Design Space Exploration of Galois and Fibonacci Configuration based on Espresso Stream Cipher

Zhengyuan Shi[1], Gangqiang Yang *[1], Hailiang Xiong[1], Fudong Li[2], and Honggang Hu[3]

[1]Shandong University
[2]University of Alberta
[3]University of Science and Technology of China,
zshi0616@mail.sdu.edu.cn, g37yang@sdu.edu.cn,
hailiangxiong@sdu.edu.cn, fudongl@ualberta.ca, hghu2005@ustc.edu.cn

Abstract: Galois and Fibonacci are two different configurations of stream ciphers. Because the Fibonacci configuration is more convenient for cryptanalysis, most ciphers are designed as Fibonacci-configured. So far, although many transformations between Fibonacci and Galois configurations have been proposed, there is no sufficient analysis of their respective hardware performance. The 128-bit secret key stream cipher Espresso, its Fibonacci-configured variant and linear Fibonacci variant have a similar security level. We take them as examples to design the optimization strategies in terms of both area and throughput, investigate which configuration is more efficient in a certain aspect. The Fibonacci-configured Espresso occupies 52 slices on Spartan-3 and 22 slices on Virtex-7, which are the minimum solutions among those three Espresso schemes or even smaller than 80-bit secret key ciphers. Based on our throughput improvement strategy, parallel Espresso design can perform 4.1 Gbps on Virtex-7 FPGA and 1.9 Gbps on Spartan-3 FPGA at most. In brief, the Fibonacci cipher is more suitable for extremely resource-constrained or extremely high-throughput applications, while the Galois cipher seems like a compromise between area and speed. Besides, the transformation from nonlinear feedback to linear feedback is not recommended for any hardware implementations.

Keywords: lightweight cryptography; Espresso; FPGA Optimization; stream cipher; Galois NFSR; Fibonacci NFSR

1 Introduction

The stream ciphers have high throughput for software applications and highly restricted resources for hardware applications. Although some stream ciphers have been proved to contain design flaws [1], [2], [3], but it is undeniable that in addition to security level, throughput and area are also two significant factors for evaluating ciphers. Therefore, ECRYPT launched the eSTREAM project [4], many stream ciphers for confidentiality, integrity and implementation efficiency have been proposed and widely deployed, such as Trivium [5], Grain [6] and MICKEY [7]. Many

*The corresponding author is Gangqiang Yang security protocols based on the stream ciphers have also been accepted as international standards, for example, the 3rd Generation Partnership Project (3GPP) specifies UEA2, UIA2 [8] based on stream cipher SNOW 3G [9] and EEA3, EIA3 3GPP:EEA3 based on stream cipher ZUC [10].

Block cipher produces the ciphertext after nonlinear substitution with the plaintext input. Different from block cipher, stream cipher is also regarded as a pseudo-random keystream generator and produces the keystream bits sequence with the same length as plaintext. The ciphertext is generated bit by bit by XOR operation between plaintext and keystream. Generally, the simplified model of stream cipher is divided into two parts: feedback shift register (FSR) and filter function, as in Figure 1, The initial internal state of stream cipher is determined by the input data, including the secret key and initial vector (IV). Then, the internal state, stored in FSR, is updated in each round by the feedback function. When the feedback function in the FSR is nonlinear, the FSR is also called nonlinear feedback shift register (NFSR). Conversely, when the feedback function is linear, the register is called linear feedback shift register (LFSR). The filter function outputs 1 bit as the keystream bit in each round according to the internal state.

The feedback shift register (FSR) is one of the most critical components of stream cipher, which is divided into Galois configuration and Fibonacci configuration according to the feedback bits. As in figure 1, the Fibonacci configuration is more straightforward. Every feedback shift register only has one feedback function which feeds to the $n-1$th bit at each round, the other bits are consecutive moved one-bit step. Conversely, in the Galois configuration, the FSR has any number of feedback functions.

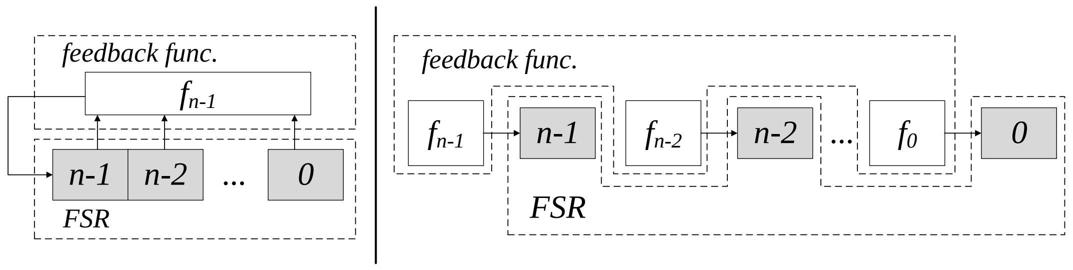


Figure 1: The Fibonacci configuration (left) and the Galois configuration (right) of stream ciphers

The stream cipher contains Fibonacci NFSR or Fibonacci LFSR is named as Fibonacci-configured cipher and the cipher contains Galois NFSR or LFSR is called as Galois-configured cipher. Because Fibonacci FSR conforms to cryptanalysis formally, most of the stream ciphers are designed in this configuration. For instance, Grain [6] consists of both Fibonacci NFSR and Fibonacci LFSR, each series [11] [12] [13] in WG family contains a Fibonacci LFSR.

Meanwhile, previous work provided the universal transformations, which can transform the Fibonacci-configured stream cipher and Galois-configured stream cipher to each other [14], and even transform the nonlinear feedback shift register to linear feedback shift register [15].

The analysis and optimization of FSR determine the area occupation and throughput of cipher implementations. In general, the depth of combinational logic circuits is smaller than that in Fibonacci configuration, hence, the Galois-configured stream cipher has a higher maximum frequency. The Galois-configured Grain [16] has a larger throughput of 41.3% than Fibonacci-configured Grain. According to the transformation [14], the feedback login is divided into several parts to drive various bits in feedback shift registers in Galois configuration, without adding any extra feedback login. Therefore, the author implied that in TSMC 90nm ASIC technology library, the circuit area of Galois-configured Grain is not much larger than that of Fibonacci-configured

Grain, even smaller [16].

1.1 Espresso Stream Cipher

Espresso stream cipher was proposed in 2015 [17] considering hardware size and throughput simultaneously. The original Espresso stream cipher, supporting 128-bit secret key is designed in Galois configuration, consists of a 256-bit Galois-configured non-linear feedback shift register (NFSR) with 14 individual feedback functions and a 20-variable keystream filter function. It supports 128 bits secret key and 96 bits initial vector. The implementations on ATmega328P and ESP8266 processors have shown that Espresso requires the minimum program size, global variables storage and computation time among Grain v1, Grain 128 and Grain 128a [18].

Based on the transformation mentioned above, another two Espresso variants are introduced. , Fibonacci-configured Espresso [17] (noted as Espresso-F) and Espresso-like LFSR filter generator [15] (noted as Espresso-L) have been introduced in the previous cryptography designs.

Espresso-F can be broken by related key chosen IV attack with 2^{42} IVs and time complexity 2^{64} [19], Espresso-L can be broken with time complexity $2^{68.44}$ and $2^{66.86}$ under standard algebraic attack and the Ronjom-Helleseth attack respectively [15]. Although the cryptanalysis has proved that the variants do not fulfill the 128-bit security level, they can still be accepted under certain circumstances, such as for lightweight devices [20]. Consequently, the resource occupancy and throughput analysis based on Espresso and its variants may provide guidance and references for the selection of Galois or Fibonacci ciphers.

1.2 Related Work and Motivation

Unlike ASIC, the semicustom Field Programmable Gate Array (FPGA), including reconfigurable combinational logic and flip-flops, is applied to shorten time-to-market, save development costs and implement the domain-specific computing architecture. With those advantages, FPGA is more and more adopted in numerous areas, such as military, industry network, IoT, aeronautics and astronautics. Many kinds of ciphers are deployed in the programmable chips to not only evaluate hardware efficiency but also provide security solutions for FPGA application, such as WG-8 [21], Lizard [22], Grain [23] and Trivium [23].

The eSTREAM project's [4] requirements for stream cipher are high throughput and more hardware efficiency. Grain [6] and Trivium [5] are regarded as the pioneers of modern lightweight cryptography, because of their compact architectures and efficient hardware performance with only 80-bit security level. Trivium is considered as a kind of Galois-configured cipher and Grain is a typical Fibonacci-configured cipher. So far, there is still no comprehensive discussion of both advantages and disadvantages of Galois and Fibonacci configurations implemented on FPGA. Meanwhile, Espresso and its variants provide an ideal condition for the discussion, they ensure that in the case of similar cryptography attack complexity, the discussion can determine which configuration is more recommended towards a particular scenario.

1.3 Our Contributions

In our paper, we firstly provide a brief overview of original Espresso (Galois configuration [17]) and its variants, including Espresso-F in Fibonacci configuration [17] and Espresso-L, an Espresso-like LFSR filter generator [15]. Based on that, Espresso and its variants are deployed on Xilinx FPGA toward the minimum area and highest throughput, respectively. We then investigate which is the smallest solution aimed for 256 bits NFSR implementation. After that, a hybrid architecture is proposed for throughput improvement, where Espresso initializes serially and produces multiple keystream bits with a parallel architecture in the running phase during each clock, thus, throughput increases significantly.

We noticed that, unlike AISC implementations, Fibonacci-configured cipher is more area-saved than Galois-configured cipher on FPGA, but the latter is faster. For example, on Virtex-7 FPGA, Fibonacci-configured Espresso only occupies 22 slices with a maximum frequency of 427.2 MHz, while Galois-configured Espresso occupies 25 slices, with a maximum frequency of 491.4 MHz. The reason is that though both NFSRs have the same number of logic gates [14], Galois NFSR feedback functions are independent to be synthesized as look-up-tables, the minimum reconfigurable unit in FPGA, it will lead to more area than Fibonacci NFSR.

Besides, according to our throughput improvement strategy, Fibonacci configuration can support a larger parallel width, i.e., support to use extra resources to increase the number of keystream bits per cycle. The maximum throughput of Fibonacci-configured Espresso is 4.09 Gbps on Virtex-7 FPGA.

Based on our discussion, the transformation from Galois NFSR (original Espresso) to Fibonacci LFSR (Espresso-L) will generate more complex circuit than that to Fibonacci NFSR (Espresso-F). This kind of transformation should be avoided in FPGA implementation.

In brief, Fibonacci-configured stream cipher should be considered for scenarios with area requirements and parallel Fibonacci-configured stream cipher should be used for high throughput applications. Galois-configured stream cipher takes both throughput and area into account, which is suitable for compact devices but slightly requires high throughput.

In a nutshell, we list several main contributions as following:

- We conclude three different configurations of Espresso stream cipher, the original Espresso with Galois NFSR, the variant Espresso with Fibonacci NFSR and another variant Espresso with Fibonacci LFSR. Meanwhile, the comparison of Espresso and its variants show that Galois-configured cipher performs evidently faster than Fibonacci configured cipher on FPGA, but the former occupies more slices. Espresso-like LFSR filter generator variant is not recommended due to larger area and higher latency for hardware implementation and weak resistance for security analysis.

- To deploy cipher on the resource-compact devices, we provide an area optimized method with shift register look-up-table (SRL). According to the ratio of LUT to FF about different series FPGAs. We investigate the minimum area solution by enumerating the SRL replacement length of consecutive registers fragment. Based on the method, the minimum Espresso solution only takes 22 slices on Virtex-7 and 52 slices on Spartan-3. Our efficient Espresso solutions can be deployed on tiny devices, especially emergent IoT wireless devices.

- To fulfill the high throughput requirement, we propose a hybrid architecture which not only realizes the 4-bit solution but improves the parallel width to 8-bit and 16-bit. In short, the additional feedback functions tap more significant internal state bits than the serial feedback function, where, the unloaded feedback function value will be used as a variable if the variable index has been over-range. Although this method will cause more critical path propagation delay, the throughput of 16-bit hybrid architecture can reach more than 4 Gbps on Virtex-7, satisfied high-speed demand.

The following sections are organized as follows. Section 2 lists some cipher variables and notations. Section 2 provides a summary of Espresso design criteria and algorithm flow. The original Espresso in Galois configuration is implemented on FPGA in Section 3, including serial solution and hybrid solution. The Fibonacci-configured Espresso is described in Section 4. What's more, another throughput improved method achieving more than 4-bit parallel width is proposed. Section 5 briefs NFSR to Fibonacci LFSR transformation and provides implementation results on FPGA. The comparison between Espresso with its variants and other ciphers is shown in Section 6. Finally, we conclude this paper in Section 7.

2 Overview of Espresso Stream Cipher

2.1 Design of Espresso Stream Cipher

Espresso stream cipher aims at lightweight and high-speed applications. It is designed under the trade-off between hardware area and latency, with 128-bit secret key (k) and 96-bit initial vector (v). Espresso is designed as the fastest lightweight cipher in below 1500 GEs area level, better than Grain-128 and Trivium [17]. The block design diagram of Espresso stream cipher is shown in Figure 2.

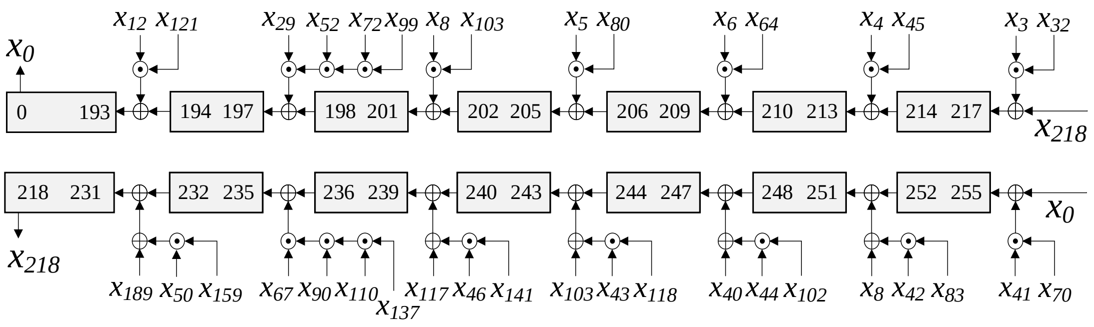


Figure 2: The block diagram of Espresso stream cipher

As a stream cipher, Espresso has 256-bit nonlinear feedback shift register (NFSR) in Galois configuration with 14 parallel feedback functions specified as follows:

$$
\begin{aligned}
f_{255}(x) &= x_0 \oplus x_{41}x_{70}\\
f_{251}(x) &= x_{252} \oplus x_{42}x_{83} \oplus x_8\\
f_{247}(x) &= x_{248} \oplus x_{44}x_{102} \oplus x_{40}\\
f_{243}(x) &= x_{244} \oplus x_{43}x_{118} \oplus x_{103}\\
f_{239}(x) &= x_{240} \oplus x_{46}x_{141} \oplus x_{117}\\
f_{235}(x) &= x_{236} \oplus x_{67}x_{90}x_{110}x_{137}\\
f_{231}(x) &= x_{232} \oplus x_{50}x_{159} \oplus x_{189}\\
f_{217}(x) &= x_{218} \oplus x_3x_{32}\\
f_{213}(x) &= x_{214} \oplus x_4x_{45}\\
f_{209}(x) &= x_{210} \oplus x_6x_{64}\\
f_{205}(x) &= x_{206} \oplus x_5x_{80}\\
f_{201}(x) &= x_{202} \oplus x_8x_{103}\\
f_{197}(x) &= x_{198} \oplus x_{29}x_{52}x_{72}x_{99}\\
f_{193}(x) &= x_{194} \oplus x_{12}x_{121}
\end{aligned}
\tag{1}
$$

We can define those 14 bits in update set U:

$$U = \{193, 197, 201, 205, 209, 213, 217, 231, 235, 239, 243, 247, 251, 255\} \tag{2}$$

Note set V includes all variables used in feedback function. The variable set V of Espresso is:

$$\begin{aligned} V = \{&0, 3, 4, 5, 6, 8, 12, 29, 32, 40, 41, 42, 43, 44, 45, 46, 50, 52, 64, 67, 70, 72, \\ &80, 83, 90, 99, 102, 103, 110, 117, 118, 121, 137, 141, 159, 189, 194, 198, \\ &202, 206, 210, 214, 218, 232, 236, 240, 244, 248, 252\} \end{aligned} \tag{3}$$

The NFSR bits which are not in update set U are loaded from higher bits. Hence, Espresso internal state update is as following:

$$x_i^{t+1} = \begin{cases} f_i(x), \ i \in U \\ x_{i+1}^t, \ i \notin U \end{cases} \tag{4}$$

It also has a 20-variable nonlinear output function $h(x)$ as keystream filter function, consists of a 6-variable linear function and a 14-variable nonlinear function:

$$\begin{aligned} h(x) =& x_{80} \oplus x_{99} \oplus x_{137} \oplus x_{227} \oplus x_{222} \oplus x_{187} \oplus x_{243}x_{217} \oplus x_{247}x_{231} \oplus x_{213}x_{235} \\ &\oplus x_{255}x_{251} \oplus x_{181}x_{239} \oplus x_{174}x_{44} \oplus x_{164}x_{29} \oplus x_{255}x_{247}x_{243}x_{213}x_{181}x_{174} \end{aligned} \tag{5}$$

2.2 Espresso Algorithm Flow

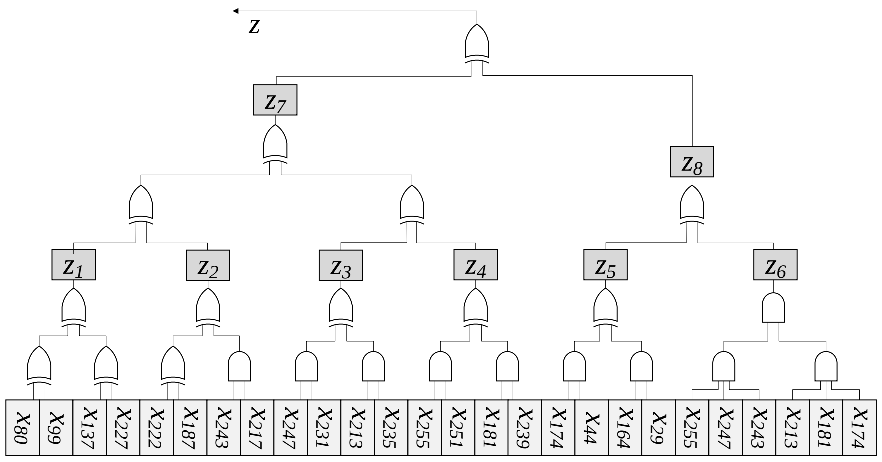
Espresso as a keystream generator only has 2 phases, they are initialization and running phases. Before executing 256 initialization rounds, the 256-bit NFSR is loaded with secret key (128 bits) and initial vector bits (96 bits) as Formula 6. The initial internal state from x_{224} to x_{254} is assigned as 1 to ensure there is no full-zero state in NFSR. The most significant bit x_{255} is set to 0.

$$\begin{aligned} &x_i^0 = k_i, 0 \leq i < 128 \\ &x_i^0 = v_{i-128}, 128 \leq i < 224 \\ &x_i^0 = 1, 224 \leq i < 255 \\ &x_i^0 = 0, i = 255 \end{aligned} \tag{6}$$

The output bit from filter function $h(x)$ result is fed into feedback functions $f_{255}(x)$ and $f_{217}(x)$ as Formula 7 during initialization phase, and the remaining feedback functions are invariable in any phases. After 256 times update, the filter result $h(x)$ is output as keystream, no longer feeding to internal state update.

$$\begin{aligned} f_{217}(x) =& x_{218} \oplus x_3x_{32} \oplus h(x) \\ f_{255}(x) =& x_0 \oplus x_{41}x_{70} \oplus h(x) \end{aligned} \tag{7}$$

Espresso also introduces a pipeline circuit implementing the 20-variable filter function, shown in Figure 3. As such 3-stage pipeline structure, Espresso has no valid output keystream at the first 3 cycles after initialization. However, the combinational logic circuit is synthesized into look-up-table, which packages 4 or 6 input variables. As a general rule, there are just 2 logic levels about the filter function under Xilinx 7 series FPGA synthesize strategy. Therefore, it's useless to adopt the pipeline structure in the following FPGA implementation.



z
z_7
z_8
z_1
z_2
z_3
z_4
z_5
z_6
x_{80} x_{99} x_{137} x_{227} x_{222} x_{187} x_{243} x_{217} x_{247} x_{231} x_{213} x_{235} x_{255} x_{251} x_{181} x_{239} x_{174} x_{44} x_{164} x_{29} x_{255} x_{247} x_{243} x_{213} x_{181} x_{174}

Figure 3: The pipeline structure of Espresso filter function

3 Implementation of Galois-configured Espresso

In this section, we explore the efficient FPGA implementation strategies for Espresso stream cipher in Galois configuration serially and parallelly. The hardware architectures in this section and the following sections have no secret key storage component and the input vectors (k and v) are loaded into Espresso hardware bit by bit. This is a common and acceptable FPGA hardware performance evaluating method used for Grain [23], ZUC [24], Snow3g [24] and WG-8 [21].

3.1 Overview of Espresso FPGA Implementation

Field programmable gate array (FPGA) has been adopted everywhere, including IoT and artificial intelligence. Different from ASIC, FPGA can be reconfigured after manufacturing like microprocessor, but the hardware circuit (registers, logic gate) can be directly mapped into FPGA resources gearing to the needs of higher speed and lower power consumption. In our paper, the optimized Espresso keystream generator architectures are deployed on Xilinx Spartan-3 and Virtex-7 FPGAs.

Take Galois-configured Espresso as an example, the architecture has clock port clk, reset port rst, 1-bit width data input port in receiving secret key or initial vector from storage component (testbench) and 1-bit width output port ks. It also has two output signal ports $load$ for initial data input enable and $work$ for the keystream output enable after initialization. Hence, Espresso or Espresso-F controller has 4 modes, they are IDLE, LOAD, INIT, WORK implemented as a finite state machine (FSM) with 9-bit counter cnt. The FSM transformation is shown in Figure 4. When counter increases to 256 during LOAD stage, the state machine transforms into the next state INIT. When the counter overflows to 0 during INIT state, the state machine changes to WORK and keeps that state until reset. Therefore, $load$ signal is synchronous with the finite state machine's LOAD state and $work$ is coincident with WORK state. The Espresso-like LFSR filter generator has another procedure for loaded compensated internal state, we will discuss the variant Espresso-L in Section 5.

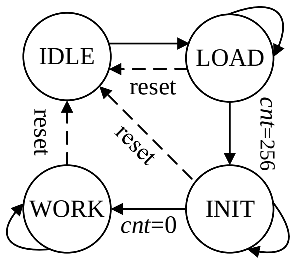


Figure 4: Espresso and Espresso-F FSM transformation diagram

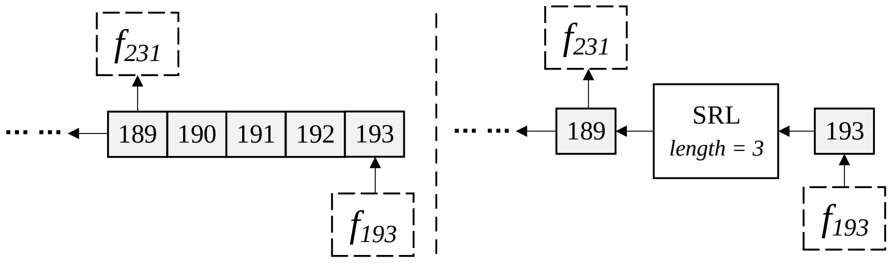


Figure 5: The consecutive register fragments optimized design

3.2 Efficient Implementation of NFSR

The 256-bit nonlinear feedback shift register is the only sequential logical component in Espresso hardware determining the latency and area occupation. Galois NFSR includes 14 feedback functions updating 14 bits internal state parallelly once triggered by clock edge. The combinational logic circuit is synthesized as LUT, specially, 6-input LUT on Virtex-7 FPGA. Hence, the 20-variable filter function of Espresso NFSR tapping 20 registers is no longer pipelined with only 2 logic levels.

The most commonly used method to reduce slices on FPGA is replacing consecutive registers (synthesized as FFs) by shift register look-up-table (SRL). On 7 series FPGA, a LUT may also be configured as SRL32 to substitute for 32-bit shift register at most, and 4-input LUT can be synthesized as SRL16 on Spartan-3. However, the internal state of SRL cannot be detected by any wire except an output port. Therefore, when cipher has a large number of consecutive internal states, the SRL optimized method is evidently effective. For instance, Espresso's registers from x_{190} to x_{192} 3 bits are not tapped for any function, the 3 flip-flops can be replaced by one SRL to save area. This SRL output port is tapped as x_{190} and loaded from x_{192}.

We define the consecutive register fragment from x_a to x_b as $R(a,b)$, whose valid length is equal to $b-a-1$, representing x_a and x_b are implemented as flip-flops to keep only one logic level between the optimized fragment beginning and ending at FFs. The fragment $R(a,b)$ two terminal bits x_a and x_b, which is tapped or updated by function. As the above example, the register fragment $R(189,193)$ left-shown in Figure 5 is optimized designed as right-shown.

Each reconfigurable slice on Virtex-7 has 4 6-input LUTs and 8 flip-flops. In general term, the ratio of LUT to FF in minimum area solutions should approach 1: 2. Noteworthily, the ratio is not necessarily equal to 1: 2 by the reason of placing and routing. Moreover, with a

Table 1: The Consecutive Register Fragments of Galois Espresso

No.	Fragment	Len.	No.	Fragment	Len.	No.	Fragment	Len.	No.	Fragment	Len.
1	$R(141,159)$	17	11	$R(110,117)$	6	21	$R(193,197)$	3	31	$R(243,247)$	3
2	$R(12,29)$	16	12	$R(174,181)$	6	22	$R(197,201)$	3	32	$R(247,251)$	3
3	$R(121,137)$	15	13	$R(181,187)$	5	23	$R(201,205)$	3	33	$R(251,255)$	3
4	$R(52,64)$	11	14	$R(159,164)$	4	24	$R(205,209)$	3	34	$R(0,3)$	2
5	$R(164,174)$	9	15	$R(217,222)$	4	25	$R(209,213)$	3	35	$R(29,32)$	2
6	$R(90,99)$	8	16	$R(222,227)$	4	26	$R(213,217)$	3	36	$R(64,67)$	2
7	$R(32,40)$	7	17	$R(8,12)$	3	27	$R(227,231)$	3	37	$R(67,70)$	2
8	$R(72,80)$	7	18	$R(46,50)$	3	28	$R(231,235)$	3	38	$R(80,83)$	2
9	$R(83,90)$	6	19	$R(137,141)$	3	29	$R(235,239)$	3	39	$R(99,102)$	2
10	$R(103,110)$	6	20	$R(189,193)$	3	30	$R(239,243)$	3	40	$R(118,121)$	2

few LUTs are used for combinational logic function rather than shift register, replacing all 2-bit length consecutive registers is discouraged. The smallest Espresso architecture has to consider the trade-off between registers implemented by FFs and registers implemented by LUTs. Along this line, we list all consecutive fragments in Table 1 according to the filter function, feedback function tapped positions and feedback function updated positions.

Firstly, every consecutive register fragment with more than or equal to 5 bits length is synthesized as SRL. Despite the longest fragment has 17-bit register, over the maximum SRL length 16 bits on 3 series FPGAs, there are 13 SRLs occupied not only on Virtex-7 but also on Spartan-3. The 17-bit register fragment $R(141, 159)$ is divided into 1-bit single register and 16-bit register, thus the length changes to 16 bits which may be synthesized as one SRL16.

Secondly, based on above solution, we continue replacing 4 bits length fragments, $R(159, 164)$, $R(217, 222)$ and $R(222, 227)$. There are three more LUTs are used, shown Table 2. Noted that there are 11 flip-flops reduced rather than 12 flip-flops, because the terminal bit x_{217} is driven by feedback function $f_{217}(x)$, and insert another flip-flop will reduce one logic level to improve frequency.

Finally, we replace 17 3-bit length fragments and 7 2-bit length fragments respectively. It's obvious that taking replacement on 3-bit and 2-bit fragments is not necessary on Virtex-7, shown in Table 2. Until now, apply SRL optimization method for every more than or equal to 4 bits length register fragment can lead to the minimum area on Virtex-7. Meanwhile, due to the different ratio of LUT to FF between Virtex-7 and Spartan-3 FPGA, the same method should be applied for Spartan-3 implementation toward the smaller solution. Table 2 also shows the implementation results on Spartan-3. In brief, if a fragment includes more than 1 registers, it should be replaced by SRL.

As a result, Espresso keystream generator architecture occupies 62 slices on Spartan-3 FPGA and 25 slices on Vritex-7 FPGA at least.

3.3 Throughput Improvement in Hybrid Architecture

Galois-configured Espresso stream cipher performs high throughput, up to 2.22 Gbps under 90nm CMOS technology [17]. We can improve throughput further by redesigning the NFSR in parallel. Denote parallel width w means parallelized cipher generates w keystream bits in each clock. In serial solution, the feedback function $f_i(x)$ used to update x_i, but in parallel solution, the functions have to be copied for w times. Table 3 showing internal state update in serial will be conducive to the comprehension and exploration of parallel scheme.

Table 2: The Results of SRL Optimized Method in Galois Configuration

Device	SRL replacement R length $\geq$	#LUT /#FF	#LUT as SRL	Area (Slices)	Freq. (MHz)	T./A. (Mbps/Slices)
Virtex-7	5 bits	45/148	13	30	572.1	19.07
	4 bits	48/137	16	**25**	491.4	19.66
	3 bits	65/93	33	29	445.6	15.37
	2 bits	72/79	40	32	422.7	13.21
Spartan-3	5 bits	63/144	13	85	176.1	2.07
	4 bits	66/133	16	79	179.3	2.27
	3 bits	83/94	33	70	182.5	2.61
	2 bits	90/80	40	**62**	198.5	3.20

Table 3: The Serially Update of the NFSR from x_{206} to x_{209}

Round/Cycle	x_{206}	x_{207}	x_{208}	x_{209}
t	x_{206}^{t}	x_{207}^{t}	x_{208}^{t}	x_{209}^{t}
$t+1$	x_{206}^{t+1} x_{207}^{t}	x_{207}^{t+1} x_{208}^{t}	x_{208}^{t+1} x_{209}^{t}	x_{209}^{t+1} $f_{209}(x^{t})$
$t+2$	x_{206}^{t+2} x_{208}^{t}	x_{207}^{t+2} x_{209}^{t}	x_{208}^{t+2} $f_{209}(x^{t})$	x_{209}^{t+2} $f_{209}(x^{t+1})$
$t+3$	x_{206}^{t+3} x_{209}^{t}	x_{207}^{t+3} $f_{209}(x^{t})$	x_{208}^{t+3} $f_{209}(x^{t+1})$	x_{209}^{t+3} $f_{209}(x^{t+2})$
$t+4$	x_{206}^{t+4} $f_{209}(x^{t})$	x_{207}^{t+4} $f_{209}(x^{t+1})$	x_{208}^{t+4} $f_{209}(x^{t+2})$	x_{209}^{t+4} $f_{209}(x^{t+3})$

Table 4: The Parallelly Update of the NFSR from x_{206} to x_{209}

Cycle	x_{206}	x_{207}	x_{208}	x_{209}
t	x^t_{206}	x^t_{207}	x^t_{208}	x^t_{209}
$t+1$	$f^0_{209}(x^t)$	$f^1_{209}(x^t)$	$f^2_{209}(x^t)$	$f^3_{209}(x^t)$

Take a part of Espresso NFSR (from x_{206} to x_{209}) as an example, the feedback function $f_{209}(x)$ drives x_{209}. At the next round (Round $= t+1$), the NFSR part from x^{t+1}_{206} to x^{t+1}_{209} is equal to x^t_{207}, x^t_{208}, x^t_{209} and $f_{209}(x^t)$, which shifts one bit and loads the function $f_{209}(x^t)$. After that, the 4-bit NFSR part shifts one more bits at round $t+2$, changes to x^t_{208}, x^t_{209}, $f_{209}(x^t)$ and $f_{209}(x^{t+1})$. Meanwhile, $f_{209}(x^{t+1})$ take NFSR $t+1$ round internal state as variable, which can be calculated by shifting x^t one bit. For the same reason, we can get the internal state at round $t+3$ and $t+4$. Thus, Espresso can update multiple rounds in each cycle has been proved.

Here we define $f^j_i(x)$ used to update $x_{i-(w-1-j)}$ in each clock in parallel schemes, as Table 4. Every index of $f^j_i(x)$ variables should add j on the basis of original index, similar as getting the variables for function $f_i(x)$ at the next j rounds in serial. For example, function $f_{209}(x) = x_{210} \oplus x_6x_{64}$ drives internal state x_{209}, if we note the parallel width is 4 bits, there are another 4 inferred functions based on $f_{209}(x)$:

$$\begin{aligned} f^0_{209}(x) =& x_{210+0} \oplus x_{6+0}x_{64+0} \\ f^1_{209}(x) =& x_{210+1} \oplus x_{6+1}x_{64+1} \\ f^2_{209}(x) =& x_{210+2} \oplus x_{6+2}x_{64+2} \\ f^3_{209}(x) =& x_{210+3} \oplus x_{6+3}x_{64+3} \end{aligned} \tag{8}$$

For the hybrid-designed Espresso, the cipher update by the Formula 4 before generating keystream phase. After 256 clocks for initialization, Espresso update w rounds in each clock following:

$$x^{t+1}_i = \begin{cases} f^{w-1-j}_{i+j}(x),\ \exists j : (i+j) \in U, j = 0, 1, ..., w-1 \\ x^t_{i+w},\ other \end{cases} \tag{9}$$

According to the above formulas, x_{206} updates by $f^0_{209}(x)$, x_{207} updates by $f^1_{209}(x)$, x_{208} updates by $f^2_{209}(x)$ and x_{209} updates by $f^3_{209}(x)$ respectively. Hence, the hybrid Espresso take forward for 4 rounds once trigged by clock edge.

However, the minimum number of spare registers between tapped bit and the near bit in update set U determines the maximum parallel width. If we increase one bit parallel ($w = 5$), another additional function may be noted as $f^4_{209}(x) = x_{210+4} \oplus x_{6+4}x_{64+4}$. But the monomial x_{210+4} is not correct because x_{213} is driven by another feedback function instead of the more significant bit x_{214}. Hence, one of the $f^4_{209}(x)$ variables is not existed in current internal state, but equal to $f^0_{213}(x)$:

$$f^4_{209}(x) = f^0_{213}(x) \oplus x_{6+4}x_{64+4} \tag{10}$$

In this case, the data path of $f^4_{209}(x)$ based on $f^0_{213}(x)$ will lead to much latency and we will adopt this method in the following Fibonacci variants Espresso hybrid architecture. Therefore, we stipulate that the maximum parallel width of Galois Espresso is 4 bits. For example, Trivium stream cipher has the maximum parallel width 64 because there are at least 64 non-tapped iterations lower than modified bit [5].

We conclude the maximum parallel width in this method is w_{max}, which can be gotten by:

$$w_{max} = min(u - v + 1), (u \in U, v \in V, u > v) \tag{11}$$

Table 5: Implementation Results of Espresso Hybrid Architecture

Device	Strategy	#LUT /#FF	Area (Slices)	Freq. (MHz)	Thro. (Mbps)	T./A. (Mbps/Slices)
Virtex-7	Serial	48/137	25	491.4	491.4	19.66
	Hybrid x4	211/267	60	373.3	1493.1	24.88
Spartan-3	Serial	90/80	62	198.5	198.5	3.20
	Hybrid x4	371/267	198	163.3	653.3	3.30

Furthermore, there is another problem which restricts Espresso parallelized implementation. Espresso's filter function $h(x)$ variables x_{255}, x_{247}, x_{243} and x_{213} belong to set U, along the above discussion, the variables of $h^1(x)$ function includes one more significant bit of $h^0(x)$ (i.e. $h(x)$) variables, but we can't get the next bits in set U just according to current internal state.

In another way, the filter function is no more fed into NFSR after initialization. Espresso can firstly update for 4 rounds and then produce 4 keystream bits, the 4 filter functions are noted as $h^0(x)$, $h^{-1}(x)$, $h^{-2}(x)$ and $h^{-3}(x)$. Consequently, other device collects keystream from $z_{w(t-2)}$ to $z_{w(t-1)-1}$ at clock t rising edge, which represents once hybrid Espresso detects the first rising edge in finite state machine WORK state, there is no valid keystream bit. Precisely, the first w bits keystream $z_0, z_1, ..., z_{w-1}$ are sampled at the second clock $t = 2$ rising edge during running phase. That can be named as *first update then filter* strategy.

As a result, our optimized Espresso is designed in hybrid architecture, Espresso update serially during load and initialization phases and update multiple rounds in each clock during processing plaintext phase. The FPGA implement results are listed in Table 5.

4 Description and Implementation of Fibonacci-configured Espresso

In this section, we will provide a brief description about Galois-to-Fibonacci transformation method [14]. Subsequently, the serial and hybrid architectures are introduced for efficient FPGA implementation.

4.1 Galois-to-Fibonacci Transformation

Fibonacci NFSR is a special kind of Galois NFSR, whose feedback functions $f_i(x) = x_{i+1}$ except the most significant bit. For n-bit Fibonacci NFSR, the update set only has one element, $U = \{n-1\}$. Earlier research has concluded that Galois NFSR is more efficient due to the parallel feedback functions [25], for example, Galois-configured Grain-80 [16] can perform 58% higher frequency than that in Fibonacci configuration, but Fibonacci NFSR has significant advantages for security analysis due to few number of feedback functions.

According to transformation method in [14], Espresso NFSR can be configured as two Fi-

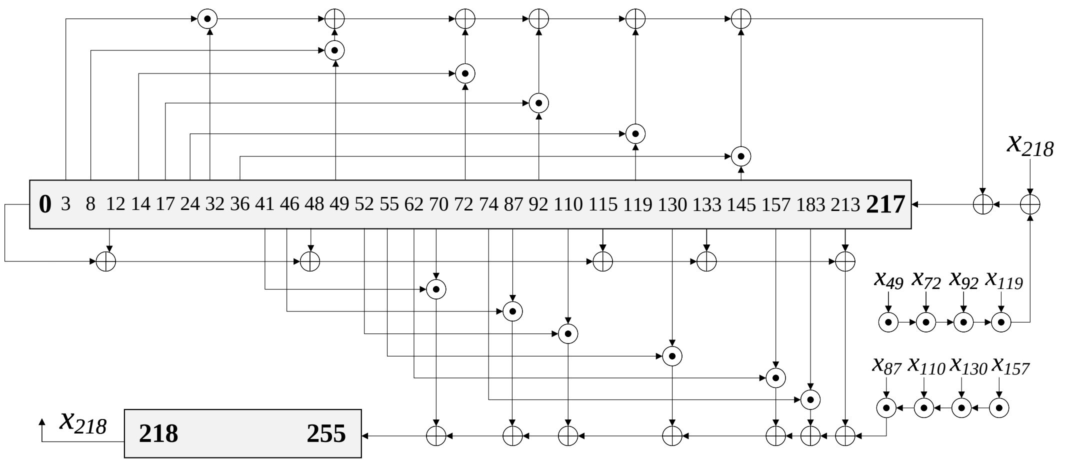


Figure 6: The block diagram of Fibonacci-configured Espresso-F

bonacci NFSRs, updated by function $f_{255}(x)$ and function $f_{217}(x)$:

$$\begin{aligned}
f_{255}(x) =& f_L(x) \oplus f_N(x) \\
=& x_0 \oplus x_{12} \oplus x_{48} \oplus x_{115} \oplus x_{133} \oplus x_{213} \\
& \oplus x_{41}x_{70} \oplus x_{46}x_{87} \oplus x_{52}x_{110} \oplus x_{55}x_{130} \oplus x_{62}x_{157} \oplus x_{74}x_{183} \oplus x_{87}x_{110}x_{130}x_{157} \\
f_{217}(x) =& x_{218} \oplus f'_N(x) \\
=& x_{218} \\
& \oplus x_3x_{32} \oplus x_8x_{49} \oplus x_{14}x_{72} \oplus x_{17}x_{92} \oplus x_{24}x_{119} \oplus x_{36}x_{145} \oplus x_{49}x_{72}x_{92}x_{119}
\end{aligned} \tag{12}$$

It's evidently that $f_{255}(x)$ combines a 6-variable linear function $f_L(x)$ and a 12-variable nonlinear function $f_N(x)$. Meanwhile, $f_{217}(x)$ has the similar specification, XORed by x_{218} and shifted version of 12-variable nonlinear function $f'_N(x)$. We can note the Fibonacci configuration Espresso as Espresso-F, shown in Figure 6.

4.2 Efficient Implementation in Fibonacci Configuration

In the above transformation method, a Fibonacci-configured NFSR has one element in update set U. Despite the fact that only one feedback function, the extra monomials in $f_{255}(x)$ are shifted from other functions. Thus, the quantity of logic gates is theoretically constant between Galois-configured NFSR and Fibonacci-configured NFSR. However, hardware implementation on FPGA is different from that on ASIC, namely, combinational logic gate is synthesized as look-up-table similar to distribute RAM stored the logic truth table. The special feature represents several logic gates can be packaged into one LUT, the Fibonacci NFSR with larger depth of logic circuit gates has significant advantage.

On another hand, only one bit x_{255} is driven by feedback function in Espresso-F, 13 update tapped positions has been deleted, potentially forming longer consecutive register fragments. We can also list all consecutive fragments in Table 6. The longest fragment is $R(187, 213)$ with 25 bits length.

Table 6: The Consecutive Register Fragments of Espresso-F

No.	Fragment	Len.	No.	Fragment	Len.	No.	Fragment	Len.	No.	Fragment	Len.
1	$R(187,213)$	25	12	$R(157,164)$	6	23	$R(115,119)$	3	34	$R(251,255)$	3
2	$R(145,157)$	11	13	$R(174,181)$	6	24	$R(133,137)$	3	35	$R(0,3)$	2
3	$R(99,110)$	10	14	$R(74,80)$	5	25	$R(183,187)$	3	36	$R(14,17)$	2
4	$R(119,130)$	10	15	$R(3,8)$	4	26	$R(213,217)$	3	37	$R(29,32)$	2
5	$R(164,174)$	9	16	$R(24,29)$	4	27	$R(218,222)$	3	38	$R(41,44)$	2
6	$R(62,70)$	7	17	$R(36,41)$	4	28	$R(227,231)$	3	39	$R(49,52)$	2
7	$R(137,145)$	7	18	$R(87,92)$	4	29	$R(231,235)$	3	40	$R(52,55)$	2
8	$R(17,24)$	6	19	$R(110,115)$	4	30	$R(235,239)$	3	41	$R(130,133)$	2
9	$R(55,62)$	6	20	$R(222,227)$	4	31	$R(239,243)$	3			
10	$R(80,87)$	6	21	$R(8,12)$	3	32	$R(243,247)$	3			
11	$R(92,99)$	6	22	$R(32,36)$	3	33	$R(247,251)$	3			

Table 7: Results of SRL Optimized Method in Fibonacci Configuration

Device	SRL replacement R length $\geq$	#LUT /#FF	#LUT as SRL	Area (Slices)	Freq. (MHz)	T./A. (Mbps/Slices)
Virtex-7	5 bits	44/147	14	28	533.3	19.05
	4 bits	50/123	20	24	415.6	17.32
	3 bits	64/81	34	**22**	427.2	19.42
	2 bits	71/67	41	25	397.1	15.89
Spartan-3	5 bits	56/141	15	91	167.1	1.84
	4 bits	62/117	21	79	172.6	2.19
	3 bits	76/75	35	59	169.7	2.88
	2 bits	83/61	42	**52**	195.4	3.76

Based on the table, we investigate the variation between area and SRL replaced fragment threshold length, similar to the serial Galois Espresso discussion. The implementation results are shown in Table 7. There are 25 slices occupied on Virtex-7 and 52 slices on Spartan-3. Compared with Galois Espresso, the Fibonacci-configured Espresso-F uses less reconfigurable resources. Although Espresso-F critical data path has less route delay with smaller placement, more combinational logic level will lead to much propagation time. On balance, Fibonacci configuration is smaller but slower than Galois configuration on FPGA.

It should be noted that $R(187,213)$ with 25 bits length is divided into $R(187,204)$ and $R(204,213)$ on Spartan-3, because for 4-input LUT, the synthesized SRL maximum length is $2^4 = 16$ bits. The solutions on Spartan-3 have one more SRL than that on Virtex-7.

4.3 Hybrid Fibonacci-configured Espresso Implementation

Fibonacci-configured Espresso consists of a 218-bit NFSR and a 38-bit NFSR driven by functions $f_{217}(x)$ and $f_{255}(x)$ respectively. Compared with Galois feedback shift register, each Fibonacci FSR has only one update function, thus, the Espresso-F update set U_A and feedback function variable set V_A are following:

$$\begin{aligned}
U_A &= \{217, 255\} \\
V_A &= \{0, 3, 8, 12, 14, 17, 24, 32, 36, 41, 46, 48, 49, 52, 55, 62, 70, \\
&\quad 72, 74, 87, 92, 110, 115, 119, 130, 133, 145, 157, 183, 213, 218\}
\end{aligned} \tag{13}$$

According to Formula 11, when the iteration element u in set U is 213 and v is 217, the maximum parallel width w_{max} is still 5.

To make further investigation, we take the 5-bit parallel solution as an example:

$$\begin{aligned}
f^0_{255}(x) =& f^0_L(x) \oplus f^0_N(x) \\
f^1_{255}(x) =& f^1_L(x) \oplus f^1_N(x) \\
f^2_{255}(x) =& f^2_L(x) \oplus f^2_N(x) \\
f^3_{255}(x) =& f^3_L(x) \oplus f^3_N(x) \\
f^4_{255}(x) =& f^4_L(x) \oplus f^4_N(x)
\end{aligned} \tag{14}$$

If we increase 1 more bit width to produce 6 keystream bits in each clock, the additional function $f^5_{255}(x)$ should be:

$$f^5_{255}(x) = f^5_L(x) \oplus f^5_N(x) = x_{0+5} \oplus x_{12+5} \oplus x_{48+5} \oplus x_{115+5} \oplus x_{133+5} + f^0_{217}(x) + f^5_N(x) \tag{15}$$

Therefore, the 6 bits parallel width solution in Fibonacci configuration is shown in Figure 7. We can find that f^5_{255} variable includes one bit from f^0_{217}. Although this combining strategy will cause much propagation time with longer data path, we can implement the 8-bit and 16-bit parallel width Espresso-F solutions, shown in Table 8.

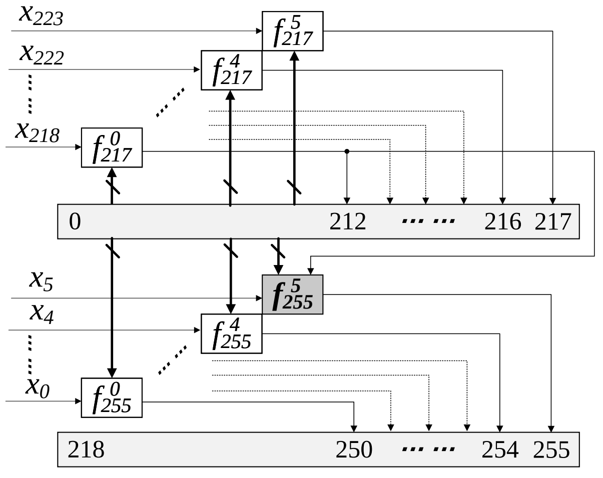


Figure 7: The block diagram of hybrid Espresso-F x6

It should be noted that the multiple bits generation strategy of Espresso-F is not *first update then filter*, but *first filter then update*, which means Espresso-F produces keystream bits z_0, z_1, ..., z_{w-1} directly at the first clock after initialization. Accordingly, w filter functions are listed as $h^0(x)$, $h^1(x)$, ..., $h^{w-1}(x)$.

Table 8: Implementation Results of Espresso-F Hybrid Architecture

Device	Width (bit)	#LUT /#FF	Area (Slices)	Freq. (MHz)	Thro. (Mbps)	T./A. (Mbps/Slices)
Virtex-7	1	64/81	22	427.2	427.2	19.42
	4	207/267	60	348.2	1392.8	23.21
	8	279/267	75	341.1	2728.5	36.38
	16	415/267	112	255.4	4085.8	36.48
Spartan-3	1	83/61	52	195.4	195.4	3.76
	4	356/267	197	134.2	536.8	2.73
	8	449/267	239	123.4	987.2	4.13
	16	641/267	332	118.7	1899.1	5.72

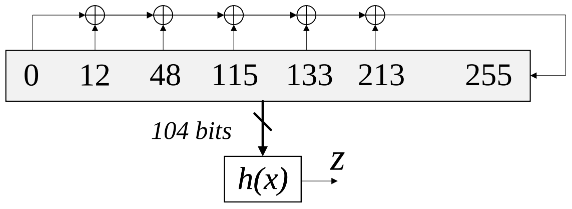


Figure 8: The block diagram of Fibonacci-configured Espresso-L

5 Description and Implementation of Espresso-like LFSR filter generator

We have introduced the implementation of Fibonacci-configured Espresso stream cipher in the last section. In this section, another transformation method from Galois NFSR to Fibonacci LFSR with compensation lists is briefly summarized, noted as Espresso-L.

5.1 Galois NFSR to Fibonacci LFSR Transformation

Another Espresso variant [15] is same as a LFSR filter generator, consists of 256-bit LFSR and 104-variables keystream filter function, named as Espresso-L, shown in Figure 8. After shifting all monomials to $f_{255}(x)$, the unique feedback function is linear function following:

$$f_{255}(x) = x_0 \oplus x_{12} \oplus x_{48} \oplus x_{115} \oplus x_{133} \oplus x_{213} \tag{16}$$

Denote $m|_d$ represents each variables x_i in monomial m are change to x_{i+d}. Compensation list is generated during monomial transformation from $f_a(x)$ to $f_b(x)$, $b = 255$ in Espresso-L configuration as Formula 17, where $p \in U$.

$$C_p = \begin{cases} 0, p \leq a \\ m|_{p-a-1}, p > a \end{cases} \tag{17}$$

There are totally 14 compensation lists because 14 feedback functions are transformed to $f_{255}(x)$. In order to achieve better comprehension, we note that $f_{255}(x)$ is also converted to itself

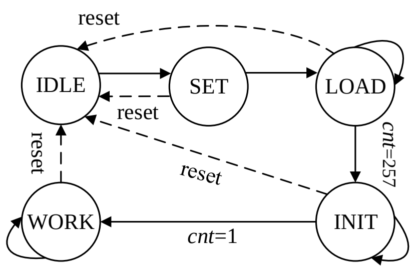


Figure 9: Espresso-L FSM transformation diagram

$f_{255}(x)$, while every element in list C_{255} is equal to 0. We combine 14 lists as:

$$C[i] = \sum_{p \in U} C_p[i] \tag{18}$$

The compensated internal state $\hat{x}_i$ is generated by $\hat{x}_i = x_i \oplus C[i]$. It's obviously that the compensated internal state $\hat{x}_i$, where $i \leq 193$, are same as x_i because compensation list $C[i]$ is empty.

The filter function $h(x)$ variables x_i are replaced by $\hat{x}_i$. And the 256 bits initial internal state before rounding are also loaded as $\hat{x}_i^0 (i = 0, 1, ..., 255)$.

$$\begin{aligned} h(x) = & x_{80} \oplus x_{99} \oplus x_{137} \oplus \hat{x}_{227} \oplus \hat{x}_{222} \oplus x_{187} \oplus \hat{x}_{243}\hat{x}_{217} \oplus \hat{x}_{247}\hat{x}_{231} \oplus \hat{x}_{213}\hat{x}_{235} \\ & \oplus \hat{x}_{255}\hat{x}_{251} \oplus x_{181}\hat{x}_{239} \oplus x_{174}x_{44} \oplus x_{164}x_{29} \oplus \hat{x}_{255}\hat{x}_{247}\hat{x}_{243}\hat{x}_{213}x_{181}x_{174} \end{aligned} \tag{19}$$

Until now, there are three Espresso modes, they are original Galois-configured Espresso, Fibonacci-configured Espresso-F and Fibonacci-configured LFSR filter generator Espresso-L.

5.2 Hardware Design of Espresso-L

Due to the influence of compensation list, the initial internal state $\hat{x}_i^0$ is changed to x_i^0, in especial, the 32 bits from $\hat{x}_i^{224}$ to $\hat{x}_i^{256}$ are no longer constant. There should be extra circuit realizing the compensation list and state update, so another state is added based on Galois Espresso finite state machine, SET state represents 256 bits internal state are XORed with $C[i]$ respectively after loading initial vector. The SET state lasts for one clock period before INIT state, shown in Figure 9.

The internal states from x_0 to x_{254} are updated following Formula 20, no longer just relying the more significant bit x_{i+1}. Hence, the SRL optimized is not available for Espresso-L, and each register (synthesized to flip-flop) should be driven by multiplex circuit (synthesized to look-up-table), which causes much more area.

$$x_i = \begin{cases} x_i \oplus C[i], \ state = SET \\ x_{i+1}, \ others \end{cases} \tag{20}$$

We implement the Espresso-L in both serial architecture and hybrid architecture, the results are listed in Table 9. It's obviously that Espresso-L requires much area for compensation list and leads to much critical path delay because of 104-variables filter function.

Table 9: Implementation Results of Espresso-L Variant

Device	Width (bit)	#LUT /#FF	Area (Slices)	Freq. (MHz)	Thro. (Mbps)	T./A. (Mbps/Slices)
Virtex-7	1	227/268	72	275.3	275.3	3.82
	4	509/268	143	271.9	1087.5	7.61
	8	572/268	153	269.0	2151.7	14.06
	16	735/268	197	208.2	3330.6	16.91
Spartan-3	1	550/268	303	113.8	113.8	0.38
	4	1104/268	573	113.4	453.6	0.79
	8	1209/268	624	98.2	785.2	1.26
	16	1462/268	756	86.9	1390.6	1.84

5.3 Security Analysis on Galois-to-Fibonacci Transformation

So far, there is no method that evinces active attacks on original Espresso, but Galois-to-Fibonacci transformation has been confirmed revealing possible security weakness [19] and [15]. Based on the transformation [14], another Fibonacci variant Espresso-A with two feedback function $f_{254}(x)$ and $f_{255}(x)$ has not 128-bit security level resistance, i.e. the 128-bit secret key can be recovered with only two pairs of related key-IVs, less than 2^{41} chosen IVs and $\mathcal{O}(2^{64})$ computational complexity [19]. Meanwhile, the LFSR filter generator variant Espresso-L may be broken with complexity $\mathcal{O}(2^{68.44})$ under algebraic attack and $\mathcal{O}(2^{66.86})$ under Ronjom-Helleseth attack [15]. However, they are sufficiently to be used in the tiny devices. Besides, our results have demonstrated that the Espresso-like LFSR filter generator cannot be implemented efficiently on hardware. Therefore, the LFSR variant Espresso-L is not recommended for ultra-lightweight cases.

6 Hardware Performance Comparison of Espresso and other ciphers

6.1 Comparison of Espresso and its variants

In this paper, in Table 10, we investigate the hardware performance of the stream ciphers Espresso and its two variants, who have the similar security level. Our optimizations and FPGA implementations aimed at evaluating the cipher's hardware performance in Galois and Fibonacci configuration, targeting for both high-speed and resource-constrained scenes.

The original Galois-configured Espresso has 14 bits internal state driven by the 14 feedback functions. The minimum distance between them is 4 bits, so the original Espresso can be upgraded to the parallel Espresso with 4 bits parallel width, which produces 4 bits keystream at each clock.

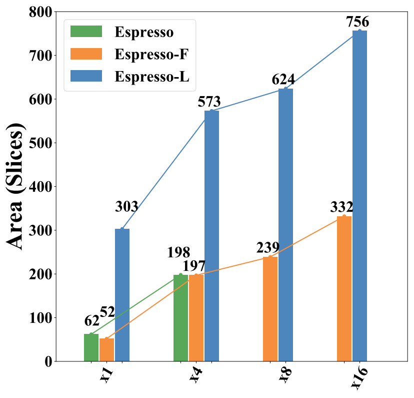
The Fibonacci-configured Espresso (Espresso-F) consisting of 2 Fibonacci NFSRs, are transformed [14] by the original Espresso. There are only 2 bits driven by the 2 feedback functions respectively. Espresso-F is not constrained by the maximum parallel width of 4 bits and we implement up to Espresso-F x16 to evaluate the throughput improvement strategy. The comparison between Galois-configured Espresso and Fibonacci-configured Espresso reveals which configuration is more efficient for various FPGA applications.

As shown in Figure 10, for the serial or low-width parallel solutions (i.e. x4), the Galois

Table 10: The Optimal Results of Espresso and its Variants on Spartan-3 FPGA

Cipher (Variant)	#LUTs /#FFs	Area (Slices)	Freq. (MHz)	Thro. (Mbps)	T./A. (Mbps/ Slices)	Pow. (mW)	T./P. (Mbps/ mW)	T./(P. · A.) (Mbps/ (mW · Slices))
Espresso x1 [Sec. 3]	90/80	62	198.5	198.5	3.20	2	99.2	1.60
Espresso x4 [Sec. 3]	371/267	198	163.3	653.3	3.30	11	59.4	0.30
Espresso-F x1 [Sec. 4]	83/61	52	195.4	195.4	3.76	2	97.7	1.88
Espresso-F x4 [Sec. 4]	356/267	197	134.2	536.8	2.73	11	48.8	0.25
Espresso-F x8 [Sec. 4]	449/267	239	123.4	987.2	4.13	18	54.8	0.23
Espresso-F x16 [Sec. 4]	641/267	332	118.7	1899.1	5.72	33	57.5	0.17
Espresso-L x1 [Sec. 5]	550/268	303	113.8	113.8	0.38	9	12.6	0.04
Espresso-L x4 [Sec. 5]	1104/268	573	113.4	453.6	0.79	19	23.9	0.04
Espresso-L x8 [Sec. 5]	1209/268	624	98.2	785.2	1.26	31	25.3	0.04
Espresso-L x16 [Sec. 5]	1462/268	756	86.9	1390.6	1.84	46	30.2	0.04
Espresso Best Item	-/-	52	198.5	1899.1	5.72	2	99.2	1.88

configuration has a higher throughput, but is larger than the Fibonacci configuration. The reason is that the split combinational logic in the Galois configuration do not lead to much logic level and route delay, but has to be synthesized in the extra look-up-tables.



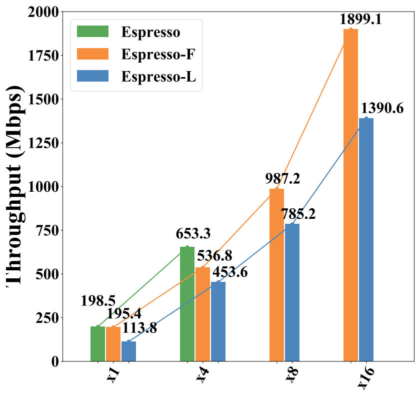


Figure 10: Area and throughput comparison between Espresso and its variants

Another variant is Fibonacci-configured Espresso 256-bit LFSR filter generator (Espresso-L), transformed [15] from nonlinear FSR to linear FSR. Although there is only one simplified linear feedback function, consisting of only 6 variables, much nonlinear feedback logic is shifted to filter function, forming a 104-variables keystream filter function. This kind of transformation does not increase the throughput or reduce the area, but is not conducive to hardware implementation. The aggregate index Throughput/Area (T./A.) of Espresso-L x1 is only 11.9% of Espresso x1 and 10.1% of Espresso-F x1. This kind of cipher containing linear FSR may only be adopted on specific programmable chips, which has coarse-grain reconfigurable linear feedback shift registers [26], [27], [28].

Table 11: The Optimal Results of Espresso and its Variants on Virtex-7 FPGA

Cipher (Variant)	#LUTs /#FFs	Area (Slices)	Freq. (MHz)	Thro. (Mbps)	T./A. (Mbps/ Slices)	Pow. (mW)	T./P. (Mbps/ mW)	T./(P.· A.) (Mbps/ (mW· Slices))
Espresso x1 [Sec. 3]	48/137	25	491.4	491.4	19.66	1	491.4	19.66
Espresso x4 [Sec. 3]	211/267	60	373.3	1493.1	24.88	4	373.3	6.22
Espresso-F x1 [Sec. 4]	64/81	22	427.2	427.2	19.42	1	427.2	19.42
Espresso-F x4 [Sec. 4]	207/267	60	348.2	1392.8	23.21	3	464.3	7.74
Espresso-F x8 [Sec. 4]	279/267	75	341.1	2728.5	36.38	7	389.8	5.20
Espresso-F x16 [Sec. 4]	415/267	112	255.4	4085.8	36.48	9	454.0	4.05
Espresso-L x1 [Sec. 5]	227/268	72	275.3	275.3	3.82	4	68.8	0.96
Espresso-L x4 [Sec. 5]	509/268	143	271.9	1087.5	7.61	6	181.3	1.27
Espresso-L x8 [Sec. 5]	572/268	153	269.0	2151.7	14.06	9	239.1	1.56
Espresso-L x16 [Sec. 5]	735/268	197	208.2	3330.6	16.91	16	208.2	1.06
Espresso Best Item	-/-	22	491.4	4085.8	36.48	1	491.4	19.66

Overall, in Figure 11, except Espresso-L, the Fibonacci variant support increase throughput by large area, so it is more acceptable for high-throughput application without considering lightweight design. Meanwhile, the Espresso-F x1 is better performance than the original Espresso x1 for compact devices, which does not need to process the large volume of data in a short time. The Galois-configured Espresso seems to be moderate, it is not suitable for straightforward high-throughput applications, nor for resource-limited devices, but a trade-off between the two factors. As a result, the Galois-configured cipher is able to balance speed and area.

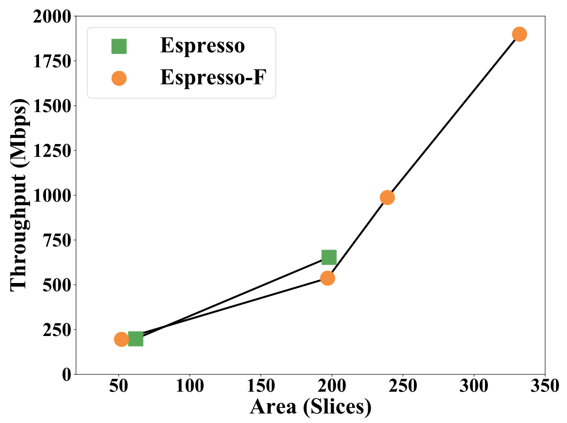


Figure 11: The hardware performance comparison among Espresso and Espresso-F

Besides, We list the optimal implementation results on Virtex-7 FPGA in Table 11 for Espresso and its variants with serial and all typical parallel widths optimizations. The minimum area of Espresso optimization on Virtex-7 FPGA only utilizes 22 slices and the highest throughput of Espresso in hybrid architecture produces keystream with more than 4 Gbps.

Table 12: Comparison with Other Stream Ciphers

Cipher (Variant)	Security Level (bits)	#LUTs /#FFs	Area (Slices)	Freq. (MHz)	Thro. (Mbps)	T./A. (Mbps/ Slices)	Devices Family	Device
Grain v1 [23]	80	-/-	44	196.0	196.0	4.45	S3	XC3S50
Grain v1 x16 [23]	80	-/-	348	130.0	2080.0	5.98	S3	XC3S50
MICKEY 2.0 [23]	80	-/-	115	233.0	233.0	2.03	S3	XC3S50
MICKEY 2.0 [24]	80	-/-	98	250.0	250.0	2.55	S3	XC3S700A
Trivium [23]	80	-/-	50	240.0	240.0	4.80	S3	XC3S50
Trivium [24]	80	-/-	149	326.0	326.0	2.19	S3	XC3S700A
WG-8 [21]	80	-/85	137	190.0	190.0	1.39	S3	XC3S1000
WG-8 x11 [21]	80	-/207	398	192.0	2112.0	5.31	S3	XC3S1000
E0 [24]	128	-/-	140	187.0	187.0	1.34	S3	XC3S700A
A5/1 [32]	128	-/-	57	174.0	174.0	3.05	S3	XC3S50
A5/1 x4 [32]	128	-/-	287	79.0	316.0	1.10	S3	XC3S50
ZUC [24]	128	-/-	1147	38.0	1216.0	1.06	S3	XC3S700A
Snow3g [24]	128	-/-	3559	104.0	3328.0	0.94	S3	XC3S700A
Grain v1 [22]	80	66/87	26	250.0	250.0	9.62	S7	XC7S50
Grain v1 x16 [22]	80	361/166	111	250.0	4000.0	36.03	S7	XC7S50
MICKEY 2.0 [22]	80	171214	51	250.0	250.0	4.90	S7	XC7S50
Trivium [22]	80	4932	22	385.0	385.0	17.50	S7	XC7S50
Lizard [22]	80	106/252	60	100.0	100.0	1.67	S7	XC7S50
Lizard x6 [22]	80	466/241	150	200.0	1200.0	8.00	S7	XC7S50
Espresso x1 [Sec. 3]	128	90/80	62	198.5	198.5	3.20	S3	XC3S50
Espresso x4 [Sec. 3]	128	371/267	198	163.3	653.3	3.30	S3	XC3S50
Espresso-F x1 [Sec. 4]	128	83/61	52	195.4	195.4	3.76	S3	XC3S50
Espresso-F x4 [Sec. 4]	128	356/267	197	134.2	536.8	2.73	S3	XC3S50
Espresso-F x8 [Sec. 4]	128	449/267	239	123.4	987.2	4.13	S3	XC3S50
Espresso-F x16 [Sec. 4]	128	641/267	332	118.7	1899.1	5.72	S3	XC3S50

6.2 Comparison of Espresso and other ciphers

Espresso stream cipher supports 128 bits secret key and 96 bits initial vector. The standardized cipher supporting 128-bit secret key include stream ciphers SNOW3G [9], ZUC [10] and block cipher AES [29]. In addition, the eStream portfolio lightweight stream ciphers with 80-bit secret key include Trivium [5], Grain [6] and MICKEY [7]. Although some of the other stream ciphers, such as A5/1 [30] for GSM and E0 [31] for Bluetooth, have been confirmed vulnerable to attacks [1], [2], [3], we take their implementation results as references to evaluate Espresso hardware adaptivity.

Table 12 compares our optimized Espresso implementations with other stream ciphers. Our best Espresso FPGA implementations can achieve 52 slices and perform more than 1.8 Gbps on Spartan-3. The results show that our optimized Espresso design on FPGA is the smallest and most efficient (evaluated according to T./A.) solutions among 128-bit secret key ciphers, and it is much smaller than MICKEY 2.0, less 10 slices larger than Grain and Trivium.

7 Conclusions

In this paper, we concluded three stream ciphers Espresso and its variants, original Galois-configured Espresso with 14 feedback functions, Fibonacci-configured Espresso consisting of 2 Fibonacci NFSRs and Fibonacci-configured Espresso 256-bit LFSR filter generator, to investigate which configuration for stream cipher is more efficient, when all of them are implemented under the optimal strategies toward area and throughput respectively.

For serial solution, we explored the smallest area architecture by adjusting the occupation ratio of FF to LUT and apply this strategy for all variants' implementations. To improve throughput, we designed the hybrid architecture without increasing the critical path delay significantly. After that, another strategy to improve throughput further was proposed, i.e., the higher feedback functions take the lower functions results as variables. This strategy caused much latency but improved throughput evidently.

According to our implementations, Fibonacci-configured Espresso FPGA architecture is smaller than that in Galois configuration, despite they both have the same quantity of logic gates, because several logic gates are packaged and synthesized as one 4-input or 6-input look-up-table. Under the premise of the equal parallel width, the Espresso hybrid architecture in Galois configuration has lower critical path propagation delay, which represents higher frequency than that in Fibonacci configuration. With regard to Espresso LFSR filter generator (Espresso-L), the variant not only has potential security weakness, but also inefficient in hardware implementation. The Espresso-L has a huge filter function with 104 variables, which lead to much combinational logic level and path delay.

The implementations of Espresso on Spartan-3 only take 52 slices under area optimized strategy in Fibonacci configuration and perform about 1.90 Gbps in hybrid architecture. Our optimal serial Espresso hardware scheme is smaller than most 128-bit secret key stream ciphers including ZUC and SNOW3G, even smaller than 80-bit secret key stream cipher MICKEY 2.0. Besides, our Espresso hardware design just occupies 22 slices on Virtex-7 FPGA at least, and parallelized design even performs more than 4 Gbps, satisfying compact design and high throughput demands.

To summarize, for the same series of ciphers, the transformation from nonlinear feedback shift register to linear feedback shift register do not improve security level, but is not suitable for hardware implementation. The Fibonacci configuration is smaller than the Galois configuration on FPGA applications, and the former can support the higher parallel width to improve throughput. When the device is not only resource-limited, but also slightly require high-throughput, i.e., the trade-off between area and speed, the Galois configuration is more worthy of recommendation.

As for the future work, we hope our analysis of Galois and Fibonacci configuration could provide reference for cipher hardware implementations, and our optimizations and throughput improvements' strategies could be used for more stream ciphers.

Acknowledgement

This work was supported by the Fundamental Research Funds of Shandong University under Grant 2019HW036.

References

[1] Eli Biham and Orr Dunkelman. Cryptanalysis of the A5/1 GSM stream cipher. In Bimal K. Roy and Eiji Okamoto, editors, *Progress in Cryptology - INDOCRYPT 2000, First*

International Conference in Cryptology in India, Calcutta, India, December 10-13, 2000, Proceedings, volume 1977 of *Lecture Notes in Computer Science*, pages 43–51. Springer, 2000.

[2] Yi Lu and Serge Vaudenay. Faster correlation attack on bluetooth keystream generator E0. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 407–425. Springer, 2004.

[3] Yi Lu and Serge Vaudenay. Cryptanalysis of bluetooth keystream generator two-level E0. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 483–499. Springer, 2004.

[4] Matthew Robshaw. The estream project. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2008.

[5] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier López, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

[6] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *IJWMC*, 2(1):86–93, 2007.

[7] Steve Babbage and Matthew Dodd. The MICKEY stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 191–209. Springer, 2008.

[8] 3GPP. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2; Document 1: UEA2 and UIA2 specifications. Technical Specification (TS) 35.215, 3rd Generation Partnership Project (3GPP), 6 2018. Version 15.0.0.

[9] 3GPP. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2; Document 2: SNOW 3G specification. Technical Specification (TS) 35.216, 3rd Generation Partnership Project (3GPP), 6 2018. Version 15.0.0.

[10] 3GPP. Specification of the 3GPP Confidentiality and Integrity Algorithms EEA3 & EIA3; Document 2: ZUC specification. Technical Specification (TS) 35.222, 3rd Generation Partnership Project (3GPP), 6 2018. Version 15.0.0.

[11] Yassir Nawaz and Guang Gong. WG: A family of stream ciphers with designed randomness properties. *Inf. Sci.*, 178(7):1903–1916, 2008.

[12] Yiyuan Luo, Qi Chai, Guang Gong, and Xuejia Lai. A lightweight stream cipher WG-7 for RFID encryption and authentication. In *Proceedings of the Global Communications Conference, 2010. GLOBECOM 2010, 6-10 December 2010, Miami, Florida, USA*, pages 1–6. IEEE, 2010.

[13] Xinxin Fan, Kalikinkar Mandal, and Guang Gong. WG-8: A lightweight stream cipher for resource-constrained smart devices. *EAI Endorsed Trans. Security Safety*, 2(3):e4, 2015.

[14] Elena Dubrova. A transformation from the fibonacci to the galois nlfsrs. *IEEE Trans. Inf. Theory*, 55(11):5263–5271, 2009.

[15] Ge Yao and Udaya Parampalli. Generalized NLFSR transformation algorithms and cryptanalysis of the class of espresso-like stream ciphers. *CoRR*, abs/1911.01002, 2019.

[16] Shohreh Sharif Mansouri and Elena Dubrova. An improved hardware implementation of the grain stream cipher. In Sebastián López, editor, *13th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, DSD 2010, 1-3 September 2010, Lille, France*, pages 433–440. IEEE Computer Society, 2010.

[17] Elena Dubrova and Martin Hell. Espresso: A stream cipher for 5g wireless communication systems. *Cryptogr. Commun.*, 9(2):273–289, 2017.

[18] Subhrajyoti Deb and Bubu Bhuyan. Performance evaluation of grain family and espresso ciphers for applications on resource constrained devices. *ICT Express*, 4(1):19 – 23, 2018. SI: CI & Smart Grid Cyber Security.

[19] Ming-Xing Wang and Dong Dai Lin. Related key chosen IV attack on stream cipher espresso variant. In *2017 IEEE International Conference on Computational Science and Engineering, CSE 2017, and IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2017, Guangzhou, China, July 21-24, 2017, Volume 1*, pages 580–587. IEEE Computer Society, 2017.

[20] Matthias Hamann, Matthias Krause, Willi Meier, and Bin Zhang. Design and analysis of small-state grain-like stream ciphers. *Cryptogr. Commun.*, 10(5):803–834, 2018.

[21] Gangqiang Yang, Xinxin Fan, Mark D. Aagaard, and Guang Gong. Design space exploration of the lightweight stream cipher WG-8 for fpgas and asics. In *Proceedings of the Workshop on Embedded Systems Security, WESS 2013, Montreal, Quebec, Canada, September 29 - October 4, 2013*, pages 8:1–8:10. ACM, 2013.

[22] Bohan Li, Meicheng Liu, and Dongdai Lin. FPGA implementations of grain v1, mickey 2.0, trivium, lizard and plantlet. *Microprocess. Microsystems*, 78:103210, 2020.

[23] David Hwang, Mark Chaney, Shashi Karanam, Nick Ton, and Kris Gaj. Comparison of fpga-targeted hardware implementations of estream stream cipher candidates. In *State of the Art of Stream Ciphers Workshop, SASC 2008*, pages 151–162, 2008.

[24] Paris Kitsos, Nicolas Sklavos, George Provelengios, and Athanassios N. Skodras. Fpga-based performance analysis of stream ciphers zuc, snow3g, grain v1, mickey v2, trivium and E0. *Microprocess. Microsystems*, 37(2):235–245, 2013.

[25] Mark Goresky and Andrew Klapper. Fibonacci and galois representations of feedback-with-carry shift registers. *IEEE Trans. Inf. Theory*, 48(11):2826–2836, 2002.

[26] K. N. Devika and R. Bhakthavatchalu. Design of reconfigurable lfsr for vlsi ic testing in asic and fpga. In *2017 International Conference on Communication and Signal Processing (ICCSP)*, pages 0928–0932, 2017.

[27] L. Shaer, T. Sakakini, R. Kanj, A. Chehab, and A. Kayssi. A low power reconfigurable lfsr. In *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, pages 1–4, 2016.

[28] L. Alaus, D. Noguet, and J. Palicot. A reconfigurable lfsr for tri-standard sdr transceiver, architecture and complexity analysis. In *2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*, pages 61–67, 2008.

[29] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 1999.

[30] Recommendation GSM ETSI. 02.09; security related network functions. Technical report, European telecommunications Standard Institute, ETSI, 1993.

[31] SIG Bluetooth. Specification of the bluetooth system-version 1.1 b, 2003.

[32] Kris Gaj, Gabriel Southern, and Ramakrishna Bachimanchi. Comparison of hardware performance of selected phase II estream candidates. State of the Art of Stream Ciphers Workshop, SASC 2007, eSTREAM, ECRYPT Stream Cipher Project, Report 2007/26.