情報保護學會論文誌
第11券 第3號, 2001. 6

FPGA/VHDL을 이용한 LILI-128 암호의 고속화 구현에 관한 연구

이 훈 재*, 문 상 재**

On a High-Speed Implementation of LILI-128 Stream Cipher Using FPGA/VHDL

Hoon-jae Lee*, Sang-jae Moon**

요 약

LILI-128 스트림 암호는 클럭 조절형 스트림 암호방식이며, 이러한 구조는 동기식 논리회로 구현시 속도가 저하되는 단점이 있다. 즉, 클럭 조절형인 LFSRd는 외부 클럭보다 1~4 배 높은 클럭을 요구하기 때문에 동일한 시스템 클럭 하에서는 데이터 전송속도에 따른 시스템 성능이 저하된다. 본 논문에서는 귀환/이동에 있어서 랜덤한 4개의 연결 경로를 갖는 4-비트 병렬 LFSRd를 제안하였다. 그리고 ALTERA 사의 FPGA 소자(EPF10K20RC240-3)를 선정하여 그래픽/VHDL 하드웨어 구현 및 타이밍 시뮬레이션을 실시하였으며, 50MHz 시스템 클럭에서 안정적인 50Mbps (즉, 45 Mbps 수준인 T3급 이상, 설계회로의 최대 지연 시간이 20ns 이하인 조건) 출력 수열이 발생될 수 있음을 확인하였다. 마지막으로, FPGA/VHDL 설계회로를 Lucent ASIC 소자 (LV160C, 0.13 μm CMOS & 1.5v technology)로 설계 변환 및 타이밍 시뮬레이션한 결과 최대 지연시간이 1.8ns 이하였고, 500 Mbps 이상의 고속화가 가능함을 확인하였다.

ABSTRACT

Since the LILI-128 cipher is a clock-controlled keystream generator, the speed of the keystream data is degraded in a clock-synchronized hardware logic design. Basically, the clock-controlled $LFSR_d$ in the LILI-128 cipher requires a system clock that is 1~4 times higher. Therefore, if the same clock is selected, the system throughput of the data rate will be lowered. Accordingly, this paper proposes a 4-bit parallel $LFSR_d$, where each register bit includes four variable data routines for feedback or shifting within the $LFSR_d$. Furthermore, the timing of the proposed design is simulated using a Max+plus II from the ALTERA Co., the logic circuit is implemented for an FPGA device (EPF10K20RC240-3), and the throughput stability is analyzed up to a rate of 50 Mbps with a 50MHz system clock. (That is higher than the T3 rate at 45 Mbps, plus the maximum delay routine in the proposed design was below 20ns.) Finally, we translate/simulate our FPGA/VHDL design to the Lucent ASIC device (LV160C, 0.13 μm CMOS & 1.5v technology), and it could achieve a throughput of about 500 Mbps with a 0.13 μm semiconductor for the maximum path delay below 1.8ns.

keyword : *LILI-128, 스트림 암호, 클럭 조절 발생기, 키 수열 발생기*

1. 서 론

암호 구현에 있어서 키 분배 또는 인증 기능이 요구되는 경우 공개 키 암호가 적용되지만, 데이터 암복호화 등 고속 처리가 요구되는 응용에는 스트림 암호나 블록 암호가 많이 사용된다. 블록 암호는 소

* 경운대학교 컴퓨터전자정보공학부(hjlee@kyungwoon.ac.kr)
** 경북대학교 전자전기공학부(sjmoon@knu.ac.kr)

프트웨어 구현이 용이한 반면 채널 에러 시 수신 단에서 블록크기만큼 에러가 확산되어 채널 효율(channel efficiency)이 떨어지며, 비도 수준에 대한 정량화가 불가능한 단점이 있다. 반면 스트림 암호는 에러 확산이 없고, 비도 수준에 대한 수학적 정량화가 가능하며, 하드웨어 구현이 용이하고, 통신 지연이 없으며, 고속 통신이 가능한 것 등의 잇 점으로 인해서 이동·무선통신 전송로 구간의 링크 암호 또는 군사·외교용으로 많이 사용되고 있다[1~3].

스트림 암호 알고리즘이란 이진화된 평문과 이진 키 수열의 배타적 논리합(XOR) 연산을 실행하여 암호문을 생성하는 알고리즘을 말하며, 이 때 출력 키 수열에 대한 특성과 발생 방법이 안전도에 직접적인 영향을 미친다. 스트림 암호 시스템 설계 시 고려 사항으로는 키수열 발생기에 대한 암호학적 안전성 (비도), 통신 채널 환경에 적합한 키 수열 동기방식 성능 (통신 신뢰성), 그리고 암호화 속도 등에 대한 분석이 필수적이다. 스트림 암호의 안전성은 여러 종류의 암호 공격에 대하여 얼마나 강한 키 수열을 발생시키느냐에 달려 있으며, 일반적으로 Beker[4], Siegen- thaler[5]와 Golic[6] 등이 제시한 아래의 기준을 따른다.

(1) 주기 (Period): 출력 키 수열은 주기에 대한 최소 값이 보장되어야 한다.
(2) 랜덤 특성 (Randomness): 출력 키 수열은 좋은 랜덤 특성을 가져야한다.
(3) 선형 복잡도 (Linear complexity): 출력 키 수열은 큰 선형 복잡도를 가져야 한다.
(4) 상관 면역도 (Correlation immunity): 출력 키 수열은 높은 상관 면역도를 갖는다.
(5) 키 수열 사이클 수 (Keystream cycle): 출력 키 수열은 1개 이상의 키 수열 사이클에서 발생되어야 한다.

70~90년대에 발표된 대표적인 키 수열 발생기로는 비메모리 형태의 Geffe 발생기[7]와 메모리 형태의 Rueppel 합산 수열 발생기 (summation generator)[8~10]를 들 수 있다. 이들 두 방식 모두 최대 주기를 갖을 뿐 아니라 랜덤 특성이 양호하고 구현이 용이한 장점이 있지만, Geffe 발생기는 선형 복잡도가 작고 상관 면역도가 없기(0) 때문에 상관성 공격(correlation attack)에 취약한 것으로 알려져 있다[11]. 그리고 Rueppel 발생기는 2개의 LFSR로 구성될 경우 선형 복잡도 및 상관 면역도가 거의 최대로 만족되지만, 연속되는 "0" 또는 "1"이 나타나는 특수한 출력에 대하여 Meier등[12]과 Dawson [13]의 상관성 공격에 취약하였으며, LM 발생기 형태[10] 또는 3개 이상의 LFSR로 구성될 때 상관성 공격으로부터 안전성을 보장받을 수 있게 된다.

최근에는 여러 종류의 클럭 조절(clock-controlled)형 키 수열 발생기가 제안된 바 있으며, 그 중에서 안전한 암호로는 유럽지역 디지털 셀룰러 폰 (GSM)의 표준 암호인 A5암호[14], Bilateral stop-and-go 발생기[15] 등이 있다. 이들과는 달리 미국 IBM의 SEAL[16]이나 RSA사의 RC4[17] 방식은 고속화에 초점을 둔 소프트웨어 형태의 암호 발생기이며, 비선형 FCSR(feedback with carry shift register)을 적용한 concoction 발생기[1]는 기존의 LFSR을 대체하는 발생기이다. 추후로는 각국마다 표준 스트림 암호를 설계하여 이동·무선 암호 등에 적용할 것으로 보이며, 특히 컴퓨터 네트워크의 초고속화로 인하여 비트 처리 형태의 스트림 암호와 블록 처리 형태의 블록 암호를 결합시킨 병렬형 스트림 암호 시스템 (parallel stream cipher system)[18] 등 안전성이 높으면서 고속화 실현이 가능한 알고리즘에 관심이 모일 것으로 기대된다.

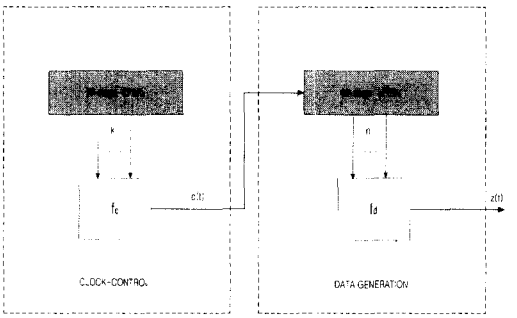
본 논문에서는 Simpson 등[19]이 제안한 LILI-128 암호에 대한 하드웨어 병렬 구현에 대하여 연구한다. LILI-128 암호는 유럽 GSM 표준암호인 A5에 이어 유럽지역 IMT-2000 표준암호 후보로 제안되고 있는 동기식 스트림 암호 알고리즘이다. 본 논문에서는 LILI-128 암호의 하드웨어 구현에 따른 구조적인 문제점을 발견하고 이를 해결함으로써 IMT-2000 표준암호로서의 병렬형 고속화 구현 방안에 대하여 접근코자 한다. LILI-128 암호는 128-비트 크기의 클럭 조절형 스트림 암호 방식[2~4]으로 이러한 구조는 동기식 논리회로 구현시 속도가 저하되는 단점이 있다. 즉, 클럭 조절형인 LFSRd는 외부 클럭보다 1~4 배 높은 클럭을 요구하기 때문에 동일한 시스템 클럭 하에서는 데이터 전송속도에 따른 시스템 성능이 저하된다. 본 논문에서는 귀환/이동에 있어서 랜덤한 4개의 연결 경로를 갖는 4-비트 병렬 LFSRd를 제안한다. 그리고 ALTERA[20]사의 FPGA 소자 (EPF10K20RC240-3)를 선정하여 그래픽/VHDL 하드웨어 구현 및 타이밍 시뮬레이션을 실시한 다음 50MHz 시스템 클럭에서 안정적인 50Mbps (즉, 45 Mbps 수준인 T3급 이

상, 설계회로의 최대 지연 시간이 20ns 이하인 조건) 출력 수열이 발생될 수 있음을 확인한다. 마지막으로, FPGA/VHDL 설계회로를 Lucent[21] ASIC 소자 (LV160C, 0.13μm CMOS & 1.5v technology)에 적합하게 설계 변환 및 시뮬레이션을 실시하여 최대 지연시간이 1.8ns 이하에서 500 Mbps 이상의 고속화가 가능함을 확인코자 한다.

Ⅱ. LILI-128 고속화 방안

LILI-128 암호의 구조는 그림 1과 같으며, 사용된 선형 귀환 이동 레지스터 (LFSR, linear feedback shift register)[5~7]는 39단 $LFSR_c$와 89단 $LFSR_d$로 구성되어 있는데, 이 중에서 $LFSR_d$는 $LFSR_c$의 출력에 의하여 클럭 통제를 받게 된다. 통제되는 클럭 수는 통상적인 경우 랜덤하게 설정된 f_c 함수에 의하여 생성된 정수 값 (1~4 범위) 만큼 LFSRd의 클럭을 이동시키며, 그 후 LFSRd의 내부 값으로부터 f_d 필터 함수를 통하여 필터 수열 (filtered sequence)[19]을 발생하게 된다.

LILI-128 암호에서 39단 $LFSR_c$, 89단 $LFSR_d$의 원시다항식 (primitive polynomial)[10~11] 및



〔그림 1〕 LILI-128 스트림 암호

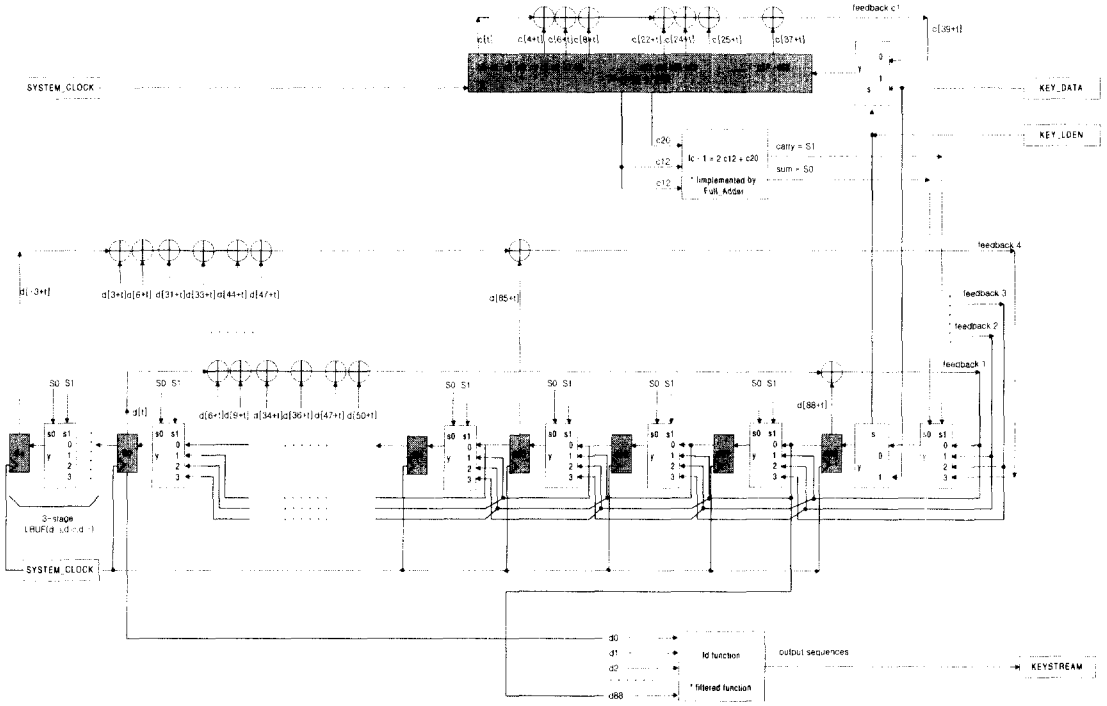
각각 좌측 이동 (left shift)될 귀환 비트 (feedback bit) 조합 입력인 $c[39+t]$, $d[89+t]$는 다음과 같이 정의된다.

$$g_c(x) = x^{39} + x^{35} + x^{33} + x^{31} + x^{17} + x^{15} + x^{14} + x^2 + 1$$

$$g_d(x) = x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{43} + x^{39} + x + 1$$

$$c[39+t] = c[37+t] \oplus c[25+t] \oplus c[24+t] \oplus c[22+t] \oplus c[8+t] \oplus c[6+t] \oplus c[4+t] \oplus c[t]$$

$$d[89+t] = d[88+t] \oplus d[50+t] \oplus d[47+t] \oplus d[36+t] \oplus d[34+t] \oplus d[9+t] \oplus d[6+t] \oplus d[t]$$



〔그림 2〕 LILI-128 고속 구현 방안 (4-bit parallel $LFSR_d$)

```
## Filename = lili-128.vhd

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity LILI is
port (
   CLK          : in  std_logic;
   KEY_LDEN     : in  std_logic;
   KEY_DATA     : in  std_logic;
   FEEDBACK1    : buffer std_logic;
   FEEDBACK2    : buffer std_logic;
   FEEDBACK3    : buffer std_logic;
   FEEDBACK4    : buffer std_logic;
   D88IN        : out std_logic;
   KEYSTREAM    : buffer std_logic
  );
end LILI;

architecture ARCH_LILI of LILI is

   component LFSRC39
   port (
      CLK       : in std_logic;
      KEY_DATA  : in std_logic;
      KEY_LDEN  : in std_logic;
      C12       : out std_logic;
      C20       : out std_logic
       );
   end component;

   component LFSRD89
   port (
      CLK     : in     std_logic;
      SEL     : in     unsigned(1 downto 0);
      KEY_LDEN  : in     std_logic;
      KEY_DATA  : in     std_logic;
      KEYSTREAM : out std_logic;
      D88IN   : out std_logic;
      FEEDBACK1 : buffer std_logic;
      FEEDBACK2 : buffer std_logic;
      FEEDBACK3 : buffer std_logic;
      FEEDBACK4 : buffer std_logic
     );
   end component;

   component FADD
   port (
      AIN    : in  std_logic;
      BIN    : in  std_logic;
      CIN    : in  std_logic;
      SUM    : out std_logic;
      COUT   : out std_logic
    );
   end component;

   signal C12 : std_logic := '0';
   signal C20 : std_logic := '0';

   signal SEL : unsigned(1 downto 0) := (others => '0');


begin

U_LFSRC : LFSRC39
        PORT MAP (
       CLK      => CLK,
       KEY_DATA => KEY_DATA,
       KEY_LDEN => KEY_LDEN,
       C12      => C12,
       C20      => C20
       );


U_LFSRD : LFSRD89
   port map(
      CLK       => CLK,
      SEL       => SEL,
      KEY_LDEN  => KEY_LDEN,
      KEY_DATA  => KEY_DATA,
      FEEDBACK1 => FEEDBACK1,
      FEEDBACK2 => FEEDBACK2,
      FEEDBACK3 => FEEDBACK3,
      FEEDBACK4 => FEEDBACK4,
      D88IN     => D88IN,
      KEYSTREAM => KEYSTREAM
     );

U_FADD : FADD
        PORT MAP (
       AIN  => C12,
       BIN  => C12,
       CIN  => C20,
       SUM  => SEL(0),
       COUT => SEL(1)
    );

end ARCH_LILI;
```

[그림 3] LILI-128 스트림 암호 구현 (FPGA/VHDL)

여기서 $LFSR_c$의 레지스터 비트 탭은 좌측으로부터 $c[0], c[1], \cdots, c[37], c[38]$, $LFSR_d$레지스터 비트 탭은 $d[0], d[1], \cdots, d[87], d[88]$로 각각 표기되며, $\oplus$는 배타 논리합인 XOR (exclusive-or) 연산을 의미한다.

LFSR의 하드웨어 구현 시에는 시스템의 안정성을 고려할 때 시스템 클럭에 맞추어 레지스터 값을 좌측 이동시키는 클럭 동기식 논리 설계 (clock-synchronous logic design) 방법이 일반적으로 많이 적용된다. 그러나 이 방법으로 LILI-128 암호를 구현함에 있어서 일반형인 $LFSR_c$는 상기의 방법으로 쉽게 구현될 수 있지만 (그림2, 3), 클럭 조절형인 $LFSR_d$는 1~4배의 고속 클럭이 별도로 요구된다. 또한 별도의 고속 클럭 추가문제를 해결하고자 주파수 채배기 (frequency multiplier)를 도입할 수도 있겠지만, 고속/초고속 통신에서는 클럭 간격(clock interval)에서의 시간 여유 (time margin)가 작기 때문에 적용이 어렵다.

LILI-128이 갖는 구조적인 문제를 해결하기 위하여 비트 이동 루트가 클럭을 초월하여 1~4 비트씩 가변적으로 이동할 수 있는 4-비트 병렬 입력 $LFSR_d$의 고속 구현 방안을 그림 2와 같이 제안한다.

그림 2의 상반부에 위치한 $LFSR_c$는 일반적인 39단 이동 레지스터 및 귀환 비트 조합으로 구현이 가능하다. 그리고 출력 f_c 회로는 $LFSR_d$의 좌측 이동 클럭 수를 결정하는 것으로서 전가산기 (full adder)를 사용하면 쉽게 구현된다. 그러나 89단 $LFSR_d$ 각 비트들은 $d_0, d_1, \ldots, d_{88}$로 나타낸 레지스터에 저장된 다음 f_c값에 따라 1~4-비트씩 이전 값 (우측 레지스터)으로부터 4-1 멀티플렉서 (4-1 MUX) 회로를 통하여 입력된다. 이 부분에 대한 설계 아이디어를 "4-비트 병렬 $LFSR_d$ (4-bit parallel $LFSR_d$)" 라고 부르며, 고속화 구현 회로의 핵심부분이다. 예를 들면, 그림에서 d_{84} 레지스터의 경우 그 이전 4개의 레지스터들 $d_{85}, d_{86}, d_{87}, d_{88}$중에서 랜덤하게 어느 한 입력이 선택 ($f_c=1$일 때는 d_{85}로부터, $f_c=4$일 때는 d_{88}로부터 각각 입력)되는데 이때 선택 신호들 ($s1, s0$)은 f_c로 구현된 전가산기의 출력으로부터 얻어진다. 그리고 $LFSR_d$의 좌측에는 3-비트 LBUF가 4개의 귀환 비트 조합을 계산하기 위하여 d_0의 출력을 차례로 보관하고 있다.

4개의 귀환 비트 조합 중에서 feedback 1은 원래의 귀환 비트와 동일한 탭의 XOR 조합을, feedback 2는 feedback 1에 비하여 1-비트씩 좌측 이동된 탭의 XOR 조합을, feedback 3는 2-비트씩 좌측 이동된 탭의 XOR 조합을, feedback 4는 3-비트씩 좌측 이동된 탭의 XOR 조합을 이룬다. 사용된 4개의 feedback 조합은 다음과 같이 표현된다.

$$\begin{aligned}
d[89+t] = {} & d[88+t] \oplus d[50+t] \oplus d[47+t] \\
& \oplus d[36+t] \oplus d[34+t] \oplus d[9+t] \\
& \oplus d[6+t] \oplus d[t] \quad : \textit{feedback } 1 \\
d[88+t] = {} & d[87+t] \oplus d[49+t] \oplus d[46+t] \\
& \oplus d[35+t] \oplus d[33+t] \oplus d[8+t] \\
& \oplus d[5+t] \oplus d[-1+t] \\
& : \textit{feedback } 2 \\
d[87+t] = {} & d[86+t] \oplus d[48+t] \oplus d[45+t] \\
& \oplus d[34+t] \oplus d[32+t] \oplus d[7+t] \\
& \oplus d[4+t] \oplus d[-2+t] \\
& : \textit{feedback } 3 \\
d[86+t] = {} & d[85+t] \oplus d[47+t] \oplus d[44+t] \\
& \oplus d[33+t] \oplus d[31+t] \oplus d[6+t] \\
& \oplus d[3+t] \oplus d[-3+t] \\
& : \textit{feedback } 4
\end{aligned}$$

마지막으로 LILI-128의 출력 수열은 그림 하단에 설정된 비선형 여과 함수 (nonlinear filter function) f_d로부터 얻어지는 비트 수열이 된다.

III. 설계 및 타이밍 시뮬레이션

제안된 고속화 병렬 구현 방안을 검증하기 위하여 그림 3의 회로를 설계하였으며, 이는 ALTERA사의 FPGA 소자 (EPF10K20RC240-3)를 선정하여 3개의 세부 블록으로 구현시킨 VHDL (Very high speed integrated circuit Hardware Description Language) 설계 회로이다. 그림에서 component LFSRC39 (LILI-c39)는 $LFSR_c$를 구현한 것으로서 그 세부 설계는 그림 4와 같다. component FADD (full_adder)에서는 $f_c = 2c[12] + c[20] + 1$의 함수를 구현한 것이며, 출력은 $LFSR_d$에 대한 클럭 이동 비트를 산출하는 함수로서 임의의 정수 값을 출력한다. 이 부분에 대한 설계는 $f_c - 1$을 사전 계산한 후 전 가산기 (full adder)로 간단히 구현하였으며, 계산 결과는 $LFSR_d$로 전달된다. $LFSR_d$에 대한 설계는 그림 3의 component LFSRD89에 나타내었으며, 그 세부 설계는 그림 5

```
## Filename = lili-c39.vhd

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity LFSRC39 is
port (
    CLK       : in std_logic;
    KEY_DATA  : in std_logic;
    KEY_LDEN  : in std_logic;
    C12       : out std_logic;
    C20       : out std_logic
    );
end LFSRC39 ;

architecture ARCH_LFSRC39  of LFSRC39  is

    signal C        : unsigned(38 downto 0);
    signal TMP_FEED : std_logic := '0';
    signal FEED_IN  : std_logic := '0';
```

```
begin

C12 <= C(12);
C20 <= C(20);

FEED_IN <= C(0) xor C(4) xor C(6) xor C(8) xor
          C(22) xor C(24) xor C(25) xor C(37);

process(CLK)
begin
    if CLK'event and CLK = '1' then
        C <= TMP_FEED & C(38 downto 1);
    end if;
end process;

process(CLK)
begin
    if CLK'event and CLK = '1' then
        if KEY_LDEN = '0' then
            TMP_FEED <= FEED_IN;
        else
            TMP_FEED <= KEY_DATA;
        end if;
    end if;
end process;

end ARCH_LFSRC39 ;
```

〔그림 4〕 LILI-128에 사용된 $LFSR_c$구현 (FPGA/VHDL)

(LILI-d89)와 같다.

그림 6은 LILI-128에 대한 타이밍 시뮬레이션 파형이다. 사용된 시스템 클럭 (SYSTEM_ CLOCK)은 50 MHz 이며, 리셋 신호 (/MASTER_RESET)에 이어 키 데이터 (KEY_DATA)로 초기화된 $LFSR_c$ 출력 신호 (LFSRc37_OUTPUT, LFSRc38_OUT)와 $LFSR_d$ 출력 신호 (LFSRd88_OUT)가 클럭에 맞추어 50 Mbps의 속도를 낸다. 이 회로에서 사용된 최장길이의 지연시간은 18ns 였기 때문에 50 MHz 클럭에 대하여 안정적인 50 Mbps의 출력을 낼 수 있게 된다.

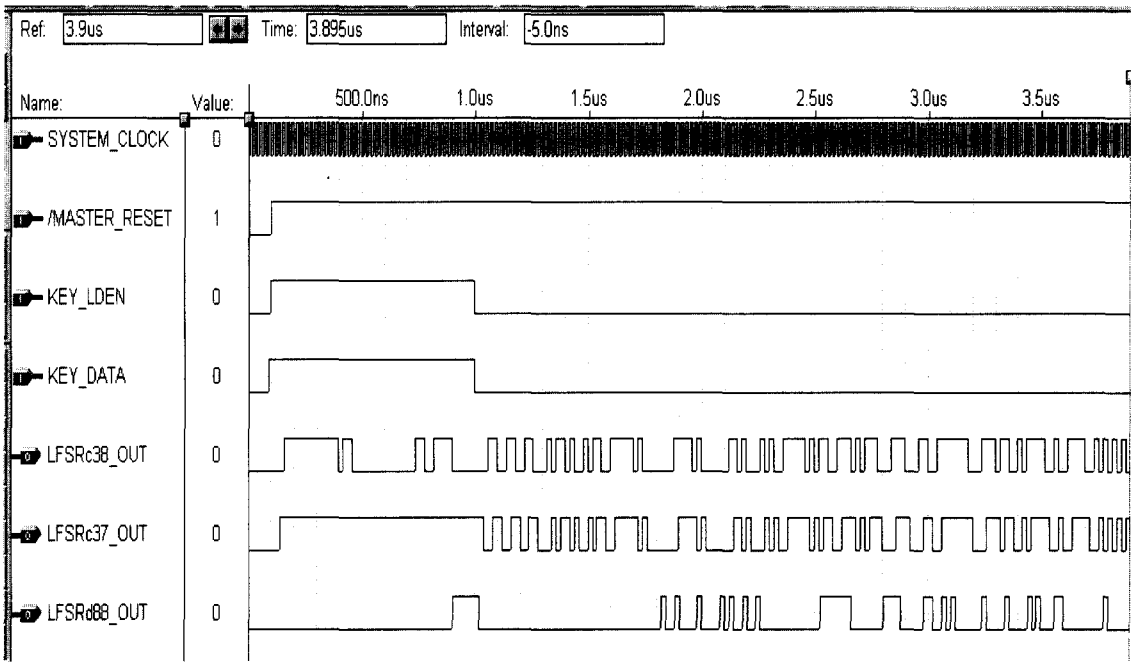
마지막으로, 상기 FPGA/VHDL 설계 회로를 Lucent[21] ASIC 소자 (LV160C, 0.13μm CMOS & 1.5v technology)에 적합하게 설계 변환 및 시뮬레이션한 결과 최대 지연시간이 1.8ns 이하였고, 500 Mbps 이상의 고속화가 가능함을 확인하였다. 표 1에서는 제안된 병렬 하드웨어 구현 방안을 기존의 구현방안과 비교하였는데 하드웨어 복잡도 측면에서 소규모 증가가 예상되지만 키 수열 발생기의 안전성을 유지하면서도 그 성능을 최대 4배까지 고속화시킬 수 있었다. 즉, 게이트 수로 살펴본 하드웨어 복잡도가 2.5배 증가되었으며, 출력 키 수열이 최대한 4배의 속도 향상이 가능함을 확인하였다. 마지막으로 50 MHz 시스템 클럭을 인가한 경우 안정적인 50 Mbps의 키 수열 출력을 낼 수 있음을 FPGA 소자를 통하여 확인할 수 있었다. 초고속 암호 통신을 위한 ASIC 설계 변환 및 시뮬레이션에서는 상기 FPGA의 성능을 10배정도 향상이 가능하였기 때문에 500 Mbps의 속도가 가능함을 확인하였다. 이 보다 더 높은 통신 속도를 위해서는 참고문헌[19]에서 제시된 바와 같이 개별 LFSR에 대하여 각각 병렬형 스트림 암호의 적용이 요구된다.

IV. 결 론

LILI-128 스트림 암호는 클럭 조절형 스트림 암호 방식으로서 이러한 형태는 동기식 논리회로에 따른 하드웨어 구현에 있어서 속도를 떨어뜨리는 구조적

```
## Filename = lili-d89.vhd

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity LFSRD89 is
port (
    CLK       : in     std_logic;
    SEL       : in     unsigned(1 downto 0);
    KEY_LDEN  : in     std_logic;
    KEY_DATA  : in     std_logic;
    KEYSTREAM : out std_logic;
    D88IN     : out std_logic;
    FEEDBACK1 : buffer std_logic;
    FEEDBACK2 : buffer std_logic;
    FEEDBACK3 : buffer std_logic;
    FEEDBACK4 : buffer std_logic
  );
end LFSRD89;

architecture ARCH_LFSRD89 of LFSRD89 is

    component DMUX41
    port (
        CLK  : in     std_logic;
        SEL  : in     unsigned(1 downto 0);
        DIN  : in     unsigned(3 downto 0);
        DOUT : out std_logic
         );
    end component;

    component DSMUX41
    port (
        CLK    : in     std_logic;
        S_EN   : in     std_logic;
        KEY_EN : in     std_logic;
        SEL    : in     unsigned(1 downto 0);
        DIN    : in     unsigned(3 downto 0);
        D88IN  : out std_logic;
        DOUT   : out std_logic
         );
    end component;

    signal D      : unsigned(88 downto 0) :=
(others => '0');
    signal LBUF : unsigned(2 downto 0)  :=
(others => '0');
    signal TMP1_FEED : unsigned(3 downto 0)
:= (others => '0');
    signal TMP2_FEED : unsigned(3 downto 0)
:= (others => '0');
    signal TMP3_FEED : unsigned(3 downto 0)
:= (others => '0');
    signal TMP4_FEED : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF84 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF83 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF82 : unsigned(3 downto
0)    := (others => '0');
    signal TMP_BUF81 : unsigned(3 downto
0)    := (others => '0');
    signal TMP_BUF80 : unsigned(3 downto
0)    := (others => '0');
    signal TMP_BUF79 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF78 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF77 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF76 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF75 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF74 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF73 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF72 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF71 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF70 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF69 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF68 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF67 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF66 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF65 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF64 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF63 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF62 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF61 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF60 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF59 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF58 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF57 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF56 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF55 : unsigned(3 downto 0)
:= (others => '0');
    signal TMP_BUF54 : unsigned(3 downto 0)
:= (others => '0');

    ..................
```

〔그림 5〕 LILI-128에 사용된 LFSRd의 설계 구현 (FPGA/VHDL)



〔그림 6〕 LILI-128 타이밍 시뮬레이션 파형 (50MHz 시스템 클럭 사용)

〔표 1〕 LILI-128의 구현 방안 비교표

Items	In general	Proposed
Hardware implementation	Clock-synchronized logic general implementation.	Multiple configured. (4-bit parallel LFSRd implemented)
Throughput of data rate for ALTERA FPGA device (EPF10K20RC240-3) with 50MHz system clock	12.5~50 Mbps variable rate.	50 Mbps fixed rate: (maximum four times higher) - Maximum path delay about 18 ns
Throughput of data rate for Lucent ASIC device (LV160C, 0.13μm, 1.5v technology) with 500MHz system clock	-	500 Mbps fixed rate: - Maximum path delay about 1.8 ns.
Number of gates used (if 1 F/F=5 AOI gate)	688 gates: - 128 D flip-flops - 2 (2-1) MUXs - 14 XORs	1,745 gates (about 2.5 times): - 131 D flip-flops - 89 (4-1) MUXs - 2 (2-1) MUXs - 35 XORs

〔Note〕 AOI gate : And-Or-Inverter gate

인 문제점을 안고 있다. 본 논문에서는 이러한 속도 저하의 문제를 해결하는 LILI-128 스트림 암호의 고속화 구현 방법을 연구하여 하드웨어 구현에 따른 구조적인 문제점을 보완하였고, 그 결과 기존의 구현 방안에 비하여 최대 4배까지 고속화가 가능함을 확인하였다. 또한 하드웨어 설계 검증을 위하여 ALTERA 사의 Max+plus II로 타이밍 시뮬레이션을 실시하였고, FPGA소자(EPF10K20RC240-3)를 선정하여 하드웨어로 구현하였다. 구현된 회로는 50MHz 시스템 클럭에서 안정적인 50Mbps

출력 수열이 발생될 수 있음을 확인하였다. 마지막으로, FPGA/VHDL 설계회로를 Lucent ASIC 소자 (LV160C, 0.13μm CMOS & 1.5v technology)에 적합하게 설계 변환 및 시뮬레이션한 결과 최대 지연시간이 1.8ns 이하였기 때문에 500 Mbps 이상의 고속화가 가능함을 확인하였다. 이보다 더 높은 통신 속도를 위해서는 병렬형 스트림 암호의 적용이 요구된다.

참 고 문 헌

[1] B. Schneier, *Applied Cryptography (2nd edition)*, John Wiley & Sons, Inc., 1996.

[2] A.J. Menezes, P.C. Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[3] D.R. Stinson, *Cryptography - Theory and Practice*, CRC Press, 1995.

[4] H. J. Beker and F. C. Piper, *Cipher systems: The Protection of Communications*, Northwood Books, London, 1982.

[5] T. Siegenthaler, "Decrypting a Class of Stream Ciphers Using Ciphertext Only," *IEEE Trans. on Computer*, C-34 (1), pp. 81-85, Jan. 1985.

[6] J. D. Golic, "The Number of Output Sequences of a Binary Sequence Generator," *LNCS 547, Advances in Cryptology-EUROCRYPT'91*, pp. 160-167, 1991.

[7] P. R. Geffe, "How to Protect Data with Ciphers that are really hard to Break," *Electronics*, pp. 99-101, Jan. 1973.

[8] R. A. Rueppel, "Correlation Immunity and the Summation Generator," *Advances in Cryptology, Proceedings of CRYPTO'85*, pp. 260-272, 1985.

[9] R. A. Rueppel, Analysis and Design of Stream Ciphers, Springer-Verlag, 1986.

[10] Hoonjae Lee, Sangjae Moon, "On An Improved Summation Generator with 2-Bit Memory," *Signal Processing*, 80(1), pp. 211~217, Jan. 2000.

[11] T. Siegenthaler, " Decrypting a Class of Stream Ciphers Using Ciphertext Only," *IEEE Trans. on Computer*, C-34(1), pp. 81-85, Jan. 1985.

[12] W. Meier and O. Staffelbach, "Correlation Properties of Combiners with Memory in Stream Ciphers," *Journal of Cryptology*, Vol. 5, pp. 67-86, 1992.

[13] E. Dawson, "Cryptanalysis of Summation Generator," *Advances in Cryptology-AUSCRYPT'92, LNCS*, Springer-Verlag, pp. 209-215, 1993.

[14] S.B. Xu, D.K. He, and X.M. Wang, "An Implemenatation of the GSM General Data Encryption Algorithm A5," *CHINACRYPT'94*, Xidian, China, 11-15 Nov. 1994, pp. 287-291.

[15] K.C.Zeng, C.H. Yang, and T.R.N. Rao, "On the Linear Consistency Test (LCT) in Cryptanalysis with Applications," *Advances in Cryptology-CRYPTO'89 Proceedings*, Springer-Verlag, 1990, pp. 164-174.

[16] P.Rogaway and D. Coppersmith, "A Software-Oriented Encryption Algorithm," Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 56-63.

[17] R.L. Rivest, "The RC4 Encryption Algorithm," *RSA Data Security*, Inc., Mar. 1992.

[18] 이훈재, 문상재 "고속 안전 통신을 위한 병렬형 스트림 암호," *한국통신학회 논문지*, 2001년 5월.

[19] L. Simpson, E. Dawson, J. Dj. Golic and W. Millan, "LILI Keystream Generator," *Proceedings of the Seventh Annual Workshop on Selected Areas in Cryptology SAC'2000* to appear in Springer-Verlag LNCS, 2000.

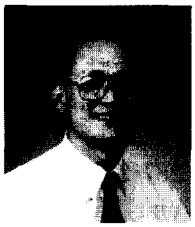
[20] Altera technical data sheets in http://www.altera.com.

[21] Lucent technical data sheets in http://www.lucent.com.

〈著者紹介〉

이 훈 재 (Hoon-jae Lee) 정회원

1985년 2월 : 경북대학교 전자공학과 졸업(공학사)
1987년 2월 : 경북대학교 전자공학과 졸업(정보통신공학, 공학석사)
1998년 2월 : 경북대학교 전자공학과 졸업(정보통신공학, 공학박사)
1987년 2월~1998년 1월 : 국방과학연구소 선임연구원
1998년 2월~현재 : 경운대학교 컴퓨터전자정보공학부 조교수
〈관심분야〉 암호이론, 네트워크보안, 디지털 통신



문 상 재 (Sang-jae Moon) 정회원

1972년 2월 : 서울대학교 공업교육과 졸업(전자공학전공, 공학사)
1974년 2월 : 서울대학교 대학원 전자공학과 졸업(공학석사)
1984년 6월 : 미국 UCLA 전기공학과 졸업(통신공학전공, 공학박사)
1984년 7월~1985년 6월 : UCLA Postdoctor 근무
1974년 12월~현재 : 경북대학교 공과대학 전자전기공학부 교수
2000년 8월~현재 : 경북대학교 이동네트워크 정보보호기술 연구센터 소장
2001년 2월~현재 : 한국정보보호학회 회장
〈관심분야〉 정보보호, 이동 네트워크