文献引用格式：张子昂，陈朝晖 .ZUC 算法高性能硬件实现关键技术研究 [J]. 信息安全与通信保密，2022(8):2-11.

ZHANG Ziang,CHEN Zhaohui.Research on Key Technologies for High-Performance Hardware Implementation of ZUC Algorithm[J].Information Security and Communications Privacy,2022(8):2-11.

ZUC 算法高性能硬件实现关键技术研究 *

张子昂，陈朝晖

（中国科学院大学 密码学院，北京 100049）

摘　要：祖冲之序列密码算法是我国自主设计的应用于 LTE 网络的国际加密标准。为适应服务器对数据实时高速加密传输、备份的需求，结合相关算法结构及特性，提出了两种基于现场可编程逻辑门阵列的祖冲之序列密码算法高效实现方案，应用了比特模加优化结构和并行流水线结构。实验最终在 XILINX Kintex-7 FPGA 平台上对设计进行了仿真和实现，并对其运行时的性能和消耗面积结果进行评估，实际结果运行达到 6.4 Gbit/s 的吞吐量。

关键词：祖冲之序列密码算法；现场可编程逻辑门阵列；进位保存加法器；流水线结构

中图分类号：TN918.4　　**文献标志码：**A　　**文章编号：**1009-8054(2022)08-0002-10

Research on Key Technologies for High-Performance Hardware Implementation of ZUC Algorithm

ZHANG Ziang, CHEN Zhaohui

（School of Cryptography, University of Chinese Academy of Sciences, Beijing 100049, China）

Abstract: ZUC is an international encryption standard for LTE networks designed independently by China. In order to meet the server's demand for real-time high-speed encrypted data transmission and backup, combined with the structure and features of related algorithms, two efficient FPGA-based implementation schemes of ZUC are proposed, which adopt optimized bit-level modular addition structure and parallel pipeline architecture. The experiments culminate in the simulation and implementation of the design on a XILINX Kintex-7 FPGA platform and the evaluation of its runtime performance and area consumption results, with experimental results running up to 6.4 Gbit/s throughputs.

Key words: ZUC; FPGA; CSA; pipeline architecture

* 收稿日期：2022-04-22；修回日期：2022-07-29　Received date: 2022-04-22; Revised date: 2022-07-29

0 引 言

祖冲之序列密码算法（ZUC）是我国国产的序列密码算法，现已成为ISO/IEC国际标准。通过分析发现，在增大ZUC硬件实现操作频率同时减少产生密钥字的时钟周期数的情况下，其实际运行的吞吐量能得到显著的提升。基于此，在本文对ZUC的硬件实现设计中，将着重优化线性反馈移位寄存器（Linear Feedback Shift Register，LFSR）的更新过程耗时和读取S盒的面积消耗。

针对LFSR更新路径的优化设计，郭泓键等人[1]提出了使用“循环左移”运算替代“2的幂指数乘法”运算的设计方案，这一思路为本文提供了基础优化策略。在第十一届国际信息与通信安全会议上，Wang等人[2]提出了实现ZUC在现场可编程逻辑门阵列（Field Progammable Gate Array，FPGA）器件上运行的“四级流水线”结构，其设计使用进位保存加法器（Carry Save Adder，CSA）树结构来实现LFSR的长更新过程，但因缺失了LFSR的初始化过程，故该方案并不适用于实际的应用。在中国密码学会2013年密码芯片学术会议上，Liu等人[3]提出了一种高吞吐量的“混合二级流水线”结构的设计方案，该方案考虑了LFSR更新过程中初始化阶段的运算，并采用三输入CSA来处理数值的连续加法运算。在INDOCRYPT 2011上，Gupta等人[4]提出了“THREE-ZUC”设计结构，并生成了一个扩展后的版本[5]，但该方案的实际运行结果并不理想。

此外，在硬件上进行ZUC实现时，S盒运算过程也是影响算法运行吞吐率的重要因素之一。在以往实现的设计中，郭泓键等人[1]提供了通过查表（Look Up Table，LUT）方式快速实现S盒变换的思路。郁宁亚等人[6]提出了使用只读存储器（Read-Only Memory，ROM）对S盒中固定数值进行存储的方案，该思路也被应用于本文的设计中。

本文首先完善了“四级流水线”[2]结构中LFSR的初始化过程，并在此基础上提出了一种改进型的“五级流水线”结构方案，该结构在理论上能达到较高的吞吐量。

1 ZUC在FPGA上的实现

ZUC在逻辑上采用3层结构设计：线性反馈移位寄存器LFSR、比特重组BR和非线性函数F，其结构如图1所示，本文基于Verilog HDL对硬件器件、信号及时序可直接操作的特性对其硬件实现进行编码设计。

根据算法规范[7]，可知式（1）是ZUC算法系统运行时最为耗时的运算路径，后续本文将详细阐述对该过程的设计优化策略。同时，针对非线性函数过程中S盒读取操作的设计，本文通过重用一半S盒存储资源的消耗面积的方式来进行优化。另外，文中的设计均使用ROM对S盒固定数据值进行存储，以加快在非线性函数过程运行时寄存器单元R_1和R_2的更新频率。

$$2^{15}S_{15}+2^{17}S_{13}+2^{21}S_{10}+2^{20}S_4+(1+2^8)S_0 \bmod(2^{31}-1) \quad (1)$$

1.1 基础：四级流水线结构

为了实现ZUC的高吞吐量运行，文献[2]中设计了一种四级流水线结构，如图2所示。因其在硬件实现时关键路径只包含一个加法运算和一个模$2^{31}-1$运算，所以该设计能够实现较高

的吞吐量。但由于在初始化阶段关键路径中包含了一个额外用于计算长模加运算结果值和外部输入 U 的模加结果 S_{16} 的 32 比特加法器，且 U 和 S_{16} 必须是在连续的时钟周期中计算，这就限制了该设计的流水线结构不能达到“包含初始化阶段计算过程”和“限制流水线关键路径为一个两输入模加运算”两个目标的共存，故该方案不灵活。

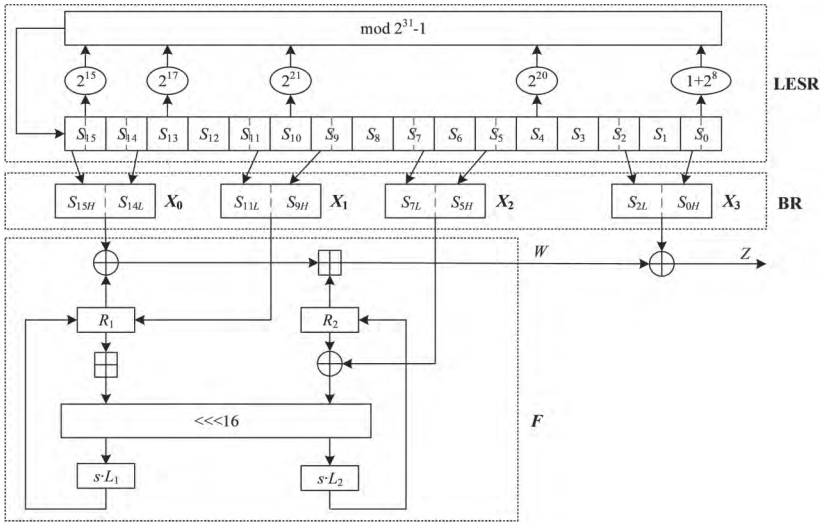

图 1　ZUC 算法逻辑结构

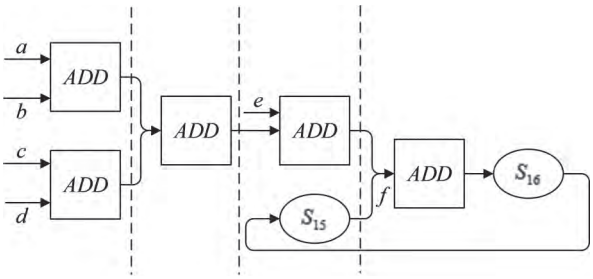

图 2　仅包含工作阶段的四级流水线结构

1.2　贡献点：完善的四级流水线

本节设计了一个完整的四级流水线结构以实现在一个时钟周期内完成 LFSR 关键路径和密钥字生成的运算。

1.2.1　算法体系结构

该算法将 LFSR 过程的关键路径缩减为一个三输入加法并模 $2^{31}-1$ 运算，且需要添加额外的寄存器资源来存储每一级流水线 $(x+y)$ 或 $(x+y+z)$ 的输出数据值。同时，针对最后一级的三输入运算，本节采用优化结构 CSA 来编程实现，这部分内容在下文详细叙述。如图 3 展示了完整四级流水线的体系结构。

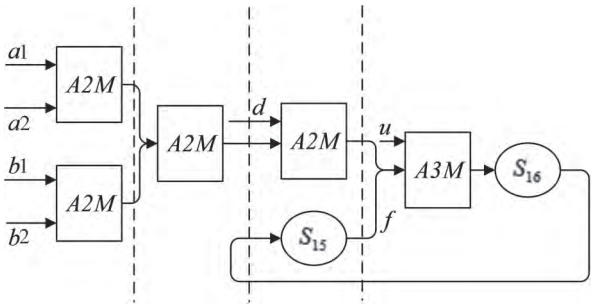

图 3　完整四级流水线结构

1.2.2　流水线过程分析

在该流水线结构初始化阶段，LFSR 中的 16 个寄存器右移更新操作将持续 4 个时钟周期不执行，之后进入算法执行过程。如表 1 所示，为完整四级流水线结构的运算过程。

表 1　完整四级流水线结构的运算过程

时钟周期	过程 A	过程 B	过程 C	过程 D	过程 E
1	$A2M(S_0, 2^8 S_0)$	$A2M(2^{20} S_4, 2^{21} S_{10})$	—	—	—
2	$A2M(S_1, 2^8 S_1)$	$A2M(2^{20} S_5, 2^{21} S_{11})$	$A2M(A_1, B_1)$	—	—
3	$A2M(S_2, 2^8 S_2)$	$A2M(2^{20} S_6, 2^{21} S_{12})$	$A2M(A_2, B_2)$	$A2M(C_2, 2^{17} S_{13})$	—
4	$A2M(S_3, 2^8 S_3)$	$A2M(2^{20} S_7, 2^{21} S_{13})$	$A2M(A_3, B_3)$	$A2M(C_3, 2^{17} S_{14})$	$A3M(D_3, 2^{15} S_{15}, U)$
5	$A2M(S_3, 2^8 S_3)$	$A2M(2^{20} S_7, 2^{21} S_{13})$	$A2M(A_4, B_4)$	$A2M(C_3, 2^{17} S_{14})$	$A3M(D_4, 2^{15} S_{15}, U)$

在流水线初始化过程的第一个时钟周期中，运算过程 A 和 B 将并行计算。其余的过程在这个时钟周期中不工作，它们均处于等待状态。

在第二个时钟周期，过程 A 和过程 B 都将进入到第二轮的计算，相当于提前一个周期计算结果值，并且这个周期 LFSR 不执行右移更新操作。同时，过程 C 将对上一个时钟周期中过程 A 和过程 B 产生的结果值进行运算，其余的运算过程仍处于等待状态。

时钟脉冲到第三个周期时，过程 A、过程 B、过程 C 都将在 LFSR 不更新的情况下进行运算，且过程 D 获取到上一个周期过程 C 产生的结果值并进行运算。而过程 E 的运算处于等待状态。

第四个时钟周期与第三个时钟周期的算法执行过程相似，但过程 A、过程 B、过程 C 和过程 D 继续运算并把结果值带入到下一个计算回合。即在这一个时钟周期是表中的过程 E 的第一次运算过程，并最终得到结果值 S_{16}，即下一轮运算时所需要的 LFSR 中的值 S_{15}。在这一个时钟周期，需要注意对运算输入 31 比特数据 U 的选通。

在第五个时钟周期，需要移动 S_{16} 赋值给 S_{15}，然后按照算法逻辑结构中的布局 LFSR 进行右移操作以使其余的寄存器单元更新。

此时，该流水线结构的初始化过程完成，整个算法系统将进入实际运行阶段，且 LFSR 的寄存器单元在之后的每个时钟周期都进行右移更新操作，其完整的算法伪代码如算法 1 所示。

算法 1：四级流水线算法

定义：

S_{16}：LFSR 过程中的下阶段的 S_{15} 更新值

U：LFSR 过程的输入值

$LFSR$[16]：LFSR 的寄存器单元变量

a_out、b_out、c_out、d_out：流水线每一级的输出

CNT：ZUC 算法运行的时钟计数器

逻辑：

```
1    CNT = 0;
2    while CNT ≥ 0 do
         /* 选取计算输入 */
```

```
3            if CNT == 1 then
4                In(LFSR[0], LFSR[4], LFSR[10]) ;
5            end
6            else if CNT == 2 then
7                In(LFSR[1], LFSR[5], LFSR[11]) ;
8            end
9            else if CNT == 3 then
10               In(LFSR[2], LFSR[6], LFSR[12], LFSR[13]) ;
11           end
12           else if CNT ≥ 4 then
13               In(LFSR[3], LFSR[7], LFSR[13], LFSR[14], LFSR[15]) ;
14           end
             /* 计算过程 */
15           when posedge CLK do
16               a_out = Add2Mod_00(a1,a2); // a1、a2 为 In() 输入的变量选择值
17               b_out = Add2Mod_01(b1,b2); // b1、b2 为 In() 输入的变量选择值
18               c_out = Add2Mod_02(a_out,b_out);
19               d_out = Add2Mod_03(c_out,d); // d 为 In() 输入的变量选择值
20               S15 = Add3Mod(d_out,LFSR[15],U);
21           end
22           CNT ++ ;
23       end
```

1.2.3 具体优化点

在以上的算法中，“$Add2Mod()$”代表两输入的模加运算，本节对其在硬件上的实现进行了优化，以缩短模加运算的耗时。这里将两输入加法运算$(A+B)$的进位设置为$Carry$。之后，第一部分操作是计算“$A+B+1$”，相当于提前计算进位为 1 时的模运算结果值；第二部分操作是直接计算“$A+B$”，把这一步产生的进位值赋值给$Carry$。以上两部分操作并行计算、互不干扰，最后通过对第二部分产生的进位值进行选通来确定最后模加运算的结果值。该方法的体系结构如图 4 所示。

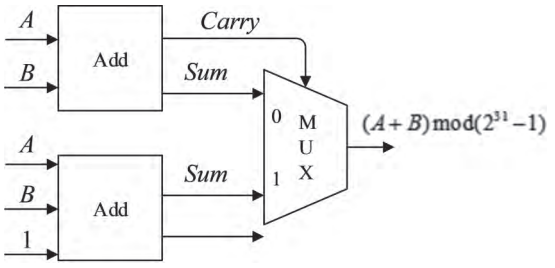

图 4　优化后的两输入模加运算结构

在利用硬件实现上述算法中的三输入模加运算“$Add3Mod()$”时，普通的运算方式会消耗大量的资源且不能达到较好的耗时要求。因此，本节使用如图 5 所示的 CSA 结构来进行这

部分运算。

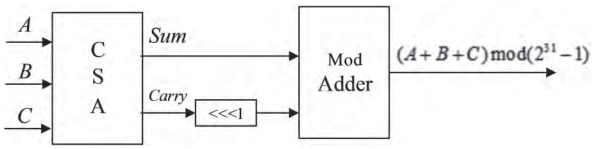

图 5　三输入模加运算结构

CSA 的具体逻辑过程为：当计算 $(A+B+C)\ \mathrm{mod}(2^{31}-1)$ 时，首先产生 3 个数据求和的进位为 $Carry$，之后计算三数相加的本位保留值 Sum，最后利用进位传播加法器（Carry Propagate Adder，CPA）结合公式计算模运算的结果值。如式（2）所示。

$$\begin{cases} Carry = A\&B \mid A\&C \mid B\&C \\ Sum = A \oplus B \oplus C \\ Carry \times 2 + Sum = A + B + C \\ Carry \times 2(\mathrm{mod}(2^{31}-1)) = Carry <<< 1 \\ Out = (Sum + (Carry <<< 1)) \bmod (2^{31}-1) \end{cases} \quad (2)$$

式中：$Carry$ 为 A，B，C 3 个数相加的进位；Sum 为 A，B，C 3 个数相加的本位保留值；Out 为计算 $(A+B+C)\bmod(2^{31}-1)$ 得到的结果值。

综合以上对算法中的加法运算和模运算的优化设计，该设计最终使 LFSR 的关键路径耗时缩短为一个三输入 32 比特加法器和一个两输入 32 比特加法器的延迟。ZUC 算法系统的关键路径则是在此基础上增加一个 32 比特加法器的延迟，这是因为算法系统中非线性函数过程的 32 比特输出 W 影响着 LFSR 初始化阶段的输入 U，故 W 的计算过程和完整的四级流水线结构最后一级的运算必然是串行的，这就严重影响了整个算法运行的吞吐量，下文中提出了针对这一部分耗时进行优化的具体方案。

1.3　创新点：改进的五级流水线

1.3.1　算法体系结构

由于 LFSR 在初始化阶段的路径包含了外部输入数据的运算，比其在工作阶段的耗时更大，因此本节提出了一种在“四级流水线结构”基础上的改进型流水线结构。如图 6 展示了该五级流水线的体系结构。

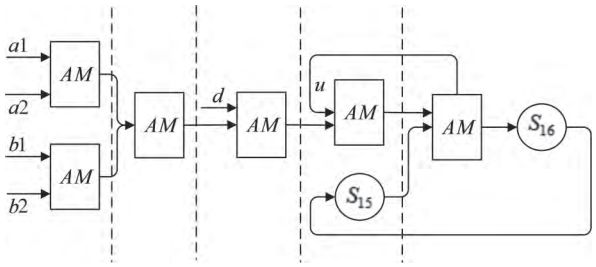

图 6　五级流水线结构

1.3.2　流水线过程分析

在这种体系结构中，LFSR 无论是在初始化模式还是在工作模式都是每个时钟周期更新一次，其算法逻辑如表 2 所示。

表 2　五级流水线结构的运算过程

时钟周期	过程 A	过程 B	过程 C	过程 D	过程 E	过程 F
1	$AM(S_0, 2^8 S_0)$	$AM(2^{20} S_4, 2^{21} S_{10})$	—	—	—	—
2	$AM(S_1, 2^8 S_1)$	$AM(2^{20} S_5, 2^{21} S_{11})$	$AM(A_1, B_1)$	—	—	—
3	$AM(S_2, 2^8 S_2)$	$AM(2^{20} S_6, 2^{21} S_{12})$	$AM(A_2, B_2)$	$AM(C_2, 2^{17} S_{13})$	—	—
4	$AM(S_3, 2^8 S_3)$	$AM(2^{20} S_7, 2^{21} S_{13})$	$AM(A_3, B_3)$	$AM(C_3, 2^{17} S_{14})$	$AM(D_3, U)$	—
5	$AM(S_4, 2^8 S_3)$	$AM(2^{20} S_8, 2^{21} S_{13})$	$AM(A_4, B_4)$	$AM(C_4, 2^{17} S_{15})$	$AM(D_4, U)$	$AM(E_4, 2^{15} S_{15})$
6	$AM(S_4, 2^8 S_3)$	$AM(2^{20} S_8, 2^{21} S_{13})$	$AM(A_5, B_5)$	$AM(C_5, 2^{17} S_{15})$	$AM(D_5, U)$	$AM(E_5, 2^{15} S_{15})$

相对于上一节提出的四级流水线体系结构，本节的方案仅仅是把 LFSR 过程外部输入 U 的模加运算作为一级流水线，并且该运算位于五级流水线计算下一轮 S_{15} 值的操作之前。在四级流水线体系结构中，最后一级过程的 W 运算和 S_{16} 运算是串行的，这意味着必须先有 W 的结果值而后才能计算 S_{16} 的值。而在本节提出的五级流水线结构中，下一轮需要的 W 值将提前计算，这个计算过程的前提输入是该时钟周期在线网器件上的经流水线最后一级计算而得的 S_{16} 值（将在下一个时钟节拍上升沿赋值到寄存器单元中）。

1.3.3 具体优化点

在表 2 描述的流水线算法流程中，每一级都是一个两输入的模加法运算。针对这一运算过程，使用如图 7 所示的体系结构进行运算。该结构由两个 32 比特加法器直接相连接实现加法运算、模 $2^{31}-1$ 运算，这种方案实现的运算耗时是两个 32 比特加法器的延迟。该体系结构是处于“控制流水线关键路径为一个两输入模加运算”和“在完整四级流水线基础上改进最后的三输入模加运算”两种方案折中的优化，并在实际的硬件实现上能够得到比 1.2 节提出的完整四级流水线体系结构更好的性能结果。

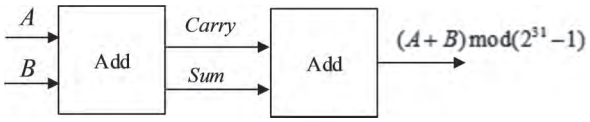

图 7　两输入模加运算结构

另外，在编码实现时，使用如算法 2 所示的逻辑以控制整个算法的时钟节拍的统计与定位，这样方便控制 ZUC 的初始化阶段和工作阶段的转换。

算法 2：时序控制算法

定义：

CLK：ZUC 算法运行的时钟

IN_EN：外部输入的使能信号

IN_EN_MAIN：算法系统中存储的使能信号值（慢一个周期）

TIMER_COUNT：算法系统中的时钟计数器

逻辑：

```
1   while posedge CLK do
        /* 第一次计算 */
2       if !IN_EN_MAIN && IN_EN then
3           TIMER_COUNT = 16' d1;
4       end
5       else if IN_EN_MAIN && IN_EN then
6           TIMER_COUNT = TIMER_COUNT + 16' d1;
7       end
8       else then
9           TIMER_COUNT = 16' d0;
10      end
11  end
```

2 实验结果

2.1 仿真及 FPGA 测试结果

在设计具体实现时，使用硬件描述性语言 Verilog HDL 来编程，如图 8 展示了利用 Vivado 在 XILINX-AX7325 开发板上布线后时序仿真的结果。

将工程源程序综合、布线、生成比特文件并下载到 FPGA 上，运行后可看到如图 9 中的调试结果。

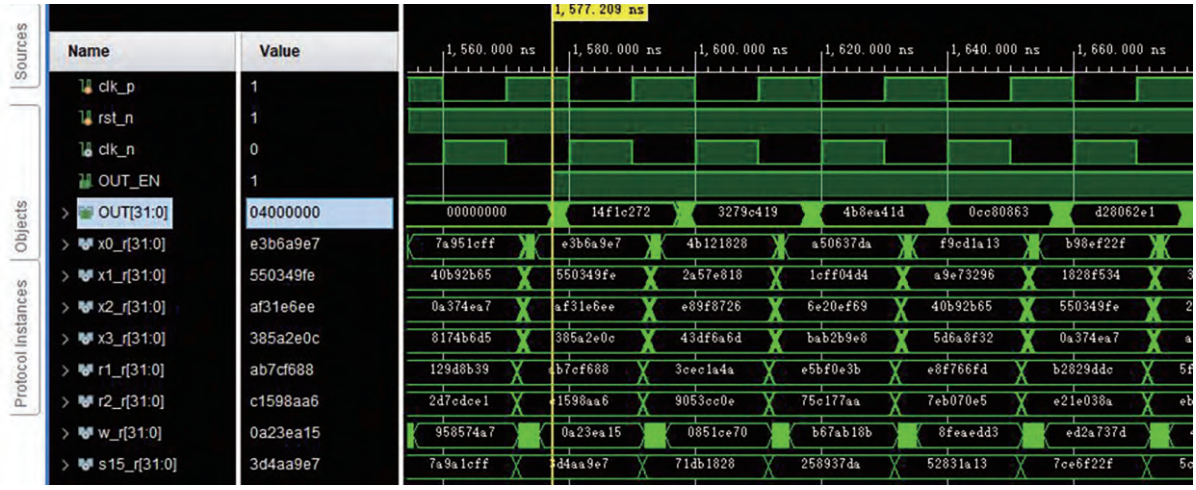

图 8　布线后时序仿真

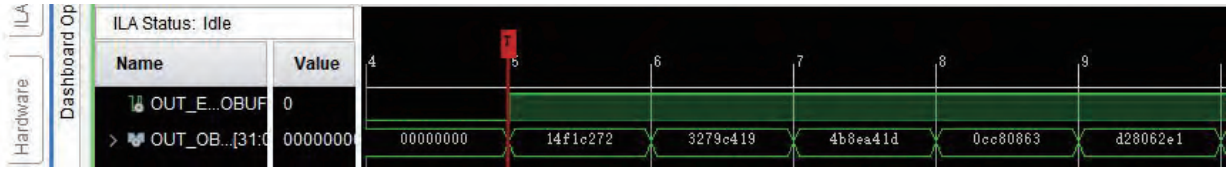

图 9　上板后调试结果

这里对本文提出的两种方案的实际运行结果进行记录及分析。评估算法系统的运行效率时，主要考虑吞吐率和芯片面积两个指标 [8]。其中，吞吐量的计算方式为：吞吐量 =（输入信息数据块比特数 × 算法系统最大工作频率）÷ 算法系统处理一个数据分块需要消耗的时钟周期数 [9]。

如表 3 所示，祖冲之序列密码算法能够根据在硬件上不同的实现来满足其对吞吐量和资源消耗的调整需求，这符合对现代密码算法的设计要求。实验使用 Vivado 开发工具对源程序进行综合布线，表 4、表 5 分别显示了四级流水线和五级流水线结构在进行手动物理布线优化后的资源面积消耗情况。

表 3　ZUC 理论运行结果

体系结构	理论消耗资源面积 /slices	理论工作频率 /MHz	理论吞吐量 /（Gbit/s）
四级流水线	355	221.93	7.10
五级流水线	360	241.60	7.73

表 4　四级流水线布线后资源消耗结果

类型	可使用数量	实际使用数量	占比 /%
Slice LUTS	1 440	1 254	87.08
LUT as Logic	1 440	1 254	87.08
LUT as Memory	360	0	0.00
Slice Registers	2 880	939	32.60
Register as Filp Flop	2 880	782	27.15
Registers as Latch	2 880	157	5.45
F7 Muxes	720	0	0.00
F8 Muxes	360	0	0.00
Slice	360	358	99.44
SLICEL	270	268	99.26
SLICEM	90	90	100.00
Unique Control Sets	360	10	2.78
BUFGCTRL	0	3	–NA–

表 5　五级流水线布线后资源消耗结果

类型	可使用数量	实际使用数量	占比 /%
Slice LUTS	1 440	1 126	78.19
LUT as Logic	1 440	1 126	78.19
LUT as Memory	360	0	0.00
Slice Registers	2 880	1 062	36.88
Register as Filp Flop	2 880	808	28.16
Registers as Latch	2 880	254	8.82
F7 Muxes	720	0	0.00
F8 Muxes	360	0	0.00
Slice	360	360	100.00
SLICEL	270	270	100.00
SLICEM	90	90	100.00
Unique Control Sets	360	10	2.78
BUFGCTRL	0	1	–NA–

对比本文提出的四级流水线和五级流水线两种体系结构，发现后者在前者的基础上仅仅进行了运算结构上的改变，就可以在增大极小部分资源消耗面积的情况下显著提高算法系统的运算吞吐量。

2.2　算法系统关键路径对比

本文提出的两种方案与以往设计的关键路径对比如表 6 所示。

表 6　不同体系结构的关键路径对比

体系结构	关键路径
完整四级流水线	32 bit Adder → Three CSA → 32 bit Adder → 32 bit Adder
五级流水线	Less Than 32 bit Adder → 32 bit Adder → 32 bit Adder
混合二级流水线	On Level CSA → 31 bit Adder → 2 × a multiplexer
THREE-ZUC	4 × 31 bit Adder → a multiplexer
不完整四级流水线	32 bit Adder → 32 bit Adder → 32 bit Adder

3　结　语

为满足服务器对数据实时高速加密传输、备份的需求，本文提出了两种基于 ZUC 在 FPGA 硬件平台上的高吞吐量设计方案，其理论工作频率分别为 7.10 Gbit/s 和 7.73 Gbit/s。但受硬件平台晶振产生的时钟频率大小的限制，以上结构在测试所用的 XILINX-AX7325 平台上实际最大吞吐量为 6.4 Gbit/s。通过分析文中的算法系统，可以发现其运行时的关键路径还不是最短的，因此，之后将在技术设计方面继续深入探寻具备更短关键路径的体系结构。

参考文献：

[1] 郭泓键 , 董秀则 , 高献伟 . 基于 FPGA 的祖冲之算法硬件实现 [J]. 计算机工程 ,2014,40(8):268-272.

[2] WANG L,JING J W,LIU Z B,et al.Evaluating Optimized Implementations of Stream Cipher ZUC Algorithm on FPGA[M]//Information and Communications Security. Heidelberg: Springer Berlin Heidelberg,2011:202-215.

[3] LIU Z B,GAO N,JING J W,et al.HPAZ: a High-throughput Pipeline Architecture of ZUC in Hardware[C]// 中国密码学会 2013 年密码芯片学术会议 ,2013.

[4] SEN GUPTA S,CHATTOPADHYAY A,KHALID A.HiPAcc-LTE: An Integrated High Performance Accelerator for 3GPP LTE Stream Ciphers[C]// International Conference on Cryptology in India,2011.

[5] SEN GUPTA S,CHATTOPADHYAY A,KHALID A.Designing Integrated Accelerator for Stream Ciphers with Structural Similarities[J].Cryptography and Communications,2013,5(1):19-47.

[6] 郁宁亚 , 朱宇霞 . 基于 FPGA 的祖冲之序列密码算法实现 [J]. 信息技术 ,2015,39(9):125-129.

[7] 国家质量监督检验检疫总局 , 中国国家标准化管理委员会 . 信息安全技术 祖冲之序列密码算法 第 1 部分：算法描述：GB/T 33133.1—2016[S]. 北京：中国标准出版社 ,2016.

[8] 江丽娜 , 高能 , 马原 , 等 . 祖冲之序列密码算法 IP 核的设计与实现 [J].2012(8):219-222.

[9] 李歌 , 陶琳 , 高献伟 . 基于 FPGA 的祖冲之算法研究与实现 [J]. 北京电子科技学院学报 ,2012,20(4):13-18.

作者简介：

张子昂（1999—），男，硕士研究生，主要研究方向为信息安全；

陈朝晖（1995—），男，博士研究生，主要研究方向为密码工程。