \,(\text{mod}\ p)$。其中$\text{trunc}\,(z)$的运算指的是取$z$的低192 bit。如无进位，其取模的结果就是$x + y$。可以使用循环移位操作和单比特加法，在硬件上有效地实现单比特加法后，不需要再考虑进位的问题。

(2) 乘以2的幂：首先考虑乘以2的情况。假设有192 bit的数x，希望计算$2x$。那么有两种情况，如x的首位$x\,[191]$是0，只需要左移1 bit得到$2x$；如x的首位$x\,[191]$是1，那么$2x = \text{trunc}\,(2x) + 2^{192} \equiv \text{trunc}\,(2x) + 1 \,(\text{mod}\ p)$。在这两种情况下，都只需要将$x$循环左移1 bit，即可实现乘以2的模乘。以此类推，如果要计算$2^j x$，只需要对x做循环左移$j$bit，即可实现x乘以2^j的模乘。

(3) 减法：因为$2^{96} \bmod p = -1$，所以$x - y \equiv x + 2^{96} y \,(\text{mod}\ p)$，其中$2^{96}$是常数的乘法系数。这样，减法就转换成了上述的乘法和加法，可以较为简便地计算。

对于将192 bit取模得到64 bit的运算，可以把一个192 bit的数表示成$z = 2^{160}a + 2^{128}b + 2^{96}c + 2^{64}d + 2^{32}e + f$，其中$a$, b, c, d, e和f都是32 bit的数。只需模加和模减就能够满足快速取模的操作。

$$z \equiv \left(2^{32}e + f\right) + \left(2^{32}d + a\right) - \left(2^{32}b + c\right) - \left(2^{32}a + d\right) \,(\text{mod}\ p) \tag{12}$$

特别地，对于X_0的计算，操作数相加的中间结果只有96 bit，其取模操作更为简单。

$$z \equiv 2^{32}d + 2^{32}e + f - d \,(\text{mod}\ p) \tag{13}$$

5　硬件实现

图4显示了数论变换基32运算单元的硬件框图，包括操作数生成和合并模块，模加模块和取模模块，这些模块中应用了上节推导的操作数合并算法和快速取模方法。基32运算单元的输入是32个64 bit数据，输出是数论变换后的32个64 bit数据。

操作数生成和合并模块主要应用操作数合并算法，根据输出序号的不同，将输入数据切割、扩展并合并为：(1)32个96 bit操作数，针对X_0；(2)11个192 bit操作数，针对奇数序列；(3)16个192 bit操作数，针对X_8, X_{16}和X_{24}；(4)12个192 bit操作数，针对剩余的偶数序列。

因为合并后的操作数数量不一，共有4种针对11, 12, 16和32输入操作数的模加模块，图5展示了11输入操作数和16输入操作数的模加模块。11输入操作数和12输入操作数加法中，应用了192 bit的快速取模方法。X_0的32输入操作数，因从64 bit填充

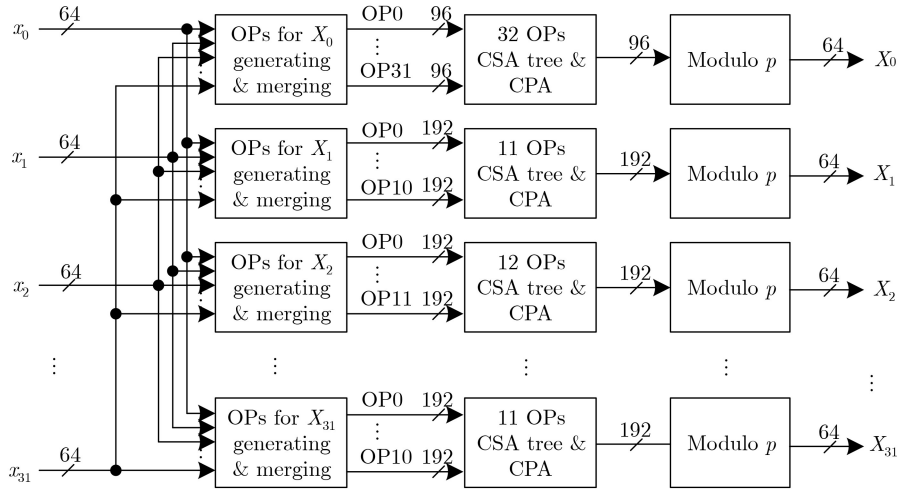
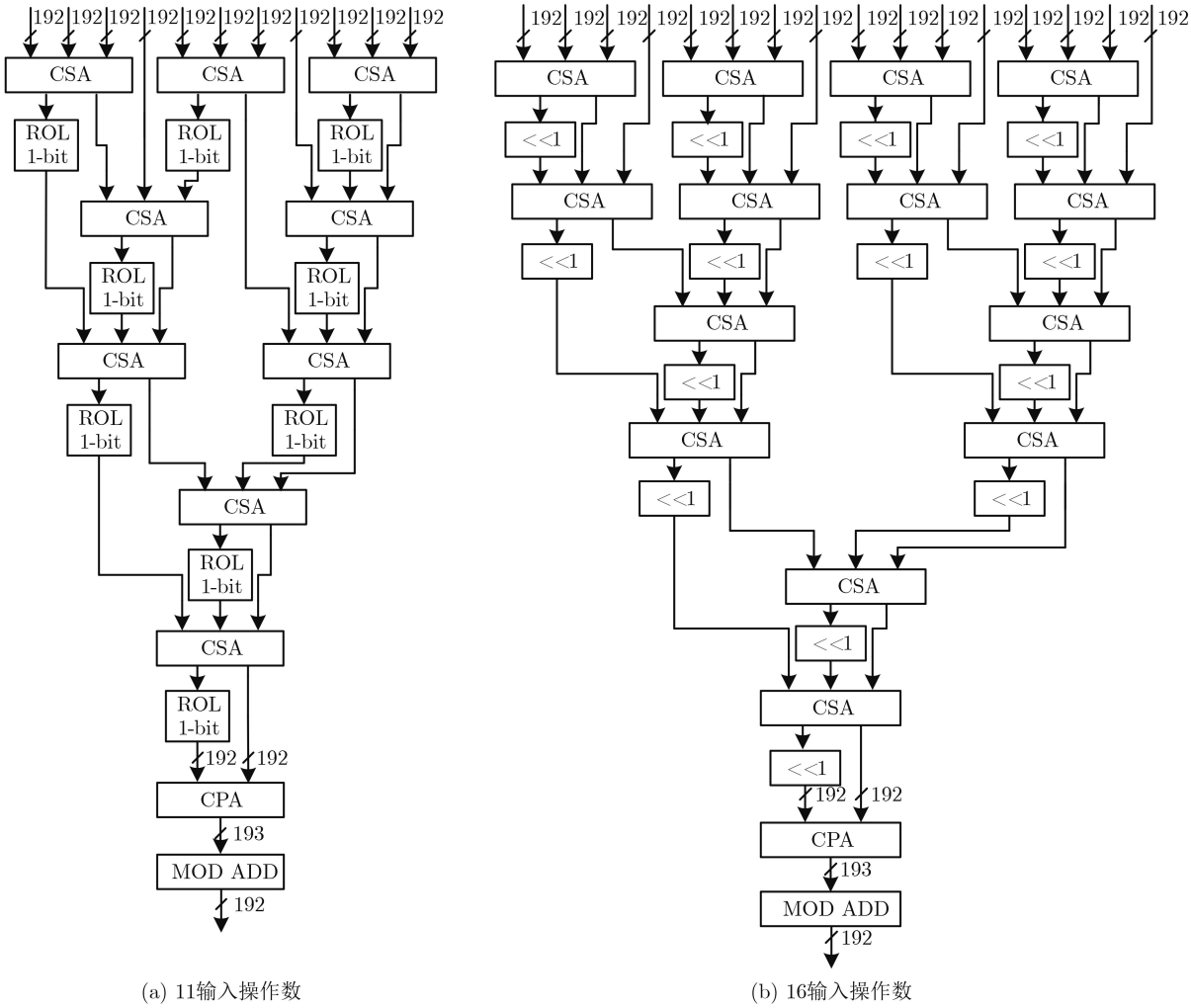


图 4 数论变换基32运算单元的硬件框图



(a) 11输入操作数 (b) 16输入操作数

图 5 输入操作数的模加模块

到了96 bit，其操作数的加法中间结果和最终结果都不会超过96 bit，加法计算中不需要取模操作。X_8, X_{16}和X_{24}的16输入操作数，高32 bit填充了0，其操作数的加法中间结果和最终结果都不会超过192 bit，加法计算中也不需要取模操作。

6 实现结果

本文提出的用于全同态加密的1M点数论变换乘法基32运算单元的优化算法和架构，使用Verilog硬件描述语言编码，结合软件库NTT Library[15]的结果进行UVM验证。使用Synopsys Design Compiler

在SMIC 90 nm RVT 0.9 V Slow工艺下进行综合，电路的最高工作频率为600 MHz，运算需要7个时钟周期，面积为1.714 mm^2，功耗202 mW。功耗的99%为动态功耗，与频率直接相关。FPGA选用了Xilinx Kintex UltraScale 平台和Vivado进行综合，电路的最高工作频率为320 MHz，运算需要7个时钟周期，需要38.9k CLB LUTs, 23.1k CLB Registers和1.24k CARRY8。为了与其他型号的FPGA对比，CLB LUTs可以折合成62.2k logic cells。

本文提出的优化算法，可以扩展到其他2的幂次方基的运算单元上，可将基16和基32运算单元的操作数减少到43.8%和39.1%，由操作数减少带来的加法器所占的面积和关键路径延时有相应的减少。表3是本文结果与其他工作的对比。

表 3　实现结果对比

	操作数总和		最大频率(MHz)	
	基16	基32	ASIC (@90 nm)	FPGA
文献[11]	256		200	229.4 (@Altera Stratix V)
文献[14]		1024		98.02 (@Altera Stratix V)
本文	112	400	600	320 (@Xilinx Kintex UltraScale)
操作数减少(%)	43.8	39.1		

7　结束语

本文利用操作数移位后的“零填充”的空位，合并数论变换乘法蝶形运算的操作数，并采用快速取模，分别可将基16和基32运算单元的操作数减少到43.8%和39.1%。在此基础上，设计实现了数论变换基32运算单元的硬件设计架构。在SMIC 90 nm工艺下和Xilinx Kintex UltraScale FPGA平台上的综合结果显示，电路的最高工作频率为600 MHz和320 MHz，运算需要7个时钟周期，需要的资源分别为芯片面积1.714 mm^2，以及38.9k CLB LUTs, 23.1k CLB Registers和1.24k CARRY8。实验结果表明，该优化算法提升了数论变换乘法蝶形运算的计算效率。

参 考 文 献

[1] ALBRECHT M, CHASE M, CHEN Hao, *et al*. Homomorphic encryption standard[R/OL]. http://homomorphicencryption.org/wp-content/uploads/2018/11/HomomorphicEncryptionStandardv1.1.pdf, 2018.

[2] 陈克非, 蒋林智. 同态加密专栏序言[J]. 密码学报, 2017, 4(6): 558–560. doi: 10.13868/j.cnki.jcr.000207.
CHEN Kefei and JIANG Linzhi. Preface on homomorphic encrpytion[J]. *Journal of Cryptologic Research*, 2017, 4(6): 558–560. doi: 10.13868/j.cnki.jcr.000207.

[3] 李增鹏, 马春光, 周红生. 全同态加密研究[J]. 密码学报, 2017, 4(6): 561–578. doi: 10.13868/j.cnki.jcr.000208.
LI Zengpeng, MA Chunguang, and ZHOU Hongsheng. Overview on fully homomorphic encryption[J]. *Journal of Cryptologic Research*, 2017, 4(6): 561–578. doi: 10.13868/j.cnki.jcr.000208.

[4] ARCHER D, CHEN L, CHEON J H, *et al*. Applications of homomorphic encryption[EB/OL]. http://homomorphicencryption.org/white_papers/applications_homomorphic_encryption_white_paper.pdf, 2017.

[5] GENTRY C. Fully homomorphic encryption using ideal lattices[C]. The 41st Annual ACM Symposium on Theory of Computing, Bethesda, USA, 2009: 169–178. doi: 10.1145/1536414.1536440.

[6] ABOZAID G, EL-MAHDY A, and WADA Y. A Scalable multiplier for arbitrary large numbers supporting homomorphic encryption[C]. 2013 Euromicro Conference on Digital System Design, Los Alamitos, USA, 2013: 969–975. doi: 10.1109/DSD.2013.110.

[7] RAFFERTY C, O'NEILL M, and HANLEY N. Evaluation of large integer multiplication methods on hardware[J]. *IEEE Transactions on Computers*, 2017, 66(8): 1369–1382. doi: 10.1109/TC.2017.2677426.

[8] VAN METER R and ITOH K M. Fast quantum modular exponentiation[J]. *Physical Review A*, 2005, 71(5): 052320. doi: 10.1103/PhysRevA.71.052320.

[9] GARCÍA L. Can Schönhage multiplication speed up the RSA encryption or decryption?[EB/OL]. https://www.informatik.tu-darmstadt.de/cdc/home_cdc/index.de.jsp, 2005.

[10] FÜRER M. Faster integer multiplication[C]. The 39th Annual ACM Symposium on Theory of Computing, San Diego, USA, 2007: 57–66. doi: 10.1145/1250790.1250800.

[11] WANG Wei, HUANG Xinming, EMMART N, *et al*. VLSI design of a large-number multiplier for fully homomorphic encryption[J]. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2014, 22(9): 1879–1887. doi: 10.1109/TVLSI.2013.2281786.

[12] FENG Xiang and LI Shuguo. Design of an area-effcient million-bit integer multiplier using double modulus NTT[J].

IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2017, 25(9): 2658–2662. doi: 10.1109/TVLSI.2017.2691727.

[13] 施佺, 韩赛飞, 黄新明, 等. 面向全同态加密的有限域FFT算法FPGA设计[J]. 电子与信息学报, 2018, 40(1): 57–62. doi: 10.11999/JEIT170312.

SHI Quan, HAN Saifei, HUANG Xinming, *et al*. Design of finite field FFT for fully homomorphic encryption based on FPGA[J]. *Journal of Electronics & Information Technology*, 2018, 40(1): 57–62. doi: 10.11999/JEIT170312.

[14] 谢星, 黄新明, 孙玲, 等. 大整数乘法器的FPGA设计与实现[J]. 电子与信息学报, 2019, 41(8): 1855–1860. doi: 10.11999/JEIT180836.

XIE Xing, HUANG Xinming, SUN Ling, *et al*. FPGA design and implementation of large integer multiplier[J]. *Journal of Electronics & Information Technology*, 2019, 41(8): 1855–1860. doi: 10.11999/JEIT180836.

[15] Project Nayuki. Number-theoretic transform (integer DFT)[EB/OL]. https://www.nayuki.io/page/number-theoretic-transform-integer-dft, 2017.

华斯亮：男，1981年生，副研究员，研究方向为专用集成电路设计、高性能计算.

张惠国：男，1978年生，副教授，研究方向为集成电路设计、功率电子技术.

王书昶：男，1985年生，讲师，研究方向为半导体光电子材料与器件.

责任编辑：马秀强