Multi-precision for NPU

: SHA1, SHA512, Speck32/64

https://youtu.be/2_KgYRzq1hA

Overview

- **이전에 보고 드렸던 내용 기반으로 과제 진행 중...**
  - 싱가포르 다녀오고.. 8월 말쯤 시작해서.. 9월부터 빨리 해보겠습니다.

- **NPU에서 지원하는 연산자 사용**
  - WARBOY, ATOM에서 지원하는 연산자 테스트 (WARBOY (2차 발표) + ATOM (완료, 3차에 넣을 예정))

- **NPU에서 사용 가능한 자료형 고려**
  - WARBOY : 16/8-bit
  - ATOM : 8-bit

- **Multi-precision을 적용하여 블록 암호 구현**
  - **우선 8/4-bit 조합 사용** → 변수 숫자만 바꾸면 될 듯
  - **SHA1, SHA512, Speck32/64**

- **+ 2022 암호연구회 결과 on (NPU and GPU)**
  - KTNF에서 서버 제공 받으면 돌려 볼 예정

-(Multi-precision 구현) vs. (채널, 고차원 텐서 활용) <- 해당 부분에 대한 의견을 여쭤보고 싶습니다.

멀티 프리시즌 구현은 한번 시도해 볼만합니다. **→ 완료 (NPU 명령어 테스트 했던 것으로 바꾸면 됨)**

채널, 고차원 텐서는 가능하다면 해보시구요. **→ 블록 수가 증가하면 텐서 차원을 추가하여 병렬 처리가 가능할지 생각 중..**

-블록암호 설계를 통해서 NPU를 암호 구현에 활용할 수 있는 최대치(..?)를 분석하고 실제 블록암호 가속에 AI 가속기가 활용되기에는 갭이 있다고 하는 것을 최종 목표로 두면 될까요?

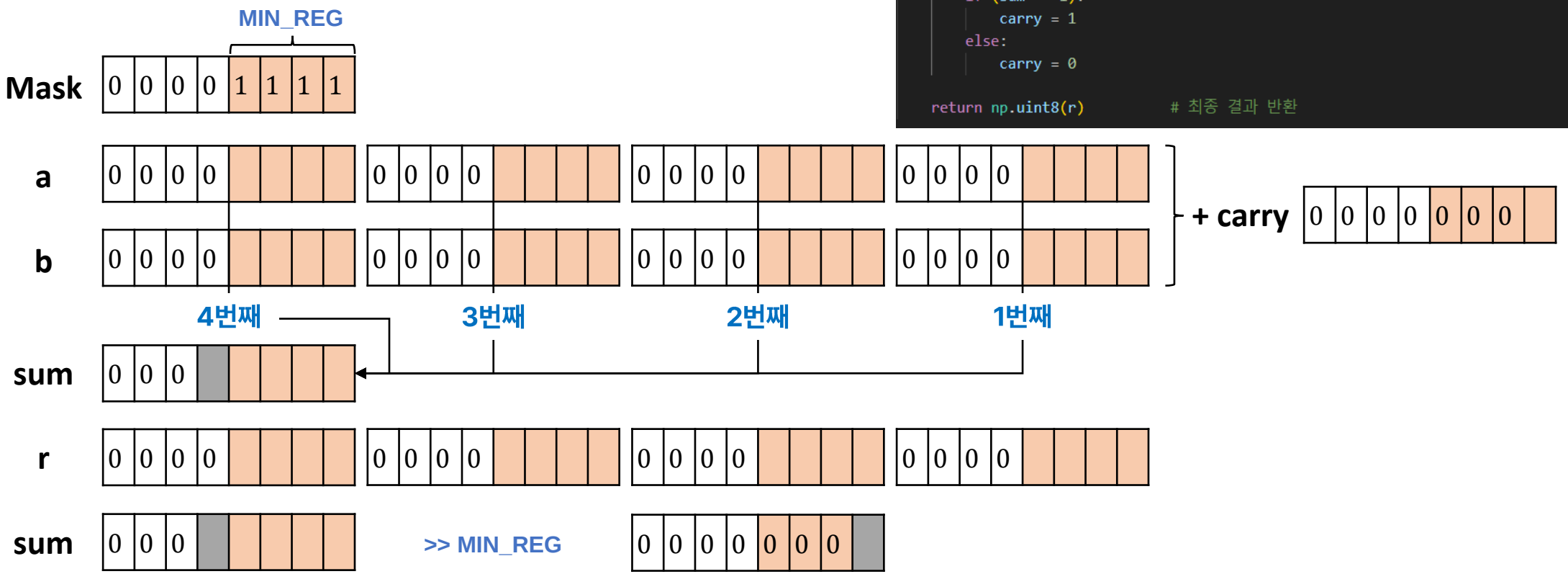
 # 넵 그렇게 하시면 됩니다. 그리고 SIMON/SPECK 같은 경우에는 16비트 연산이 가능하니 이러한 알고리즘에는 어느정도 사용이 가능할수도 있습니다. **→ 블록 암호에 적용은 가능하고, 일부 연산은 가속 불가할 것**

그리고 원래는 블록암호에 NPU를 맞추어야 하지만 반대로 NPU에서 구현이 고속으로 가능한 블록암호를 가상으로 설계해봐도 좋을듯 합니다. 대충 이러이러한 구조로 될 경우 가속이 가능하다가 결론이 될듯 합니다.

Multi-precision: Addition

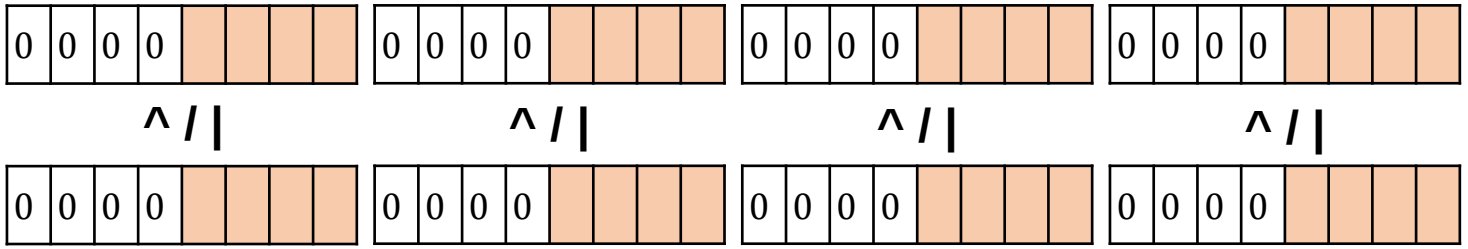
- **NPU에서 사용 가능한 자료형 ($2n$-bit)에 따라 조합 결정**
  - $2n$-bit / n-bit (e.g.,8-bit / 4-bit)
- **아래와 같이 8-bit 중 4-bit만 사용하며, 배열 형태로 구성**
  - 16-bit의 데이터를 표현하기 위해 8-bit 변수 4개 사용
  - 연산 대상 길이에 따라 배열 길이 '**LENGTH**'가 달라짐

```python
def ADD(a, b): # a : [4-bit, 4-bit, 4-bit, 4-bit]   b: [4-bit, 4-bit, 4-bit, 4-bit]

    MASK = 0x0F
    sum = np.uint8(0)
    r = [[0x0] * LENGTH for j in range(len(a))]

    carry = np.uint8(0)

    for i in range(LENGTH):

        # Step 1: uint4 a + uint4 b -> uint8로 변환하여 저장, 어차피 4비트 데이터라 9비트로 overflow 되진 않음
        sum = (a[LENGTH-1-i] + b[LENGTH-1-i]) + carry

        # Step 2: sum을 uint4로 변환하여 r에는 캐리 미포함 4비트 값 저장
        r[LENGTH-1-i] = sum & MASK

        # Step 3: sum을 4비트 shift해서 캐리만 남김
        sum >>= MIN_REG

        if (sum == 1):
            carry = 1
        else:
            carry = 0

    return np.uint8(r)          # 최종 결과 반환
```

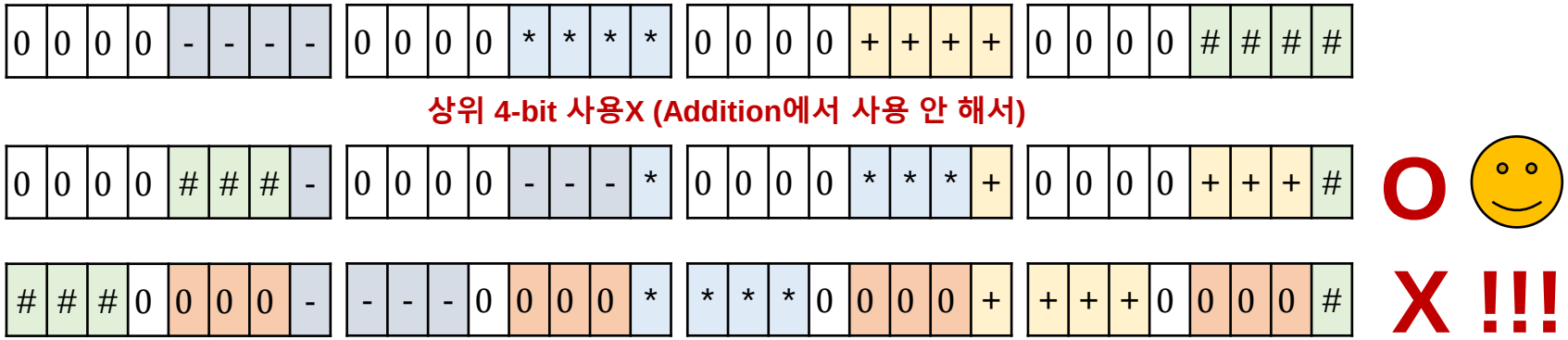


MP addition 이외의 연산 구현

- **Addition 연산에서 사용하는 형태로 연산하기 위해** 다른 연산들 또한 아래와 같이 구성
  - **16-bitwise XOR / OR**



  - **16-bitwise Rotation (Left, Right)**
    - ex: >>>3



O

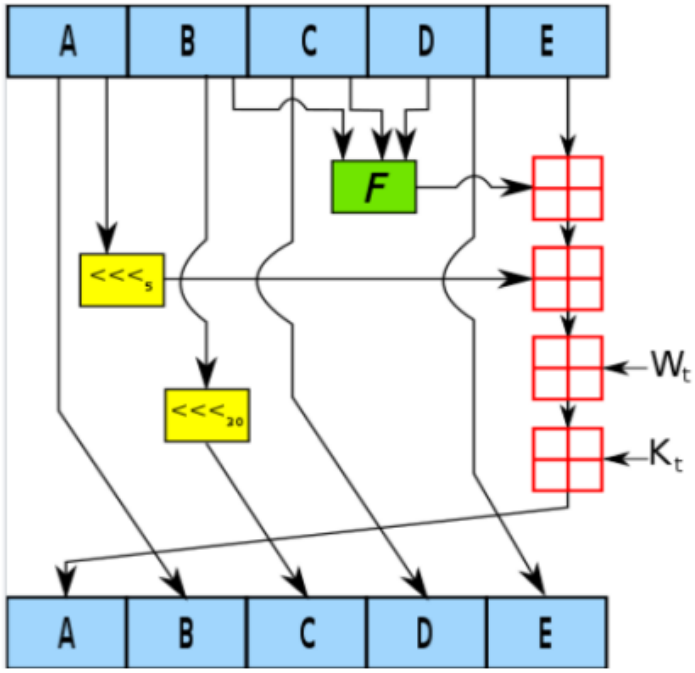
X !!!

SHA-1

- **구조**
  - **32-bit 워드** 단위를 사용하며, **160-bit의 해시**를 출력
  - MD5와 비슷하게 **라운드 함수, modulo addition 및 left rotation으로 구성**
    - **라운드 함수 (F)** : B, C, D 블록을 입력 받음
    - W_i 또는 K_i는 이전 값과 **Modulo addition**
    - A와 B 블록은 각각 **5-bit 및 30-bit Left rotation**

<사용되는 논리 연산자>

⋘	Left rotation
∧	or
⊞	2^{32} Modulo addition



SHA-1

- 구현
    - 코드로 대체

SHA512 암호 논리의 핵심 연산 정리

<사용되는 논리 연산자>

$\ggg$	Right rotation
$\oplus$	XOR
$\wedge$	or
$\boxplus$	2^{64}Modulo addition

- **전처리 단계와 해시 연산 단계로 구성**되며, **512-bit의 해시를 생성**
  - **전처리 단계** : 패딩 및 파싱
  - **해시 연산 단계** : 메시지 확장 및 라운드 함수

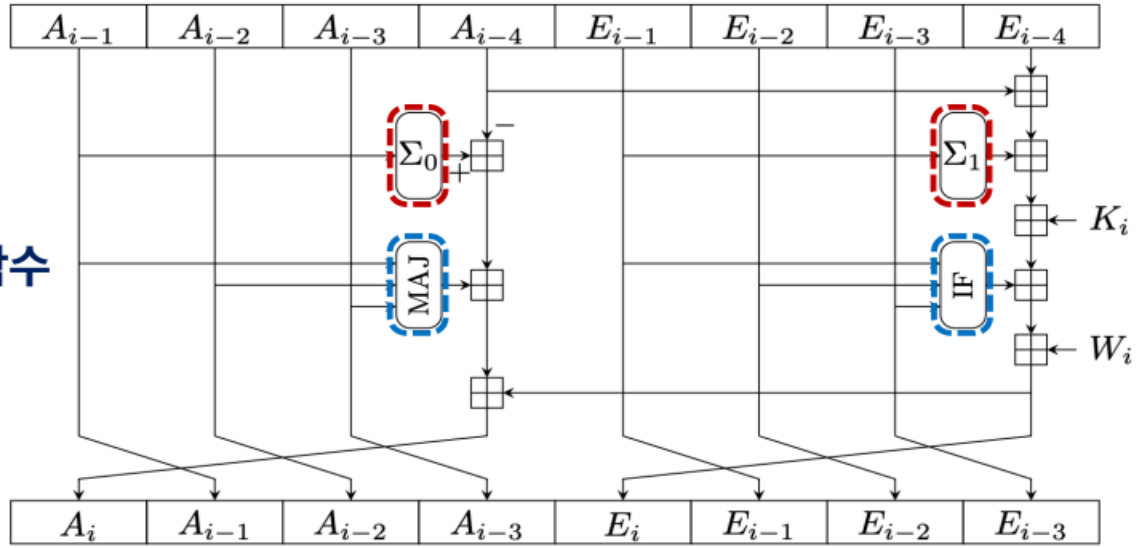
- **1024-bit의 블록을 16개의 64-bit의 워드로 나눈 후 (M_i $(i = 0, \ldots, 15)$), 80개의 워드로 확장**

메시지 확장

$$W_i = \begin{cases} M_i & 0 \le i < 16, \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & 16 \le i < 80. \end{cases}$$

$$\sigma_0(x) = (x \ggg 1) \oplus (x \ggg 8) \oplus (x \gg 7),$$
$$\sigma_1(x) = (x \ggg 19) \oplus (x \ggg 61) \oplus (x \gg 6).$$

라운드 함수



Linear function

$$\Sigma_0(x) = (x \ggg 28) \oplus (x \ggg 34) \oplus (x \ggg 39),$$
$$\Sigma_1(x) = (x \ggg 14) \oplus (x \ggg 18) \oplus (x \ggg 41).$$

Bitwise Boolean function

$$\mathrm{IF}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus z,$$
$$\mathrm{MAJ}(x, y, z) = (x \wedge y) \oplus (y \wedge z) \oplus (x \wedge z),$$

Next hash block

$$h_{j+1} = (A_{79} + A_{-1}, \ldots, A_{76} + A_{-4}, E_{79} + E_{-1}, \ldots, E_{76} + E_{-4}).$$

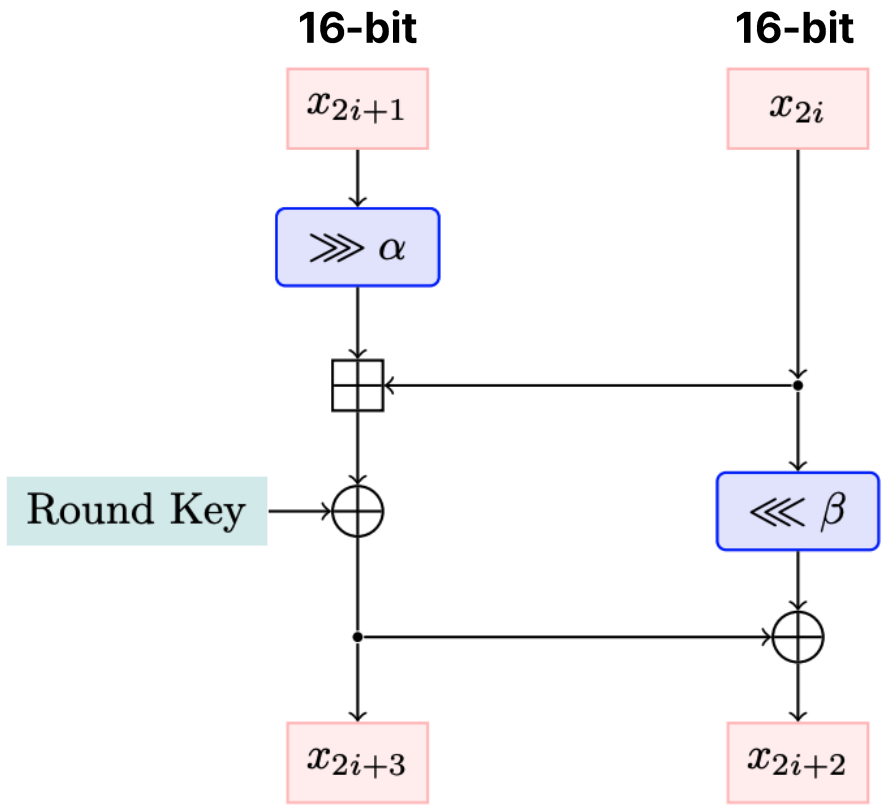
SHA-512

- 구현
    - 코드로 대체

Speck32/64

- **구조**
  - Speck 32/64
    - **32-bit 평문/암호문**
    - **16-bit 단위로 나누어 연산**

  - **사용되는 논리 연산자**
    - Addition
    - Rotation (left, right) → rotation 구현을 위한 OR 연산 필요
    - XOR



Speck32/64

- 구현
  - 코드로 대체

NPU 구현을 위한 고려 사항

- **ATOM 명령어 가속 여부**
  - **Addition**은 가속 가능한 연산자에 속함
  - 그러나, **비트단위의 논리 연산자는 지원하지만 가속 불가**
    - **블록 암호의 필수 연산자가 가속되지 않는 한계점**
  - **16-bit, 8-bit 자료형 사용 가능**

PyTorch OPs	Supported	Accelerated
torch.abs	O	O
torch.acos	O	-
torch.add	O	O
torch.bitwise_and	O	-
torch.bitwise_not	O	-
torch.bitwise_xor	O	-

ATOM

- **WARBOY 명령어 가속 여부**
  - **ATOM에 비해 가속 가능한 연산자가 많음**
    - **ADD, Bitwise AND/NOT/OR ...**
  - 그러나 NPU에서는 **8-bit 자료형만 지원 및 가속 가능**

- **Warboy에서 실행 가능한 ONNX에서 지원하는 가속 연산자 중 암호 논리 연산에 사용될 수 있는 연산자**
  - **ONNX 형식으로 구현해야 함** (뒤의 모든 구현들 또한 ONNX 형식 사용)

ADD, logical AND, Bitwise AND, Bitwise NOT, Bitwise OR, Bitwise XOR, Bit shift, MOD, MUL, Matrix MUL

- **구현은 FP, INT, BF 가능하지만, NPU 실행 시 INT8 타입으로 양자화 후 사용해야 함**

CPU	NPU
Floating Point	INT8
INT	
BFloat	

지원하는 자료형

WARBOY

(암호연구회 2차 발표 자료..)

NPU 구현을 위한 고려 사항

- **9월**
  - **8월에 Multi-precision 구현 한 것을 NPU 코드로 변경**
  - **블록 개수에 따라 병렬 처리 가능한지 확인**
  - **하다 보면 또 다른 요소들이 있을 것으로 생각됨...**

(10월)

1) NPU 상에서의 암호 구현 결과에 대해 각 칩에 대한 비교분석 (Warboy, ATOM)

- NPU를 활용할 수 있는 블록 암호 제시
- 암호 연산에 필요한 데이터 타입, 메모리, 연산자, SW 지원 등의 필요 사항 정리
- 속도, 계산 효율성, NPU 적용의 이점 etc.

2) AI 가속기로 구현 가능한 블록 암호를 설계함으로써 NPU를 통한 암호 가속에는 한계점이 있음을 제시

- 1)에서 제시한 구현과 분석 결과를 기반으로 한계점 (블록 암호 가속과 NPU의 활용의 갭) 분석

3) NPU를 활용함에 있어서 암호가 전체적으로 가속되려면 어떤 요소들이 더 필요한지 분석

- 예를 들어, 'NPU가 INT32를 지원해야한다', 'NPU의 핵심 자료형인 FP 타입의 활용 방안' 등과 같이 어떤 식으로 바꾸면 가능할 지 등에 대한 분석

감사합니다.