HPAZ: a High-throughput Pipeline Architecture of ZUC in Hardware

Zongbin Liu[1], Neng Gao[1], Jiwu Jing[1], and Peng Liu[2]

[1]Institute of Information Engineering, Chinese Academy of Sciences, China.
{zbliu,gaoneng,jing}@lois.cn
[2]The Pennsylvania State University, University Park, USA
pliu@ist.psu.edu

Abstract. In this paper, we propose a high-throughput pipeline architecture of the stream cipher ZUC which has been included in the security portfolio of 3GPP LTE-Advanced. In the literature, the schema with the highest throughput only implements the working stage of ZUC. The schemas which implement ZUC completely can only achieve a much lower throughput, since a self-feedback loop in the critical path significantly reduces operating frequency. In this paper we design a mixed two-stage pipeline architecture which not only completely implements ZUC but also significantly raises the throughput. We have implemented our architecture on FPGA and ASIC. On FPGA platform, the new architecture increases the throughput by 45%, compared with the latest work, and particularly the new architecture also saves nearly 12% of hardware resource. On 65nm ASIC technology, the throughput of the new design can up to 80Gbps, which is 2.7 times faster than the fastest one in the literature, in particularly, it also saves at least 40% of hardware resource. In addition to the academic design, compared with the fastest commercial design, the new architecture doubles the throughput of that. To the best of our knowledge, this evaluation result is so far the best outcome. It can be assumed that hardware implementations of ZUC following our architecture will fit in future LTE equipments better.

Key words: Stream Cipher, FPGA, Hardware evaluation, ZUC

1 Introduction

ZUC [1–4] is a word-oriented stream cipher and consists of two stages (the initialization stage and the working stage). ZUC has three logical layers as seen in Fig.1. The top layer is a linear feedback shift register (LFSR) with 16 cells, the middle layer is the bit-reorganization, and the bottom layer is a nonlinear function F. In the initial stage, LFSR is constructed using a 128 bit key, a 128 bit IV and a 240-bit long constant string, and during the first 32 iterations, the output of the FSM is added to the feedback loop for LFSR update [5]. After the first 32 iterations, ZUC moves into the working stage and outputs 32 bits of key per iteration.

Since the throughput of hardware implementation of ZUC is determined by the ratio of operating frequency to the number of clock cycles to generate per 32-bit key, we use T to denote the number of clock cycles. In order to acquire a high throughput, we should diminish T or increase the operating frequency. In the contemporary proposed works, $T = 1$ is quite often applied in order to achieve a high throughput, which means that the LFSR needs updating per clock cycle to realize an output of 32-bit key every clock cycle.

Operating frequency is determined by the critical path in ZUC. The critical path of the ZUC in hardware implementation is determined by the updating routine of the LFSR. The updating routine of LFSR employs a series of modulo $2^{31} - 1$ multiplications and additions, while the addition module is a time-consuming and resource-consuming component, due to which this data path is much longer than others.

In the literature, many works try to shorten this path to increase the throughput of the ZUC. The highest one [6] only implemented the working stage of ZUC, which is not applicable in practical applications. The scheme proposed in INDOCRYPT2011 by Gupta et al [5] implemented the two stages of ZUC, however, the self-feedback loop in their critical path significantly reduces operating frequency of their architecture.

As the critical path in the initialization stage is longer than that in the working stage, the throughput of the previous works [7–9] which includes both stages are much lower than that work [6] which includes the working stage only.

Motivation.

We consider the following problem in hardware implementations of ZUC.

As ZUC has a self-feedback loop in the critical path, it is harder to increase the key stream throughput by means of traditional pipeline, since the traditional pipeline method not only increase operating frequency but also increases the number of clock cycles required when generating each 32-bit cipher words. S. Gupta et al. [9] first raised this problem but the puzzle remains unsolved. And the later works [5] [7] did not solve this problem either, which demonstrably threw solid proof upon the impossibility of increasing ZUC throughput via traditional pipeline method at present.

In short, a lot of effort is being spent on improving this weakness, the efficient and effective method has yet to be developed in ***past three years***.

Contribution.

Our primary contribution is that we propose a novel mixed two-stage pipeline architecture of ZUC to considerably increase the throughput of ZUC in hardware. On FPGA platform, the new architecture increases the throughput by 45%, compared with the latest work [10], and saves nearly 12% of hardware resource. On ASIC platform, our new architecture provides at least *twice* the throughput compared to any existing academic designs and commercial designs.

To achieve a high-throughout ZUC hardware implementation, the LFSR should be updated per clock cycle for the purpose of producing 32-bit key works in a higher operating frequency. Although the long path is undertaken in the initialization stage only, yet it slows down the operating frequency in the entire ZUC implementation. It seems unworthy to lose the greater for the less.

Based on this observation, in our mixed two-stage pipeline architecture, it actually has two modes, namely half-pipeline mode and full-pipeline mode. In the initialization stage of ZUC, our architecture works in half-pipeline mode, in which the LFSR is updated every two clock cycles, so the long data path in the initialization stage can be divided into two small sub-blocks and operating frequency is boosted. After 64 clock cycles when the initialization stage ends in our architecture, the pipeline architecture will be transferred into the full-pipeline mode. During this mode, the LFSR updates per clock cycle and at the same time 32-bit key stream is generated in each clock cycle, and moreover, the full-pipeline architecture is totally different from that of the previous works.

In particular, in our new architecture, we do not use any optimization techniques which are specific to a certain platform. Thus our architecture can perform well on different platforms (FPGA and ASIC) without any changes. If someone wants to further improve the throughput, the specific optimization skills corresponding to certain platform can be used to further improve the throughput, and moreover, compared with previous works, our architecture uses less hardware resource.

In order to verify the accuracy of our mixed two-stage pipeline architecture, we have implemented our architecture in Xilinx V5 and Xilinx V6 FPGA, achieving a throughput of 7.9 Gbps with 350 slices and 11.2 Gbps with 328 slices respectively. It is deduced that our evaluation results are so far the optimal outcomes in Xilinx FPGA given a comparison of existing pipeline and non pipeline architecture properties.

We also implement the design in 65 nm ASIC platform, as the result shown, our design at least increases the throughput by 2 times compared with the academic designs and the commercial designs.

Owing to the highest throughput and smaller hardware size, the designs following our architecture will perform better in future LTE equipments.

2 Preliminaries: ZUC Algorithm

The new stream cipher ZUC is a word-oriented stream cipher [1]. It takes a 128-bit initial key and a 128-bit initial vector as input, and outputs a key stream of 32-bit words. The execution of ZUC has two stages: initialization stage and working stage. In the first stage, a key initialization

is performed, i.e. the cipher is clocked without producing output. The second stage is a working stage. The algorithm produces a 32-bit word of output in per loop of the working stage.

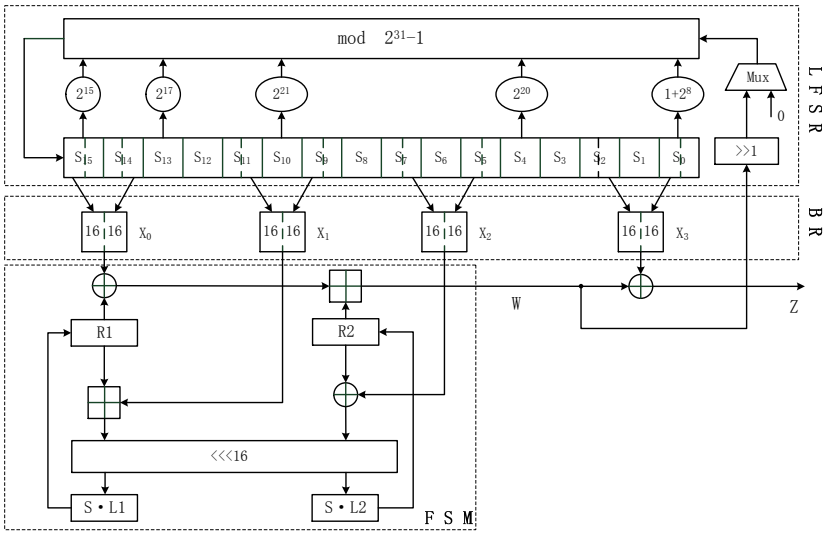

Fig. 1. The architecture of ZUC

2.1 The linear feedback shift register (LFSR)

The LFSR has 16 31-bit cells $(S_0, S_1, \cdots, S_{15})$. Each cell $S_i (0 \leq i \leq 15)$ is registered to take values from the following set: $\{1, 2, 3, \ldots, 2^{31} - 1\}$. The LFSR has 2 modes of operations: the initialization and the working mode. In the initialization mode, the LFSR receives a 31-bit input word u, which is obtained by removing the rightmost bit from the XOR of the 32-bit output W of the nonlinear function F and the output X_3 of the bit-reorganization, i.e.,$u = (W \oplus X_3) >> 1$. More specifically, the initialization mode works as follows:

LFSRWithInitialisationMode(u)
{

1. $v = \{2^{15}S_{15} + 2^{17}S_{13} + 2^{21}S_{10} + 2^{20}S_4 + (1 + 2^8)S_0\} \pmod{2^{31} - 1}$
2. $S_{16} = v + u \pmod{2^{31} - 1}$
3. $if\ S_{16} = 0$ then set $S_{16} = 2^{31} - 1$
4. $(S_1, S_2, \ldots, S_{15}, S_{16}) \rightarrow (S_0, S_1, \ldots, S_{14}, S_{15})$

}

+ denotes the addition of two integers, *mod* denotes the modulo operation of integers. In the working mode, the LFSR does not receive any input, the LFSR works independently with other part of ZUC. The LFSR update data path of working process is shorter than that of in the initiation stage. More specifically, the working mode works as follows:

LFSRWithWorkMode()
{

1. $S_{16} = \{2^{15}S_{15} + 2^{17}S_{13} + 2^{21}S_{10} + 2^{20}S_4 + (1 + 2^8)S_0\} \pmod{2^{31} - 1}$
2. $if\ S_{16} = 0$ then set $S_{16} = 2^{31} - 1$
3. $(S_1, S_2, \ldots, S_{15}, S_{16}) \rightarrow (S_0, S_1, \ldots, S_{14}, S_{15})$

}

2.2 The Bit-reorganization

The middle layer of the algorithm is the Bit-reorganization. It extracts 128 bits from the state of the LFSR and forms 4 32-bits words, X_0,X_1,X_2, and X_3. $X_0 = S_{15H} \parallel S_{14L}$, $X_1 = S_{11L} \parallel S_{9H}$, $X_2 = S_{7L} \parallel S_{5H}$, $X_3 = S_{2L} \parallel S_{0H}$.

S_{15H} denotes the leftmost 16 bits of integer S_{15}, S_{14L} denotes the rightmost 16 bits of integer S_{14}, $S_{15H} \parallel S_{14L}$, denotes the concatenation of strings S_{15H} and S_{14L}, more detail specification can be found in [1].

2.3 The nonlinear function F

The nonlinear function F has two 32-bit width memory cells R_1 and R_2. The input of the nonlinear function is the X_0, X_1, X_2, which are the output of the bit-reorganization step, the nonlinear function F outputs a 32-bit word W.

The nonlinear function(X_0, X_1, X_2, X_3)
{

1. $W = (X_0 \oplus R_1) \boxplus R_2$
2. $W_1 = R_1 \boxplus X_1$
3. $W_2 = R_2 \boxplus X_2$
4. $R_1 = S(L_1(W_{1L} \parallel W_{2H}))$
5. $R_2 = S(L_2(W_{2L} \parallel W_{1H}))$

}

$\parallel$ denotes the concatenation of strings. $\oplus$ denotes the bit-wise exclusive-OR operation of integers. $\boxplus$ denotes the modulo 2^{32} addition . In item 4 and 5, S is a 32×32 S-box which is composed of four 8×8 SBoxes, and L_1 and L_2 are linear transformations, which are defined as equation 1, 2:

$$L_1(X) = X \oplus (X \lll 2) \oplus (X \lll 10) \oplus (X \lll 18) \oplus (X \lll 24) \tag{1}$$

$$L_2(X) = X \oplus (X \lll 8) \oplus (X \lll 14) \oplus (X \lll 22) \oplus (X \lll 30) \tag{2}$$

2.4 The execution of ZUC

Key Loading The key loading procedure will expand the initial key and the initial vector into 16 31-bit integers as the initial state of the LFSR. Let the 128-bit initial key k and the 128-bit initial vector iv be:

$$k = k_0 \| k_1 \| k_2 \| \cdots \| k_{15}$$

and

$$iv = iv_0 \| iv_1 \| iv_2 \| \cdots \| iv_{15}$$

Let D be a 240-bit long constant string composed of 16 substrings of 15 bits:

$$D = d_0 \| d_1 \| \cdots \| d_{15}$$

For $0 \leq i \leq 15$, let $s_i = k_i \| d_i \| iv_i$

The execution of ZUC The execution of ZUC has two stages: the initialization stage and the working stage. During the initialization stage, the algorithm first loads the 128-bit key, initial iv, and into the LFSR as the initial state, and set the memory cells R_1 and R_2 to be all 0. Then the cipher stays in the initialization stage in the first 32 iterations, and then moves into the working stage to generate key stream.

3 The State-of-the-Art implementation of ZUC in hardware

In this paper, we focus on designing a high-throughput architecture of ZUC. Since the throughput is determined by the critical path, here we first show the detailed description of the critical path of ZUC in hardware implementation and then review the methods adopted for shortening the critical path in the literature.

According to the description of ZUC in the above section, we can easily find that the critical path in hardware implementations of ZUC is the LFSR update operation in the initialization stage.

Compared with the same operation in the initialization stage, in the working stage, the update operation does not need an extra 32-bit addition to obtain the value of S_{16}, which is the value of S_{15} in the next round. In the working stage, the value of S_{16} can be calculated using equation (3). If this sum $S_{16} = 0$, it is replaced by $2^{31} - 1$, this step is named the check step in this paper.

$$S_{16} = \{2^{15}S_{15} + 2^{17}S_{13} + 2^{21}S_{10} + 2^{20}S_4 + (1 + 2^8)S_0\} \pmod{2^{31} - 1} \quad (3)$$

In order to achieve a high throughput, Liu et al [6] proposed a four-stage pipeline architecture of ZUC, here denoted FOUR-ZUC, which only included the working stage of ZUC. As only one modulo $2^{31} - 1$ addition is in their critical path, their design can reach to a high throughput. How does this architecture work? As the values of some cells in LFSR could be known in advance, such as the values of S_{13} in the second and third rounds, Liu et al divided this path $\{2^{15}S_{15} + 2^{17}S_{13} + 2^{21}S_{10} + 2^{20}S_4 + (1 + 2^8)S_0\} \pmod{2^{31} - 1}$ into four stages, each of which only included one modulo $2^{31} - 1$ addition as shown in Fig.2.

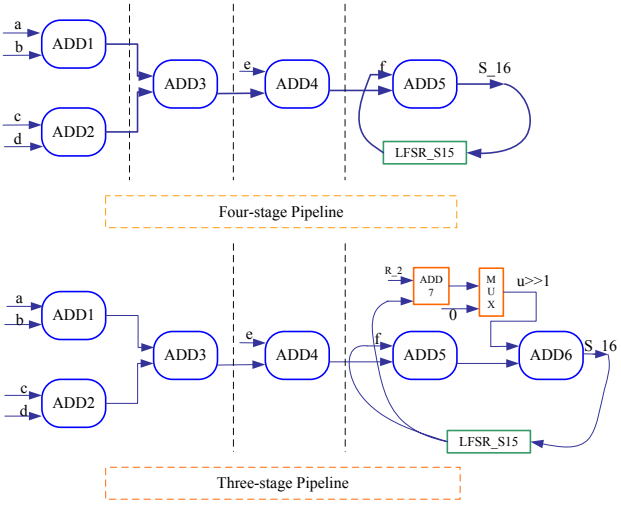

Fig. 2. Four-stage and three-stage pipeline structure of ZUC

The reason why they only implemented the working stage is that if their architecture includes the initialization stage, the long path in the initialization stage will impact the throughput of key stream. As mentioned above, the critical path in the initialization stage includes an extra 32-bit adder which is used to calculate u, however, u and S_{16} must be calculated serially in the same

clock cycle due to u derived from S_{15}. This extra 32-bit adder increases the critical path, thus reducing the throughput of key stream. Although the throughput of this architecture is high, this architecture is not flexible due to a lack of the initialization stage in hardware.

Kitsos et al [11] found above drawbacks and designed an architecture including the initialization stage. Owing to cascading three modulo $2^{31}-1$ adders in their critical path, the delay of this path was longer than that of FOUR-ZUC, thus the throughput of this work is much lower. As they did not adopt pipeline architecture which uses more hardware resource to store intermediate results, this schema can save the consumption of hardware resource.

In the most recent conference INDOCRYPT 2011, Gupta et al [9] improved the architecture of FOUR-ZUC and proposed a three-stage pipeline architecture as shown in Fig.2, here denoted THREE-ZUC. Compared with FOUR-ZUC, the advantage of THREE-ZUC is that the initialization stage of LFSR is included without much extra hardware resource. As their pipeline architecture included the initialization stage, the critical path is longer than that of FOUR-ZUC. The reason why their work can build a three-stage pipeline rather than four is that two modulo $2^{31}-1$ additions can be calculated serially in one stage of THREE-ZUC, Gupta et al need not to divide the long path into four parts. Although the architecture included the initialization stage, the throughput of this architecture was decreased by the lengthened critical path.

Gupta et al [5] proposed an extend version of their paper in INDOCRYPT 2011, but no throughput is increased in their improved architecture of ZUC, with the same reason discussed above.

Without using pipeline architecture, Wang et la [8] and Zhang et al [10] proposed architectures based on carry save adder (CSA) for implementing the ZUC algorithm on FPGA, they still cannot overcome the obstacle that the path in the initialization stage is longer than that in the working stage, despite involving carry save adder to shorten the critical path.

In short, the long path in the initialization stage greatly reduced the throughput of the key stream. Although a lot of effort is being spent on improving this weakness, the efficient and effective method has yet to be developed in past three years.

In the later sections we will propose a mixed two-stage pipeline architecture to solve this problem and this new architecture can increase the throughput drastically.

4 The Mixed two-stage pipeline architecture of ZUC

4.1 The architecture of modulo $2^{31}-1$ adder

Since the modulo $2^{31}-1$ addition shown in Fig.1 is the most time-consuming and resource-consuming component, this section first gives three methods to compute $(a+b) \pmod{2^{31}-1}$ in hardware, and then discusses the disadvantages and advantages of them, at last depicts our mixed two-stage pipeline architecture of ZUC.

The algorithm to compute $(a+b) \pmod{2^{31}-1}$ is as follows:

Let $a, b \in GF(2^{31}-1)$, the computation of $v=(a+b) \pmod{2^{31}-1}$ can be done by computing v=a+b; and then check the carry bit:

- carry $=1$, set $v=a+b+1$.
- carry $=0$, set $v=a+b$.

Method 1 This method is used in THREE-ZUC, and the architecture is shown in Fig.3. This is a direct way to implement the modulo $2^{31}-1$ addition, which concatenates two 31-bit adders directly, and the delay of this method is that of two 31-bit adders.

Method 2 Based on the observation of the long delay in Method 1, Liu et al [6] proposed this method to shorten the delay of Method 1. In order to calculate $A+B+1$, $Carry_0$ is set to 1, A, B as inputs of one adder, and $Carry_0=0$, A, B as inputs of another adder. $A+B$ and $A+B+1$ can be computed at the same time, and the last result can be selected by the carry bit of $A+B$. The delay in this method is lower than that of Method 1. The architecture of this method is shown in Fig.3.

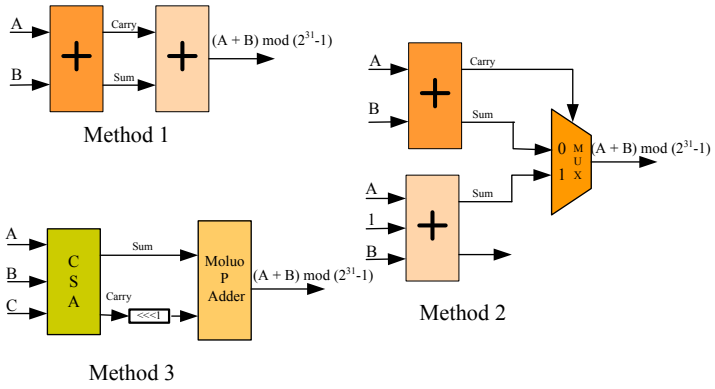

Fig. 3. The architecture of Modulo $2^{31} - 1$ Adder

Method 3 Another adder which is widely used in hardware design is carry save adder (CSA). It is simply a parallel ensemble of k full adders without any horizontal connection [12]. When adding together three or more numbers, using a CSA followed by a carry propagate adder(CPA) is faster than using two CPAs. For example, in order to calculate $(A + B + C) \pmod{2^{31} - 1}$, first use equation (4,5) produces two integer Carry and Sum, and then use a CPA to get the last result.

$$Carry = A_i \& B_i | A_i \& C_i | B_i \& C_i \tag{4}$$

$$Sum = A_i \oplus B_i \oplus C_i \tag{5}$$

$$Carry * 2 + Sum = A + B + C \tag{6}$$

$$Carry * 2 \pmod{2^{31} - 1} = Carry \lll 1 \tag{7}$$

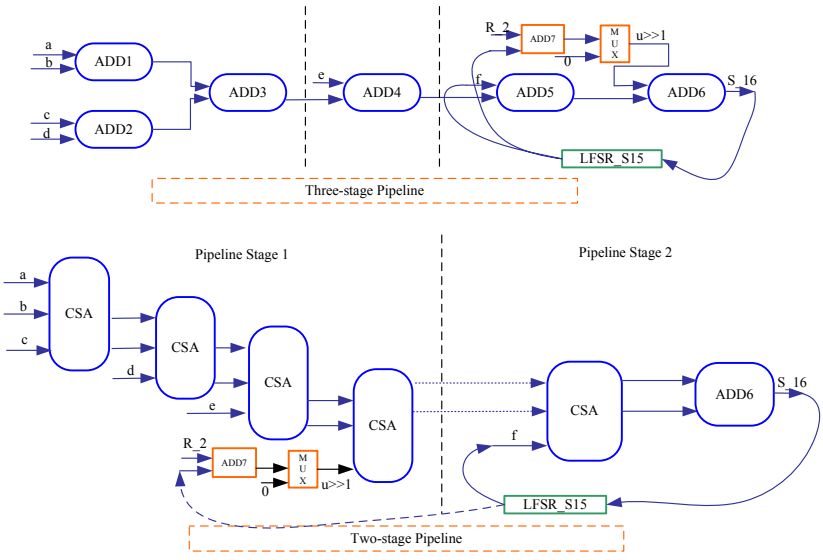

Fig. 4. Three-stage and two-stage Pipeline Structure of ZUC

It is straightforward to find that $A + B + C \pmod{2^{31} - 1}$ is equal to $(Sum + (Carry \lll 1)) \pmod{2^{31} - 1}$. Consequently, if using this CSA architecture to implement the modulo $2^{31} - 1$ addition, it will be very efficient as the total delay of this method is shorter than that of previous methods, and moreover, less hardware resource is needed in this method.

The disadvantage of this method is that the operand number of the CSA modulo $2^{31}-1$ adder must be more than three, and the last result should be gotten by Method 2 or Method 1 as shown in Fig.3.

There are many modulo $2^{31}-1$ additions in the critical path of ZUC. Based on this observation, the implementation should use as many adders following Method 3 as possible, and should not use many two-input modulo $2^{31}-1$ adders proposed by Method 1 and 2.

4.2 The Mixed Two-Stage Pipeline Architecture of ZUC

To achieve a high-throughput ZUC hardware implementation, the LFSR should be updated per clock cycle for the purpose of producing 32-bit key works in a higher operating frequency. Although the long path is undertaken in the initialization stage only, yet it slows down the operating frequency in the entire ZUC implementation. It seems unworthy to lose the greater for the less.

Based on this observation, we propose a new architecture. In this new architecture, the LFSR is updated every other clock cycle in the initialization stage, and per clock cycle in the working stage, by this means, we increase the operating frequency, thus the throughput of the key is boosted since the 32-bit key words generating per clock cycle as other architectures proposed before. In particular, the new architecture consumes less hardware resource. Next we will give the detailed information on this new architecture.

The initialization stage As discussed above, in this stage the LFSR is updated every other clock cycle, that means we can divide the original critical path into two sub-blocks, namely Pipeline Stage 1 and Pipeline Stage 2 respectively. Fig. 4 shows the structure of our architecture. From Fig. 4, the data path in the Pipeline Stage 2 is longer than in the Pipeline Stage 1, but the former is much shorter than THREE-ZUC as seen in Fig.4, it is about one fourth of the THREE-ZUC, because the modular adder using in THREE-ZUC adopts Method 1, while the Method 2 is used in our architecture.

It occurred to us that the updating job of LFSR can be promoted by pipeline construction, since S_{15} is the only necessity for Pipeline Stage 2 in the working stage, and the value of S_{15} required in Pipeline Stage 1 can be achieved by pre-computation.

Utilization of the revised method can help boost the throughout of the new design with a considerable increase compared with all the previous works.

Here we give the detailed design of Pipeline Stage 1 and Pipeline Stage 2. The architecture of Pipeline Stage 1 and 2 can be found in Fig.5 and in Fig.6 respectively.

In the Pipeline Stage 1, the first three CSA modular adders are used to calculate $(Carry_1, Sum_1)$ using equation (8). The last CSA calculates $(Carry_2, Sum_2)$ using equation (9).

$$(Carry_1, Sum_1) = \{2^{17}S_{13} + 2^{21}S_{10} + 2^{20}S_4 + (1+2^8)S_0\} \pmod{2^{31}-1} \tag{8}$$

The multiplexer in Fig.5 plays an important role in changing the working mode. In the initialization stage, $u \gg 1$ is strobed into the last CSA via the multiplexer in Fig.5, while in the working stage, 0 is strobed. In this way, when in the working stage, the multiplexer can bypass the circuit(denoted with the dotted box in Fig. 5) which special to the initialization stage. Since in the initialization stage, the LFSR is updated every other clock cycle, the expected value of S_{16} required in the initialization stage is guaranteed.

At the last part of the Pipeline Stage 1, two extra 31-bit wide registers are used to store the intermediate results $Carry_2$ andSum_2 of the last CSA. Here we do not calculate the sum of $Carry_2$ and Sum_2 directly in the Pipeline stage 1, because if we do, the path in the Pipeline Stage 1 will be increased by an extra 31-bit wide addition.

$$(Carry_2, Sum_2) = \{Carry_1 + Sum_1 + u >> 1\} \pmod{2^{31}-1} \tag{9}$$

$$(Carry_3, Sum_3) = \{Carry_2 + Sum_2 + 2^{15}S_{15}\} \pmod{2^{31}-1} \tag{10}$$

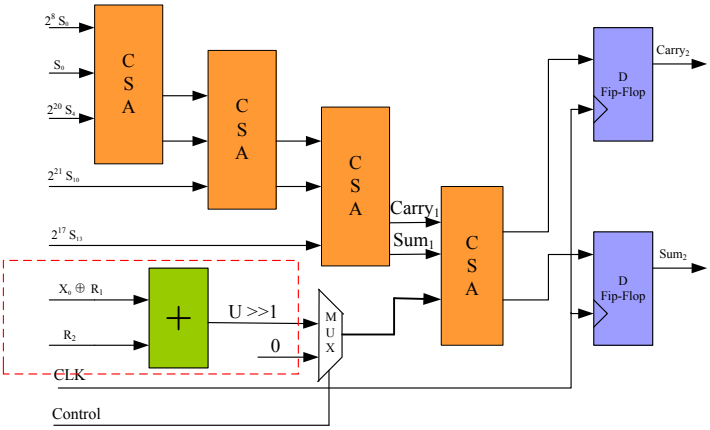

Fig. 5. The structure of pipeline stage 1

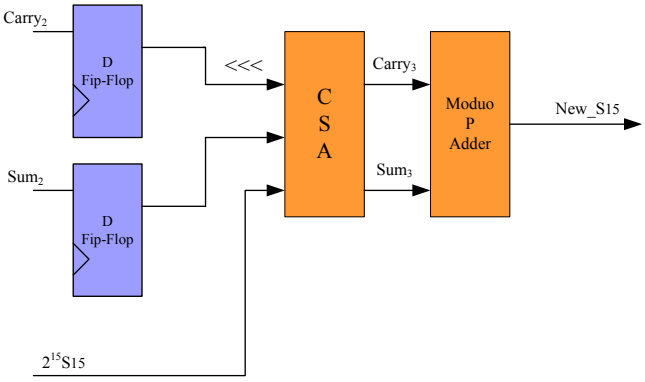

Fig. 6. The structure of pipeline stage 2

In the Pipeline Stage 2, the CSA component is used to calculate $(Carry_3, Sum_3)$ using equation (10). At the end of this stage, the modulo $2^{31}-1$ adder with Method 2 is used to derive S_{16} which is the value of S_{15} in the next iteration. When the value of S_{15} is figured out, the checking step is needed to guarantee that the value of new S_{15} is in the set $\{1, 2, \cdots, 2^{31}-1\}$. If this checking step was included in this stage, it would extend the path in the Pipeline Stage 2. However, Zhang et al [10] has proofed this step can be ignored in the hardware implementation of ZUC.

The working stage The operations of the working stage in the initial two clock cycles are different from the latter ones in that the pipeline is constructed during the first and second clock cycle. In the first clock cycle, Pipeline Stage 1 calculates $(Carry_1, Sum_1)$ using equation (8) which is the major part of S_{16} of the next iteration. At the end of the first clock cycle, all cells of the LFSR, except the fifteenth shift right to update the LFSR state, while the content of the fifteenth in the LFSR remains unchanged.

In the second clock cycle, Pipeline Stage 2 starts running and calculates the value of S_{16} using equation (10). At the same time, the major part of S_{17} , the required value of S15 after two iterations is calculated in Pipelined Stage 1 using equation (8). At the end of the second clock cycle, the value of S_{16} is simultaneously written into the fifteenth and fourteenth cells of the LFSR to update both contents, when other cells of the LFSR shift right to update the contained values. After those two clock cycles, the pipeline is constructed and begins to work, outputting a 32-bit key each clock cycle.

5 Evaluation and Analysis

5.1 Data path of ZUC

Based on the description of ZUC, there are two main data paths: one is in the LFSR and another one is in the nonlinear function F. Since the modulo $2^{31}-1$ addition is a time-consuming computation in ZUC with other five modulo $2^{31}-1$ additions in the path of the LFSR, it becomes the longest data path of ZUC in the initialization mode.

1. $u = ((S_{15H} \parallel S_{14L}) \oplus R_1) \boxplus R_2$
2. $v = \{2^{15}S_{15} + 2^{17}S_{13} + 2^{21}S_{10} + 2^{20}S_4 + (1+2^8)S_0\} \pmod{2^{31}-1}$
3. $S_{16} = v + u >> 1 \pmod{2^{31}-1}$
4. $IF S_{16} = 0$ then set $S_{16} = 2^{31}-1$

Since in the nonlinear function F, the value of R_1 and R_2 are required for the next loop computation as shown in Fig.1, if the LFSR needs to be updated per clock cycle, the value of R_1 and R_2 must be calculated in each clock. That means the data path in the nonlinear function F cannot be divided into sub blocks to increase the operating frequency. Based on this observation, the lower limit of the critical path of ZUC is determined by the path in the nonlinear function F, which consists of one 32-bit addition delay, a $SBox(8*8)$ delay, and some logic gates delay(linear transformation).

Here we will show the detailed comparison of the critical path between the existing designs and ours.

In the Pipeline Stage 2, since the cell S_{14} and S_{15} of LFSR are renewed with different values in the initialization stage and working stage, which induces an extra multiplexer into the data path in LFSR, for simplicity, this multiplexer is not marked in Fig.2. Based on this and Fig.6, the data path of our architecture in LFSR consists of one CSA, one modulo $2^{31}-1$ adder, and two multiplexers. The detailed comparison is shown in Table.1.

In reference to schema proposed by Zhang et al [10], the critical path of ours saves a 32-bit adder. As shown in Table.1, in THREE-ZUC and schema proposed by Zhang et al, the critical paths are much longer than that in the nonlinear function F. However, our data path in the LFSR is nearly one fourth of THREE-ZUC. If this path is the critical path of the ZUC, the throughput will be increased by four times. But the point is, the minimal length of a real feasible path is

Table 1. The critical path of this work and existing designs

	Critical Path
Our work(LFSR)	One Level CSA → 31-bit adder → a multiplexer → a multiplexer
Nolinear F	32-bit adder → XORs(linear transforms) → $SBox(8*8)$
THREE-ZUC(LFSR)	31-bit adder → 31-bit adder → 31-bit adder → 31-bit adder → a multiplexer
Zhang et al [10]	32-bit adder → a multiplexer → One Level CSA → one modulo $2^{31}-1$ adder

determined by the lower limit path in the nonlinear function F, which means the longer length of the two has the final say on the terminal throughput.

Is the data path in the LFSR of our architecture longer than the path in nonlinear function? The answer dependent on the implementation platform. Because the SBox delay is dependent on the hardware platform. We will discuss this issue in following subsection.

5.2 Evaluation result of the two-stage pipeline ZUC in FPGA

In order to verify the correctness and evaluate the performance of our architecture in FPGA platform, we implement the two-stage pipeline architecture in Verilog HDL and map it into Virtex-5 XC5VLX110T-3 and Virtex-6 XC6vlx75t-3 FPGA. The synthesis tool is ISE 11.5. The result of performance (in terms of throughput), consumed area (in terms of Xilinx FPGA slices), the ratio of throughput to area is given in Table.2.

Table 2. Comparison of our architecture with existing designs in Xilinx FPGA

Implementation	Technology	Frequency (MHz)	Area	Throughput (Mbps)	Throughput/Area
Proposed-ZUC	Virtex 6 XC6vlx75t-3	353	328 slices	11296	34
Proposed-ZUC	Virtex-5 XC5VLX110T-3	246	350 slices	7872	22
Kitsos et al [11]	Virtex 5	65	385 slices	2080	5.4
Wang et al [8]	Virtex-5 XC5VLX110T-3	108	356 slices	3456	9.7
Zhang et al [10]	Virtex-5 XC5VLX110T-3	172	395 slices	5504	13.9

As shown in the Table 5.2, the new architecture increases the throughput approximately by 45% compared with the latest and best implementation [10], and particularly it also saves nearly 12% of hardware resource.

5.3 Comparison with Existing Designs in ASIC

Comparison with Academic Designs in ASIC In order to compare with the exciting designs in ASIC, the gate-level synthesis was carried out using Synopsys Design Compiler Version G-2012.06-SP5, using topographical mode for a 65nm technology. The only hardware realizations for ZUC have been done in ASIC [5] so far. In order to compare with the commercial designs, the author used the best performance library in 65nm technology for the sake of fairness in their paper. Therefore, in our implementation, we also used the best performance library of 65nm target technology. The area results are reported using equivalent 2-input NAND gates.

As shown in Table. 5.3, our new architecture improves the throughput significantly comparing with the exciting designs. Since THREE-ZUC did not give the detailed information of the library which they used, we give the results synthesized by two different TSMC 65nm target libraries in Table 5.3. The rough analysis below shows the reason why our design can reach such high performance.

For the sake of simplicity, here we define a full adder delay as a unit time, denoted Δt. We assume that the adder is following CPA architecture. Here we ignore the routing delay. The 31-bit CPA delay is about $31\Delta t$. The delay of one stage CSA is about Δt. A multiplexer delay is about

Table 3. Comparison of our architecture with existing academic designs in 65nm technology

Implementation	Technology	Frequency (MHz)	Area (KGates)	Throughput (Gbps)
Proposed-ZUC	TSMC-GP 65nm	2500	12.5	80
Proposed-ZUC	TSMC-LP 65nm	2000	12.5	64
Gupta et al [5]	-	920	20	29.4

Δt, the minimum delay of the $SBox$ is about $8\Delta t$. So the critical path of our architecture is in the nonlinear function.

According to the above definition and comparison result in Table 1, the total delay of the path in nonlinear function Function is about $41\Delta t = 32\Delta t + 8\Delta t + 1\Delta t$, assuming XOR = 1 * Δt. The total delay of the critical path in our work is about $44\Delta t = 31\Delta t + 8\Delta t + 5\Delta t$. The total delay of the critical path in THREE-ZUC is about $125\Delta t = 31\Delta t + 31\Delta t + 31\Delta t + 31\Delta t + \Delta t$. From this point of view, the designs following our architecture will perform better than that of THREE-ZUC in the ASIC platform. The delay of the critical path of ours is about one third of THREE-ZUC, that means the throughput of our architecture is approximately three times the value of THREE-ZUC. Since the CLA uses more hardware resource than CPA, if our architecture and THREE-ZUC both use the adder with CPA architecture to save the hardware resource, the throughput of our design will be approximately three times the throughput of THREE-ZUC.

In fact, with the best performance constraint, the synthesized tool uses Carry Look-ahead Adder(CLA) architecture to implement addition, thus the delay of the 32-bit adder is much shorter than that of adder with CPA architecture. From the synthesized result of our design with TSMC-LP 65nm library and TSMC-GP 65nm library, our design can increase the throughput by 2 times and 2.7 times respectively compared with that of THREE-ZUC.

In reference to consumption of hardware resource, our architecture will use less hardware resource. Compared with THREE-ZUC, our new pipeline architecture utilizes Method 3 to calculate modulo $2^{31} - 1$ addition. In this way, the new architecture can save four 31-bit adders, and moreover, since only two stages in our pipeline, less hardware resource is needed to store the intermediate results. These points let our design save much more hardware area than that of THREE-ZUC.

Comparison with Commercial Designs in ASIC In the commercial area, both IP Cores Inc. [9] and Elliptic Tech Inc. [13] provide ZUC IP core in 65nm ASIC technology, and neither of them releases their architecture. As far as we know, the best implementation of ZUC in ASIC is given by IP cores Inc. This commercial ZUC IP core is released later than that of THREE-ZUC. IP Cores Inc. only claims that the best performance of their ZUC IP core can up to 40 Gbps in TSMC 65nm technology.

Table 4. Comparison of our architecture with existing commercial designs in 65nm technology

Implementation	Technology	Frequency (MHz)	Area (KGates)	Throughput (Gbps)
Proposed-ZUC	TSMC 65nm	2500	12.5	80
Elliptic Tech.	-	500	10 - 13	16
IP Cores Inc.	TSMC	-	-	40

6 Conclusion

In conclusion, we proposed a two-stage pipeline architecture of stream cipher ZUC in hardware. Compared with the previous works, the new architecture increases the throughput significantly

and saves much hardware resource in FPGA and ASIC. As the commercial IP companies have not released their designs, we hope this architecture could be a standard for hardware implementation of ZUC.

References

1. "Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC Specification version: 1.6," 2011.
2. "Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3.Document 1: 128-EEA3 and 128-EIA3 Specificationversion: 1.6," 2011.
3. "Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3.Document 3: Implementors Test Dataversion: 1.6," 2011.
4. "Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 4: Design and Evaluation Reportversion: 1.6," 2011.
5. S. S. Gupta, A. Chattopadhyay, and A. Khalid, "Designing integrated accelerator for stream ciphers with structural similarities," *Cryptography and Communications*, vol. 5, no. 1, pp. 19–47, 2013.
6. Z. Liu, L. Zhang, J. Jing, and W. Pan, "Efficient Pipelined Stream Cipher ZUC Algorithm in FPGA," in *The First International Workshop on ZUC Algorithm, December*, pp. 2–3.
7. P. Kitsos, N. Sklavos, G. Provelengios, and A. N. Skodras, "FPGA-based Performance Analysis of Stream Ciphers ZUC, Snow3g, Grain v1, Mickey v2, Trivium and E0," *Microprocessors and Microsystems*, 2012.
8. L. Wang, J. Jing, Z. Liu, L. Zhang, and W. Pan, "Evaluating Optimized Implementations of Stream Cipher ZUC Algorithm on FPGA," *Information and Communications Security*, pp. 202–215, 2011.
9. S. Sen Gupta, A. Chattopadhyay, and A. Khalid, "HiPAcc-LTE: An Integrated High Performance Accelerator for 3GPP LTE Stream Ciphers," *Progress in Cryptology–INDOCRYPT 2011*, pp. 196–215, 2011.
10. L. Zhang, L. Xia, Z. Liu, J. Jing, and Y. Ma, "Evaluating the optimized implementations of snow3g and zuc on FPGA," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pp. 436–442, IEEE, 2012.
11. P. Kitsos, N. Sklavos, and A. Skodras, "An FPGA Implementation of the ZUC Stream Cipher," in *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pp. 814–817, IEEE, 2011.
12. C. Koc, "RSA hardware implementation," *RSA Laboratories, August*, 1995.
13. " ZUC1 Ultra-Compact 3GPP Cipher Core(retrieved on February 5, 2012)," tech. rep., IP Cores Inc, http://ipcores.com/ZUC_cipher_IP_core.htm, 2012.