8-bit AVR 상에서 경량암호 Romulus의 Linear Feedback Shift Register 최적 구현

커피동아리 권혁동

서론

경량암호 Romulus

제안 기법

성능 평가 및 결론

서론

- 사물인터넷 기술이 발전하면서 **무선 보안의 중요성**이 대두됨!
  - 사물인터넷 장비는 가용 자원이 제한적
  - 대부분의 암호는 연산량이 매우 많음
  - 사물인터넷 장비 상에서 **일반적인 암호를 가동하기 곤란함**

- 사물인터넷 환경에 최적화된 **경량암호**의 등장
  - NIST에서는 경량암호 **표준화를 위한 공모전**을 개최

- 공모전 출품 작품 중 **Romulus의 모듈**을 최적 구현
  - **Linear Feedback Shift Register**
  - AVR 프로세서 상에서 최적 구현

경량암호 Romulus

- 2018년 NIST 경량암호 공모전을 개최하여, **2022년 최종 라운드**에 진입
  - 10종의 후보 알고리즘 중 Romulus는 유일한 Tweakable block cipher 기반의 암호

기반	알고리즘	해시 지원	코어 함수
Stream cipher	Grain-128 AEAD	X	Grain-128a
Block cipher	GIFT-COFB	X	GIFT-128
Tweakable block cipher	**Romulus**	**O**	**SKINNY-128-256** **SKINNY-128-384**
Permutation	ASCON	O	ASCON-320
	ISAP	X	Keccak-400 ASCON-320
	PHOTON-Beetle	X	Spongent-160/176 Keccak-200
	SPARKLE	O	Sparkle-256/384/512
	TinyJAMBU	X	JAMBU-128
	Xoodyak	O	Xoodoo-384

경량암호 Romulus

- Romulus는 SKINNY 블록 암호를 사용하는 경량암호
  - 유일한 **Tweakable block cipher** 기반의 후보
- Romulus의 종류
  - **Romulus-N**: 논스 기반의 Authenticated Encryption(AE)
  - Romulus-M: Misuse 공격에 내성을 지니는 변형
  - Romulus-T: TEDT의 개선 알고리즘
  - **Romulus-H**: 제 2역상 저항성 해시
- Romulus의 카운터 크기는 **56-bit**
  - 카운터 생성에 **Linear Feedback Shift Register**를 사용

Parameter	Romulus-N, M, T	Romulus-H
Nonce length	128-bit	-
Block length	128-bit	256-bit
Key length	128-bit	-
Counter bit length	56-bit	-
Tag length	128-bit	-
Hash length	-	256-bit

경량암호 Romulus

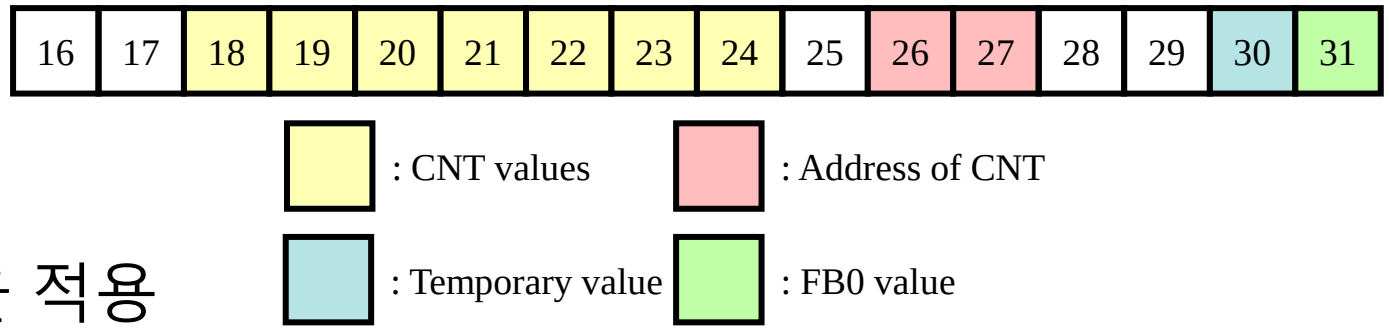
- Linear Feedback Shift Register(LFSR)
  - 일종의 시프트 레지스터
  - 이전 상태 값의 선형 함수로 출력 값을 계산하는 레지스터
  - **결정론적 알고리즘**
  - **의사 난수(Pseudo random), 의사 난수 잡음(PRN) 생성** 등에 사용
- Romulus는 카운터 생성을 위해 56-bit LFSR 사용
  - $F_{56}(x) = x^{56} + x^7 + x^4 + x^2 + 1, \mathrm{lfsr}_{56}(D) = 2^D \bmod F_{56}(x)$

$$z_i \leftarrow for\ i \in [[56]]_0 \setminus i = \{7,4,2,0\}$$
$$z_7 \leftarrow z_6 \oplus z_{55}$$
$$z_4 \leftarrow z_3 \oplus z_{55}$$
$$z_2 \leftarrow z_1 \oplus z_{55}$$
$$z_0 \leftarrow z_{55}$$

제안 기법

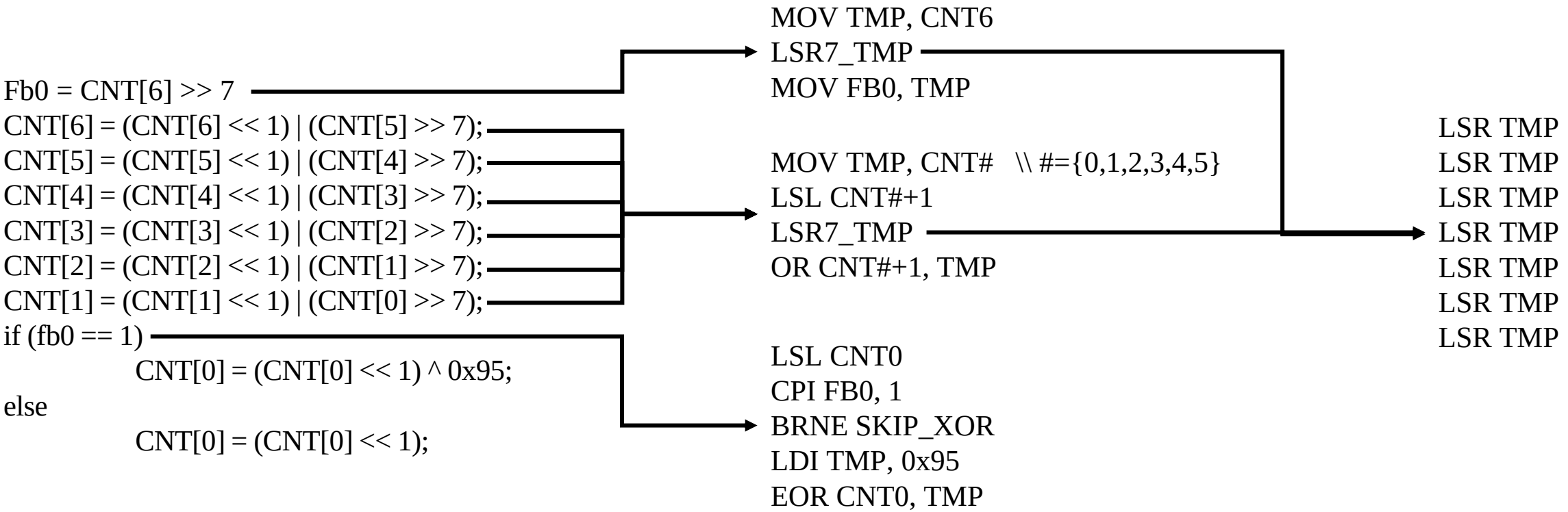
- Romulus의 LFSR을 8-bit AVR 프로세서 상에서 최적 구현을 제안
  - 제안 기법은 2가지 방식이 존재
  - **Shift를 사용한 구현**
  - **Bit store를 사용한 구현**
- 기존 Romulus의 LFSR은 다음과 같이 구현이 가능
- 기존 코드를 **AVR assembly를 사용**하여 최적 구현
- 레지스터 할당
  - R0~15: 사용하지 않음
  - R18~24: CNT0~7
  - R26, 27: CNT 배열의 주소
  - R30: 임시 변수
  - R31: FB0
  - **두 방식 모두 같은 할당 계획**을 적용

```
Fb0 = CNT[6] >> 7
CNT[6] = (CNT[6] << 1) | (CNT[5] >> 7);
CNT[5] = (CNT[5] << 1) | (CNT[4] >> 7);
CNT[4] = (CNT[4] << 1) | (CNT[3] >> 7);
CNT[3] = (CNT[3] << 1) | (CNT[2] >> 7);
CNT[2] = (CNT[2] << 1) | (CNT[1] >> 7);
CNT[1] = (CNT[1] << 1) | (CNT[0] >> 7);
if (fb0 == 1)
        CNT[0] = (CNT[0] << 1) ^ 0x95;
else
        CNT[0] = (CNT[0] << 1);
```



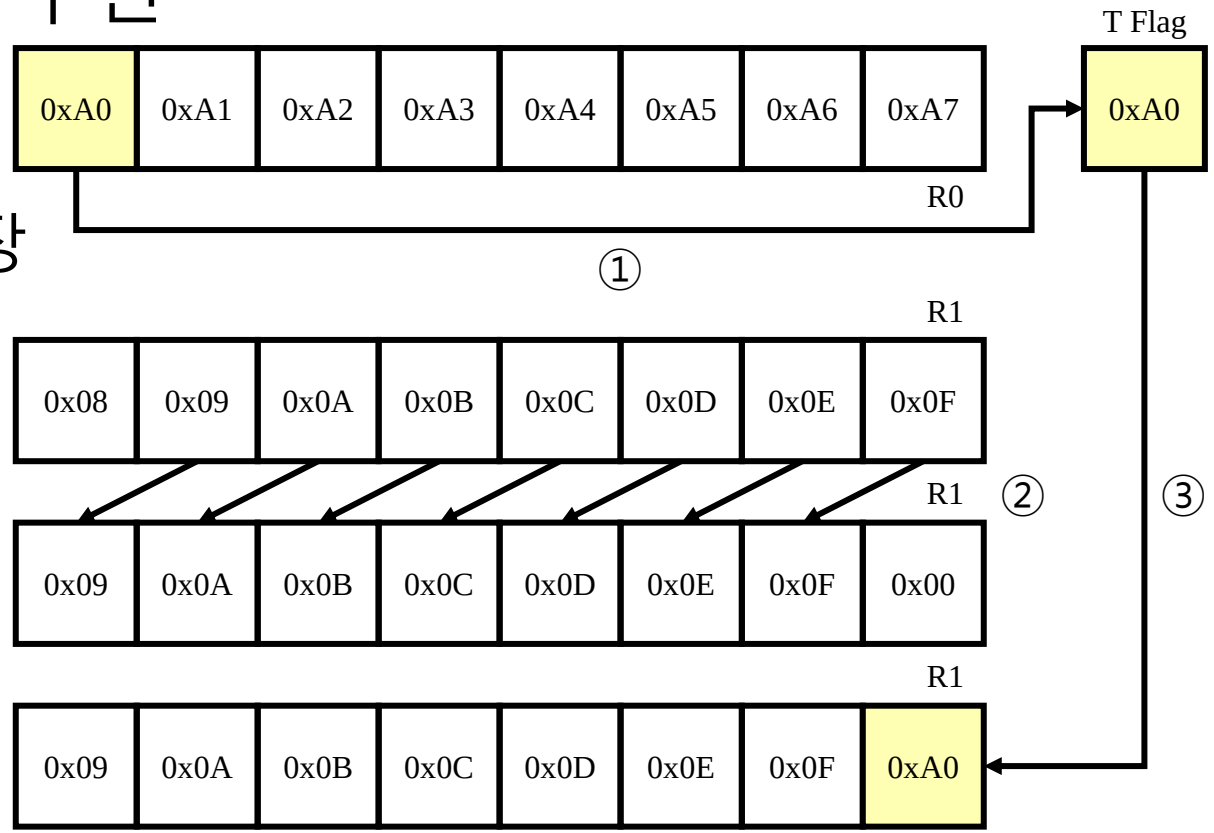
제안 기법

- Shift를 사용한 구현
  - 각 단계를 AVR assembly의 **Shift 명령어를 사용**하여 구현
  - 구현의 편의를 위해 매크로를 부분 활용
  - **Shift 횟수가 많을 수록 사용할 명령어의 수가 늘어남**



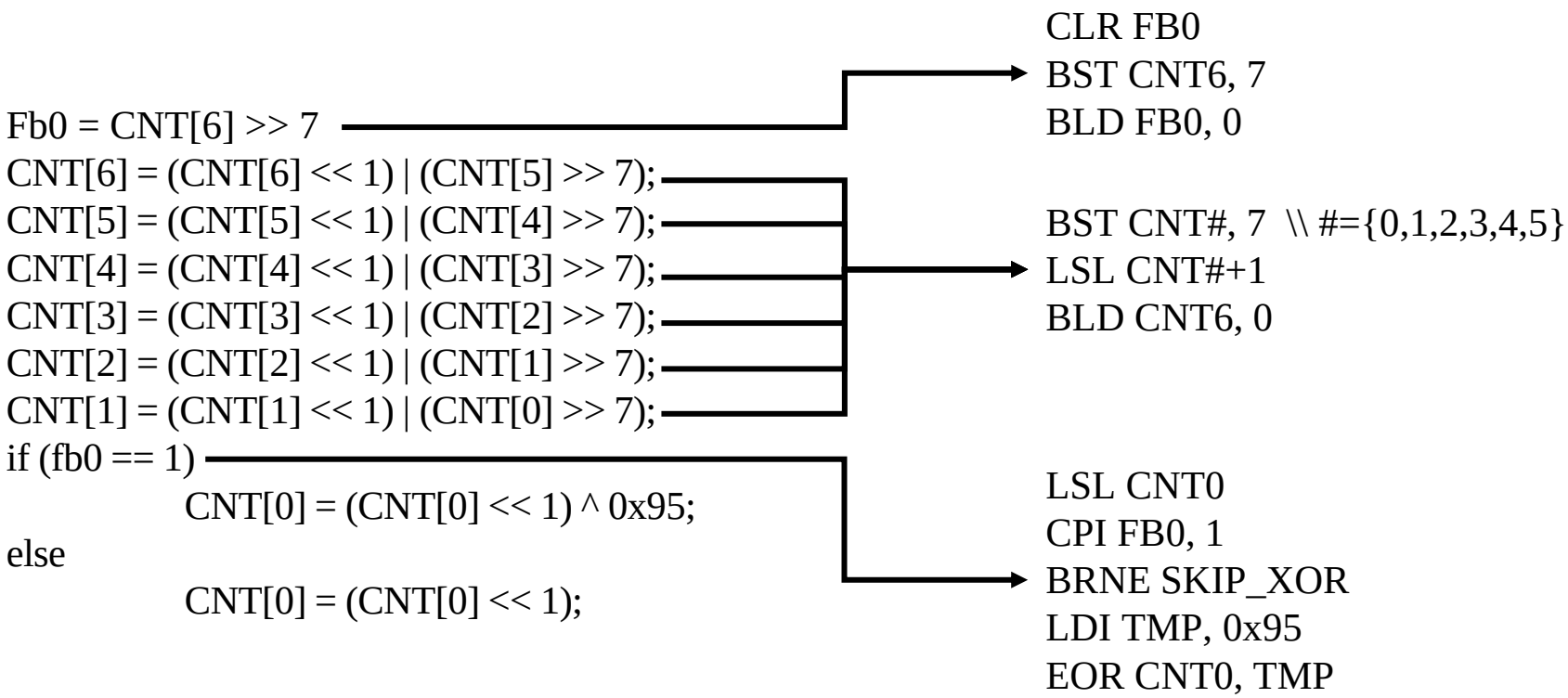
제안 기법

- Bit store를 사용한 구현
  - AVR의 **flag를 사용**한 구현
  - 레지스터 값을 shift하는 대신, flag를 사용하여 1비트 이동시키는 방법
- Right shift 7회를 Bit store로 대체하여 구현
  1. CNT#의 가장 좌측 값을 T flag에 저장
  2. CNT#+1을 Left shift
  3. T flag에서 값을 CNT#+1의 우측에 저장
  - **그림과는 달리 실제 값은 0 또는 1**



제안 기법

- Bit store를 사용한 구현
  - AVR의 **flag를 사용**한 구현
  - 레지스터 값을 shift하는 대신, flag를 사용하여 1비트 이동시키는 방법



성능 평가

- 구현 환경
  - Microchip Studio Framework
  - ATmega128 processor
  - -O3(fastest) compile option
  - Unit: clock cycles
- 비교 결과
  - Shift version: 레퍼런스 구현물 대비 30% 저조
  - **Bit store version: 레퍼런스 구현물 대비 23% 향상**
  - Shift 구현물은 기존 대비 너무 많은 shift 횟수로 효율이 떨어짐을 확인

Reference	Proposed implementation Shift version	Proposed implementation Bit store version
79	112	**64**

결론

- Romulus의 **LFSR을 AVR 프로세서 상에서의 최적 구현**
  - 구현물은 **Shift와 Bit store를 사용한 두 가지**로 제시
- 성능 측정 결과, 레퍼런스 구현보다 떨어지는 경우가 존재
  - **Shift 구현물의 경우 비효율적인 shift 횟수로 성능이 떨어짐**
- 구현 시에 알고리즘의 특성을 활용할 필요함
  - C코드를 Assembly로 옮기는 것으로 **무조건적인 성능 향상을 기대할 수 없음**
- 추가적인 부분을 구현하여 Romulus 전체를 최적 구현