高速流水线结构的大整数乘法器 FPGA 设计与实现

涂振兴，王晓蕾，杜高明，李桢旻
（合肥工业大学 微电子设计研究所，合肥 230601）

摘　要： 大整数乘法是密态数据计算中最为耗时的基本运算操作，提高大数乘法单元的计算速度在全同态加密机器学习等应用中尤为重要。提出了一种输入数据位宽为 768 kbit 的高速大整数乘法器设计方案，将核心组件 64 k 点有限域快速数论变换（NTT）分解成 16 点 NTT 实现，并通过算法分治处理，细化 16 点 NTT 的流水线处理过程。采用加法和移位来实现模减计算单元，并利用高效的无冲突地址生成算法完成数据交互，实现大整数乘法的高速化。该乘法器最后被部署在 Altera Stratix-V FPGA 开发板上，实验结果表明，电路工作频率为 169.23 MHz，大整数乘法计算总体耗时 0.317 ms。对比现有的硬件设计，在速度性能上有 1.2 倍至 7.3 倍的提升。

关键词： 高速；流水线；大整数乘法器；NTT；FPGA
中图分类号：TN492　　**文献标志码：**A　　**文章编号：**1004-3365(2022)01-0006-06
DOI：10.13911/j.cnki.1004－3365.210095

FPGA Design and Implementation of a Large Integer Multiplier with High Speed Pipeline Structure

TU Zhenxing, WANG Xiaolei, DU Gaoming, LI Zhenmin
(*Institute of VLSI Design, Hefei University of Technology, Hefei* 230601, *P. R. China*)

Abstract: Large integer multiplication is the most time-consuming operation during encrypted data calculation. It is particularly important to improve large integer multiplier speed in machine learning based on fully homomorphic encryption. A design scheme of high speed 768 kbit large integer multiplier was proposed in this paper. The critical component 64k-point finite field number theory transform (NTT) was decomposed into 16-point NTT. And through dichotomy processing, the pipeline architecture of 16-point NTT was refined. To increase the speed of the multiplier, addition and shift were adopted to achieve the modular-subtraction unit, and data interaction was accomplished by using an efficient non-conflict address algorithm. The multiplier was deployed on the Altera Stratix-V FPGA development board. And the experimental results showed that the circuit had a working frequency of 169.23 MHz and took 0.317 ms to complete the large integer multiplication. Comparing with the state-of-the-art works, our speed performance was improved by 1.2 times to 7.3 times.

Key words: high speed; pipeline; large integer multiplier; NTT; FPGA

0　引　言

全同态加密（Fully Homomorphic Encryption，FHE）机器学习具有很好的隐私保密性，受到很多密码学研究者的重视[1]。同态加密神经网络支持密文输入，经过推理之后，对密文输出进行解密，最后获得解密结果。由于在网络推理过程中使用的都是密文，因此安全性高。但是 FHE 存在计算复杂度高和计算时间过长等问题，使得同态加密神经网络

收稿日期：2021-03-10；**定稿日期：**2021-04-25
基金项目：国家重点研发计划（2018YFB2202604）；安徽省高校协同创新项目（GXXT-2019-030）
作者简介：涂振兴（1997—），男（汉族），江西南昌人，硕士，研究方向为同态加密技术与集成电路设计。

很难应用到具有实时性要求的场景中，其中上万比特级乘法运算是最为耗时的操作，如文献[2]中，为了满足加密的需求，需要使用 785 006 bit 的乘法器。因此实现快速的大整数乘法运算对促进密态机器学习等加密体制的研究具有重要意义。

64 k(1 k＝1 024)点有限域快速数论变换(Numerical Theory Transform，NTT)是构造大整数乘法的核心模块，加速该模块是提高大整数乘法速度的关键。文献[3]通过将 64 k 点 NTT 进行两级分解，拆分成 16 点 NTT 和 64 点 NTT 来实现，并分别采用并行度为 16 和 8 的树形求和单元进行设计。文献[4]同样经过两级分解，将 64 k 点 NTT 拆分成 32 点 NTT 和 64 点 NTT，利用 4 级加法树结构进行设计，并采用同址运算算法[5]访问存储单元，减少了部分逻辑资源。相对于文献[3]，文献[4]消耗的硬件资源下降，但是由于降低了流水线并行度，计算时间变长。文献[6-7]将 64 k 点 NTT 算法经过 3 次分解，拆分后可以只采用 16 点 NTT 算法来实现，优化了电路结构，降低了设计成本。

基于上述讨论，文献[3-4]虽然对算法结构进行了优化，但是采用不同长度的 NTT 作为基本计算单元会消耗额外的硬件资源，增加设计难度。此外，文献[4]中采用的加法树求和单元的计算延时较长，导致电路频率低，流水线结构还可以进一步细化。从整体模块划分来看，应采用长度统一的 NTT 模块作为底层基本计算单元来构造 64k 点 NTT 模块，如文献[6-7]的底层计算单元只包含 16 点 NTT 模块。但是文献[6-7]仍采用加法树求和单元设计，并没有在算法层面上进行优化。为了进一步提升大整数乘法速度，本文主要做出的改进如下。

1) 采用 16 点 NTT 作为基本计算单元来构造 64 k 点 NTT 模块，不同于文献[6-7]，本文对 16 点 NTT 进行算法分治处理，有效提高了数据处理的并行度。

2) 优化大整数乘法整体运算的流水线处理流程，减少每一级流水线执行的逻辑操作，有效缩减了关键电路延时，提高了电路频率。

3) 采用基于 Single-Port Merged-Bank(SPMB)结构的无冲突地址算法优化内存访问过程，提高数据交互速度。

实验结果表明，本文提出的 768 kbit 的大整数乘法器(输入数据范围为 0 至 $2^{786\,432}-1$)硬件设计与现有的设计方案相比，在 FPGA 上的运行效率最高，耗费时间最短。

1　背景技术

1.1　大整数乘法算法

大整数乘法 Schönhage-Strassen 算法[8](SSA)是一种基于 NTT 的多项式乘法算法，具体描述如下。

1) 给定输入为两个 u bit 的大数据 X 和 Y，基数 $B=2^b$ 以及模数 P。

2) 通过使用基数 B 将两个大数据拆分成两组长度为 $M=u/b$ 的数据序列，再通过填充 M 项零数据，变为两组长度为 $2M$ 的序列 $x(i)$ 和 $y(i)$。

3) 分别对数据序列 $x(i)$ 和 $y(i)$ 进行 NTT，表示为 $X(i)=\mathrm{NTT}(x(i))$ 和 $Y(i)=\mathrm{NTT}(y(i))$。

4) 将 $X(i)$ 和 $Y(i)$ 两组数据序列进行点乘，得到 $Z(i)$，表示为 $Z(i)=X(i)\cdot Y(i)$。

5) 对 $Z(i)$ 进行逆数论变换(Inverse NTT，INTT)，得到 $z(i)$，表示为 $z(i)=\mathrm{INTT}(Z(i))$。

6) 再对 $z(i)$ 进行进位处理，$z(i+1)=z(i+1)+\mathrm{Floor}(z(i)/B)$，且 $z(i)=z(i) \bmod B$，Floor()表示向下取整函数。

7) 最后结果为：

$$Z=\sum_{i=0}^{2M-1} z(i)\times B^i \tag{1}$$

SSA 算法的组成部分主要是 NTT 模块和 INTT 模块，加速长序列的 NTT 运算是提升整个乘法器性能的关键。一个 N 点的 NTT 计算式为：

$$X(k)=\sum_{n=0}^{N-1} x(n)\times W_N^{nk} (\bmod P) \tag{2}$$

式中，$k=0, 1, \cdots, N-1$，W_N 是定义在有限域中 N 点 NTT 的原根，类似地，N 点 INTT 计算式为：

$$x(n)=N^{-1}\sum_{k=0}^{N-1} X(k)\times W_N^{-nk} (\bmod P) \tag{3}$$

式中，$n=0, 1, 2, \cdots, N-1$。对比式(2)和式(3)可以看出，NTT 与 INTT 计算结构相似，因此在硬件电路设计中可以通过复用底层电路，减少资源消耗。本文选择 $b=24$，$B=2^{24}$，则输入的两个 768 kbit 大整数被分解成长度为 $M=32$ k 的数据序列，再填充 M 项零数据之后，两组数据序列的长度变为 $2M=64$ k，因此本文需要设计 $N=64$ k 点 NTT/INTT 电路。此外，为了保证最后的卷积计算结果不会溢出，模数 P 需要满足条件 $P>(N/2)\times(B-1)^2$，故本文选取 Solinas 素数作为模数 P($P=2^{64}-2^{32}+1$)。

1.2 高效的内存地址算法

对于大量数据的存储，单端口(Single-port)的 SRAM 比双端口(Dual-port)的 SRAM 有更高的面积效率。然而，由于 Single-port SRAM 不支持在一个时钟周期内对存储块同时进行读写操作，因此读写带宽受限。为了解决这一问题，文献[9]提出基于 Single-port Merged-Bank(SPMB)结构的内存地址算法，显著提高了长序列 NTT 算法的数据交互速度，其基本的设计思想如下。

将 Single-port SRAM 划分成两个 Bank，按照读写规则对这两个 Bank 进行读写操作，保证在不会出现数据冲突的前提下，对一个 Bank 执行读操作，对另一个 Bank 执行写操作，以此等效实现一个高数据吞吐率的存储块。如果采用基 r 点 NTT 来实现 N 点 NTT，将会存在 $p(p=\log_r N)$个计算阶段，此时 SPMB 结构算法执行步骤描述如下。

1) 令 $n=[n_{m-1}, n_{m-2}, \cdots, n_1, n_0]_2$，输入的数据首先按照下式规则预存到 Bank 中：

$$Bank: n_{m-s-1} \oplus n_{m-s-2} \oplus n_{m-s-3} \oplus \cdots \oplus n_1 \oplus n_0,$$
$$Address: [n_{m-s-1}, \cdots, n_1]_2 \tag{4}$$

式中，$m=\log_2 N$，$s=\log_2 r$，$\oplus$ 代表异或操作。$Bank$ 为存储数据的两个存储块的标签，由于是通过异或操作得到，所以只有 0 和 1 两种结果；$Address$ 表示当前 $Bank$ 的存储地址，每一个 $Address$ 将存放索引值为$[n_{m-1}, n_{m-2}, \cdots, n_{m-s}]$的数据，总共 2^s 个数据。

2) 定义一个蝴蝶计数值 BC：

$$BC \leftarrow j \times g + Gray(i) \tag{5}$$

式中，$j=0,1,\cdots,r-1$，$i=0,1,\cdots,g-1$，$g=N/r^2$，Gray(i)表示取 i 的格雷码。

3) 通过下式获得阶段 $t(t=0,1,\cdots,p-1)$的读写 Bank 标签 BN 和 Bank 的存储地址 MA：

$$BN(BC,t) \leftarrow \wedge RR(BC, s \times t)$$
$$MA(BC,t) \leftarrow Floor(RR(BC, s \times t)/2) \tag{6}$$

式中，$\wedge$ 表示按位异或运算，RR 和 Floor 分别表示循环右移和向下取整操作。本文采用 SPMB 结构地址算法模块生成 Bank 的读写地址，同时协同其他子模块并行流水操作，有效提高了 64 k 点 NTT 模块的运行速度。

2 大整数乘法设计方案

2.1 基 16-NTT 高速流水设计

实现 768 kbit 大整数乘法需要设计 64 k 点 NTT 模块，但是 64 k 点 NTT 在有限域中的原根取值为 0x54df9630bf79450e，如果直接使用该数据硬件实现，则会增加设计的复杂度[10]。为了简化设计，利用 16 点 NTT 在有限域中的原根取值为 2^{12}的特点，将 64 k 点 NTT 拆分成 16 点 NTT 来实现[4]，就可以将原根乘法转化为移位操作，达到降低硬件实现难度的目的。16 点 NTT 计算公式为：

$$X(k)=\sum_{n=0}^{15} x(n) 2^{12nk \% 192} \bmod P \tag{7}$$

16 点 INTT 计算公式为：

$$x(n)=16^{-1} \sum_{k=0}^{15} X(k) 2^{-12nk \% 192} \bmod P \tag{8}$$

式中，只包含加法和移位操作，因此对硬件实现友好。此外，随着累加结果越来越大，占用的寄存器资源会变多，利用 $2^{192} \bmod P=1$ 的特点，通过高位补零将 64 bit 数据扩展为 192 bit 计算，如果累加结果超过 192 bit，则通过模 P 运算，可以将累加结果缩减至 192 bit，达到减少寄存器资源的目的。

为了提高 16 点 NTT 电路频率，本文将 16 点 NTT 进行分治处理，拆分成 4 级 2 点 NTT 来实现，因此提高了计算并行度。这样设计的另一点好处是，如果采用多级流水线结构设计 2 点 NTT 模块，则 16 点 NTT 模块也实现了流水线设计，电路频率得到了很大的提升。2 点 NTT 实现 16 点 NTT 的数据流程如图 1 所示。16 点 NTT 经过分治之后，被拆分成 4 个计算阶段，每个阶段将包含 8 个 2 点 NTT 并行计算，因此 16 点 NTT 模块总共包含 32 个 2 点 NTT 模块。

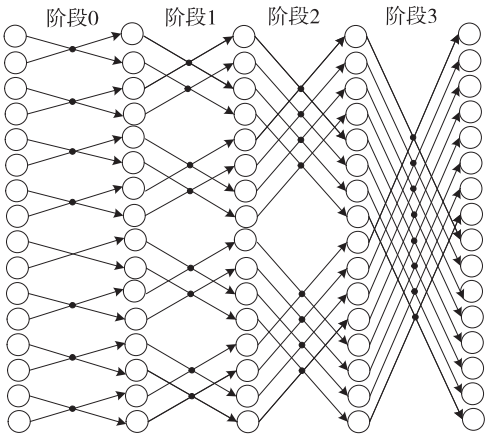

图 1 2-NTT 实现 16-NTT 流程图

在图 1 所示的流程中，上一个阶段的输出需要乘以旋转因子才能作为下一阶段的输入，此过程可能会产生进位数据，因此本文扩展了 2 点 NTT 计算公式，增加一个 192 bit 的进位输入 C：

$$X_0=(x_0+x_1+C)\bmod P,$$
$$X_1=(x_0+((x_1+C)\ll 96))\bmod P \tag{9}$$

2 点 NTT 模块硬件结构如图 2 所示。电路采用 3 级流水线实现，包含五个加法和两个移位。为了防止计算结果超过 192 bit，根据模域运算规则，将中间数据的高 192 bit(MSB)和低 192 bit(LSB)相加，作为最后结果输出。此外，2 点的 NTT 原根与 INTT 原根都为 2^{96}，因此正逆变换可以采用同一块电路完成，从而有效减少硬件资源。

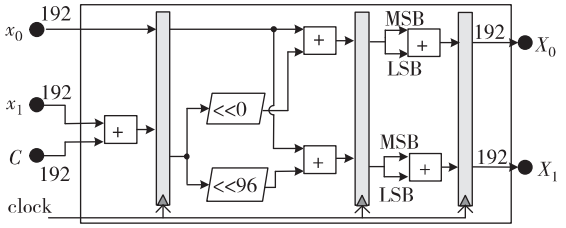

图 2　2-NTT/INTT 电路结构

2.2　模减单元流水线设计

在本文提出的硬件设计方案中，两个大整数通过 $B=2^{24}$ 分解之后得到两组位宽为 24 bit 的数据序列。因为模 P 之后的数据位宽一定小于 64 bit，所以将 24 bit 数据扩展为 64 bit 保存在 SRAM 中。然后将 64 bit 数据扩展为 192 bit 数据，输入到 16 点 NTT 处理单元中计算，得到 192 bit 输出数据。为了统一数据位宽，需要对输出的 192 bit 数据进行模 P 操作，将输出变为 64 bit 数据，此过程可以利用模数 P 的计算特点，采用快速取模操作来完成：

$$\begin{aligned} z_1\equiv&(2^{160}a+2^{128}b+2^{96}c+2^{64}d+2^{32}e+f)\\ &\bmod P\equiv(-(2^{32}-1)a-2^{32}b-c+\\ &(2^{32}-1)d+2^{32}e+f)\bmod P\equiv\\ &((e+d)2^{32}+f+a-((b+a)2^{32}+\\ &c+d))\bmod P \end{aligned} \tag{10}$$

式中，a、b、c、d、e、f 都是 32 bit 数据，模 P 运算被转化成 32 bit 数据的移位运算和加法运算。式(10)的硬件实现电路如图 3 所示，电路结构采用 3 级流水线结构。计算过程中，当 $e+d$ 或 $b+a$ 产生进位时，利用 $2^{64}\bmod P=2^{32}-1$，将进位数据复制成 32 bit 数据，再累加回求和结果，避免中间数据位宽超过 64 bit 的情况出现。

另一方面，SSA 算法中的点乘需要将两个大整数经过 NTT 运算之后的结果相乘，即需要执行 64 bit 乘法操作，得到 128 bit 数据。此时需要对该输出数据再次取模，缩减至 64 bit，才能保存到 SRAM 中。同样利用模数 P 的特点也可以实现快速模 P 运算：

$$\begin{aligned} z_2\equiv&(2^{96}a+2^{64}b+2^{32}c+d)\bmod P\equiv\\ &(-a+(2^{32}-1)b+2^{32}c+d)\bmod P\equiv\\ &((b+c)\times 2^{32}+d-(b+a))\bmod P \end{aligned} \tag{11}$$

式中，a、b、c、d 也是 32 bit 数据，因此也可转化为 32 bit 的加法、减法和移位来实现，本文提出的流水线结构电路如图 4 所示。

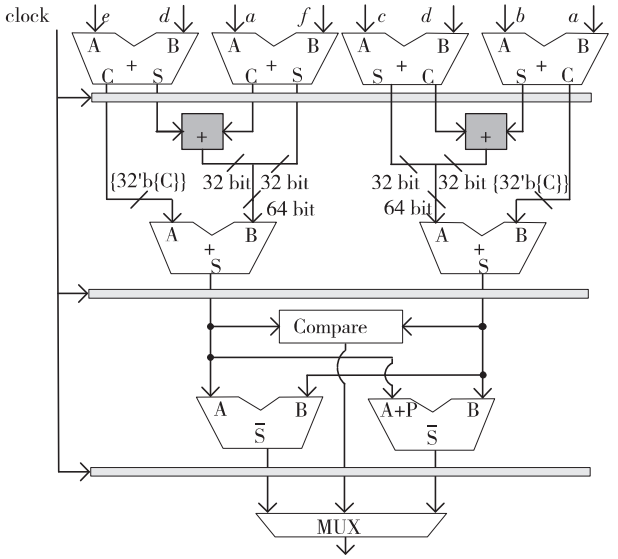

图 3　192 bit 数据模减至 64 bit 数据模块电路

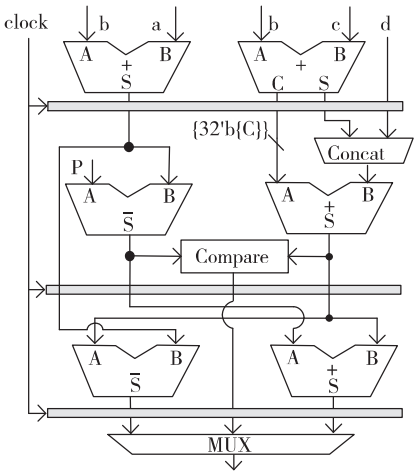

图 4　128 bit 数据模减至 64 bit 数据模块电路

2.3　64 k 点 NTT 设计实现

为了避免采用多种底层的 NTT 架构来实现 64 k 点 NTT 模块，以降低设计成本，本文只选择使用 16 点 NTT 来构造 64 k 点 NTT，此时需要经过 4($4=\log_r N$)个处理阶段，其中 $r=16$，$N=64$ k。第一级分解公式为：

$$X(k)=\sum_{n1=0}^{15}W_{16}^{n1k1}\left[\sum_{n2=0}^{4\,095}x(n_2)W_{4\,096}^{n2k2}\right]W_{16\times 4\,096}^{n1k2} \tag{12}$$

式中，$k_1=0, 1, \cdots, 15$，$k_2=0, 1, \cdots, 4\ 095$。分解之后，64 k 点 NTT 被拆分成 16 个 4 096 点 NTT。按照同样的拆分方法，将 4 096 点 NTT 拆分成 16 个 256 点 NTT，再将 256 点 NTT 拆分成 16 个 16 点 NTT 实现，最后 64 k 点 NTT 被拆分成 4 个计算阶段，每一个阶段包含 4 096 次 16 点 NTT 运算。本文提出的 64 k 点 NTT 整体结构如图 5 所示。

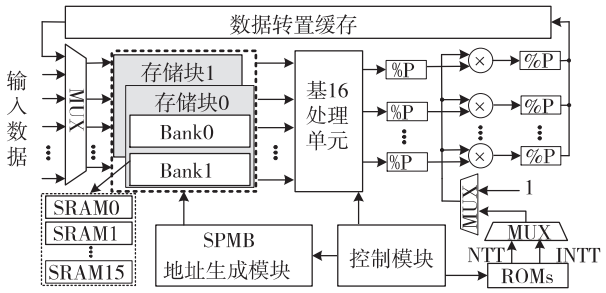

图 5　64 k 点 NTT/INTT 结构框图

图中“%”表示取模运算符。为了节省硬件资源，结构中只包含一个基 16 处理单元，通过控制信号，该处理单元可实现 NTT 和 INTT 的相互转换。两个存储块用来分别保存两组长度为 64 k 的数据序列，每一个存储块包含 2 块 Bank，用来保存输入数据和中间数据，每个 Bank 由 16 个 Single-port SRAM 组成。每一个周期从存储块中并行输出 16 个数据至基 16 处理单元，计算得到 16 个输出数据。输出数据经过取模之后，与从 ROM 模块中读出的旋转因子数据进行乘法和取模运算，然后经过转置处理，写回存储块中。

整个计算流程采用流水线处理。16 点 NTT/INTT 处理单元采用 12 级流水处理，乘法模块和模减模块分别采用 14 级和 3 级流水处理。因此，读 Bank 过程相对写 Bank 过程滞后 $3r$ 个周期，多余的计算周期被用于数据转置缓存模块。完成一次 NTT/INTT 运算，包含 4 个计算阶段，每一个阶段将花费 N/r 个计算周期。为了防止相邻计算阶段的读写交替过程中出现数据冲突的情况，在相邻计算阶段之间阻塞(Stall)了 $2r$ 个周期。数据处理流程如图 6 所示，最终的计算周期可以通过下式计算得到：

$$Cycles = \frac{N}{r}\times \log N_r + (\log N_r - 1)2r + 3r \tag{13}$$

最终的计算周期为 16 528。

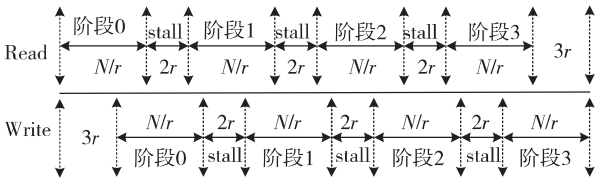

图 6　64 k 点 NTT/INTT 数据处理流程图

3　硬件架构实现与结果分析

3.1　整体硬件架构

为了减小大整数乘法电路面积，本文采用一个 64 k 点 NTT/INTT 处理模块来完成两次 NTT 和一次 INTT 计算，电路结构如图 7 所示。64 k-NTT/INTT 处理模块分别对输入的两组数据序列进行 NTT 处理，并将得到的结果保存在内部的两个存储块中。之后从存储块中读出数据，进行点乘运算，再写回其中一个存储块中。最后进行 INTT 计算，将 INTT 计算结果输入到进位累加模块，计算得到最后大整数乘法结果。

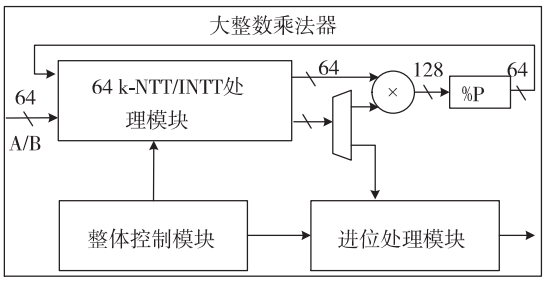

图 7　大整数乘法器整体结构

3.2　结果分析

为了验证本文硬件设计的正确性和高效性，本文采用基于 C 语言设计的 GMP 大整数运算库搭建大整数乘法软件验证平台，CPU 型号为 Intel® Xeon Gold 6154 处理器，主频为 3.0 GHz，操作系统为 Ubuntu16.04。硬件开发平台选择 Altera Stratix-V 5SGXMB5R2F43I2 FPGA 开发板。两个平台的实现结果表明，输入相同的数据，输出结果相同，验证了大整数乘法器的功能是正确的。整体电路综合实现资源如表 1 所示。设计总共花费 256 k ALUT 以及 410 k Register，最大工作频率可达 169.23 MHz，完成 768 kbit 大整数乘法总共需 53 680 个周期，花费的计算时间为 0.317 ms。硬件与 CPU 实现结果的速度对比如表 2 所示。与 CPU 平台相比，本文硬件设计的运行速率提升了 23 倍。

表 1　Altera Stratix-V FPGA 综合实现结果

逻辑资源消耗	资源利用		
	消耗	总体	利用率
ALUT	256 702	370 000	69.38%
Register	410 259	740 000	55.44%
DSP	384	399	96.24%
频率/MHz		169.23	
时钟周期		53 680	
时间/ms		0.317	

表 2　软件与硬件速度对比

实现平台	计算时间/ms	加速倍率
软件	7.286	1
硬件	0.317	23

本文工作与已有的大整数乘法硬件设计比较结果如表 3 所示。相比于文献[3-4]，本文具有最高的工作频率，因为本文采用算法分治的设计方法实现基 16 点 NTT 模块，设计中使用的模乘、模减计算单元全部采用流水线设计，关键电路延时短。虽然完成整个乘法花费的时钟周期数较多，但频率提升带来的加速性能更好，因此总体计算耗时最短。

表 3　硬件整体性能对比

设计	文献[3]	文献[4]	文献[11]	本文
位宽/kbit	768	768	768	768
硬件平台		Stratix-V		
频率/MHz	100	98.02	181	169.23
周期/M	0.037 5	0.041 1	0.420 0	0.053 7
ALUT/k	463	240	7.57	256
寄存器/k	336	236	3.44	410
时间/ms	0.375	0.419	2.3	0.317

在硬件资源消耗方面，本文采用 SPMB 结构的地址生成算法，提高了数据交互效率，简化了控制逻辑，本文硬件设计占用的 ALUT 资源是文献[3]的一半。文献[11]通过在 NTT 实现过程中加入“负卷积”操作，避免了“0 填充”，有效减少计算的输入数据，降低 FPGA 资源消耗。本文采用电路面积换取计算效率的设计策略，计算速度相比文献[11]提升约 7 倍。

4　结　论

本文基于有限域 NTT 算法，设计了一个 768 kbit 的大整数乘法模块，采用算法分治实现 16 点 NTT 模块，并利用该模块构造 64 k 点 NTT 模块。硬件设计全程采用流水线处理，有效提高电路频率。采用 SPMB 结构地址算法生成读写数据地址，在满足高速流水线工作条件下，实现数据的无冲突读取和写入。在 FPGA 开发板上的综合实现结果表明，本文设计的大整数乘法器有较高的工作频率和最短的计算时间，但是在硬件资源消耗上，本文提出的乘法器架构尚有优化空间。

参 考 文 献：

[1] PLANTARD T, SUSILO W, ZHANG Z. Fully homomorphic encryption using hidden ideal lattice [J]. IEEE Trans Inform Forens Secur, 2013, 8(12): 2127-2137.

[2] GENTRY C, HALEVI S. Implementing Gentry's fully-homomorphic encryption scheme [C] // EUROCRYPT'11. Tallinn, Estonia. 2011: 129-148.

[3] WANG W, HUANG X M. FPGA implementation of a large-number multiplier for fully homomorphic encryption [C] // IEEE Int Symp Circ Syst. 2013: 2589-2592.

[4] XING X, HUANG X M, LING S, et al. FPGA design and implementation of large integer multiplier [J]. J Elec Inform Tech, 2019, 41(8): 1855-1860.

[5] HUANG X M, WANG W. A novel and efficient design for an RSA cryptosystem with a very large key size [J]. IEEE Trans Circ Syst Ⅱ: Expr Bri, 2015, 62(10): 972-976.

[6] YE J H, SHIEH M D. Low-complexity VLSI design of large integer multipliers for fully homomorphic encryption [J]. IEEE Trans VLSI Syst, 2018, 26(9): 1727-1736.

[7] WANG W, HUANG X M. VLSI design of a large-number multiplier for fully homomorphic encryption [J]. IEEE Trans VLSI Syst, 2014, 22(9): 1879-1887.

[8] ZHANG N, QIN Q, YUAN H, et al. NTTU: an area-efficient low-power NTT-uncoupled architecture for NTT-based multiplication [J]. IEEE Trans Comput, 2020, 69(4): 520-533.

[9] LUO H F, LIU Y J, SHIEH M D. Efficient memory-addressing algorithms for FFT processor design [J]. IEEE Trans VLSI Syst, 2015, 23(10): 2162-2172.

[10] FENG X, LI S G. Design of an area-efficient million-bit integer multiplier using double modulus NTT [J]. IEEE Trans VLSI Syst, 2017, 25(9): 2658-2662.

[11] FENG X, LI S G. Accelerating an FHE integer multiplier using negative wrapped convolution and PingPong FFT [J]. IEEE Trans Circ Syst Ⅱ: Expr Bri, 2019, 66(1): 121-125.