ISSN : 1226-7244 (Print)
ISSN : 2288-243X (Online)
논문번호 20-03-30

j.inst.Korean.electr.electron.eng.Vol.24,No.3,901~908,September 2020
http://dx.doi.org/10.7471/ikeee.2020.24.3.901
225

모듈러 역원 연산의 확장 가능형 하드웨어 구현

A Scalable Hardware Implementation of Modular Inverse

최 준 백*, 신 경 욱*★

Jun-Baek Choi*, Kyung-Wook Shin*★

Abstract

This paper describes a method for scalable hardware implementation of modular inversion. The proposed scalable architecture has a one-dimensional array of processing elements (PEs) that perform arithmetic operations in 32-bit word, and its performance and hardware size can be adjusted depending on the number of PEs used. The hardware operation of the scalable processor for modular inversion was verified by implementing it on Spartan-6 FPGA device. As a result of logic synthesis with a 180-nm CMOS standard cells, the operating frequency was estimated to be in the range of 167 to 131 MHz and the gate counts were in the range of 60,000 to 91,000 gate equivalents when the number of PEs was in the range of 1 to 10. When calculating 256-bit modular inverse, the average performance was 18.7 to 118.2 Mbps, depending on the number of PEs in the range of 1 to 10. Since our scalable architecture for computing modular inversion in GF(p) has the trade-off relationship between performance and hardware complexity depending on the number of PEs used, it can be used to efficiently implement modular inversion processor optimized for performance and hardware complexity required by applications.

요 약

몽고메리 모듈러 역원 연산을 확장 가능형 하드웨어로 구현하기 위한 방법에 대해 기술한다. 제안되는 확장 가능형 구조는 워드 (32-비트) 단위로 연산을 수행하는 처리요소의 1차원 배열 구조를 가지며, 사용되는 처리요소의 개수에 따라 성능과 하드웨어 크기를 조절할 수 있다. 설계된 확장 가능형 몽고메리 모듈러 역원기를 Spartan-6 FPGA 소자에 구현하여 하드웨어 동작을 검증하였다. 설계된 역원기를 180-nm CMOS 표준 셀로 합성한 결과, 사용되는 처리요소의 개수 1~10에 따라 동작 주파수는 167~131 MHz, 게이트 수는 60,000~91,000 GEs (gate equivalents)로 평가되었다. 256 비트 모듈러 역원 연산의 경우, 처리요소의 개수 1~10에 따라 평균 18.7~118.2 Mbps의 연산성능을 갖는 것으로 예측되었다. 제안된 확장 가능형 모듈러 역원 연산기는 사용되는 처리요소의 개수에 따라 연산성능과 게이트 수 사이에 교환조건이 성립하며, 따라서 응용분야에서 요구되는 연산성능과 하드웨어 요구량에 최적화된 모듈러 역원 연산회로를 구현할 수 있다.

Key words : Modular inverse, Montgomery inverse, ECC, Public-key cryptography, Scalable architecture

* School of Electronic Engineering, Kumoh National Institute of Technology
★ Corresponding author (E-mail : kwshin@kumoh.ac.kr, Tel : +82-54-478-7427)
※ Acknowledgment
- This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. NRF-2020R1I1A3A04038083)
- This research was supported by the KIAT (Korea Institute for Advancement of Technology) grant funded by the Korea Government (MOTIE: Ministry of Trade Industry and Energy). (No. N0001883, HRD Program for Intelligent semiconductor Industry)
- Authors are thankful to IDEC for supporting EDA software.

Manuscript received Sep. 1, 2020; revised Sep. 25, 2020; accepted Sep. 28, 2020.

Ⅰ. 서론

유한체(finite field) GF(p) 상에서 정수 a에 대한 곱의 역원 $a^{-1}(\bmod p)$을 계산하는 모듈러 역원(modular inverse)은 모듈러 곱셈과 함께 타원곡선 암호(elliptic curve cryptography : ECC), RSA 등의 공개키 암호에서 필수적으로 사용되는 연산이다[1-3]. 예를 들어, Diffie-Hellman(DH) 및 EC-DH (elliptic curve DH) 키 교환, EC-DSA(EC digital signature algorithm) 전자서명 및 검증 등의 공개키 암호 프로토콜들은 모듈러 곱셈, 모듈러 나눗셈, 모듈러 역원 연산을 포함한다. 모듈러 나눗셈은 모듈러 역원과 모듈러 곱셈 연산으로 구현되기도 한다. 공개키 암호에서 모듈러 역원은 모듈러 곱셈에 비해 사용되는 횟수는 적으나, 연산 복잡도가 크고 시간이 많이 소요되는 특성이 있다. 타원곡선 암호의 경우, 적용되는 좌표계(affine, projective, mixed)에 따라 요구되는 모듈러 역원 및 나눗셈의 연산량이 달라질 수 있으나 이를 완전히 제거하는 것은 불가능하며, 따라서 모듈러 역원을 효율적으로 계산하기 위한 연산 알고리듬과 하드웨어 구조에 대한 연구가 활발하게 진행되어 왔다.

모듈러 역원을 구하는 방법으로는 대표적으로 페르마 소정리(Fermat's little theorem)를 이용하는 방법과 최대 공약수(GCD)를 이용하는 방법이 있다. 페르마 소정리는 정수 a와 소수 p에 대하여 식 (1)과 같이 표현된다. 식 (1)에서 양변을 a^2으로 나누면 식 (2)가 되므로, 모듈러 곱셈의 반복 연산으로 모듈러 역원을 계산할 수 있다.

$$a^p \equiv a\,(\bmod p) \tag{1}$$

$$a^{p-2} \equiv a^{-1}\,(\bmod p) \tag{2}$$

페르마 소정리를 이용하는 방법은 소수 p가 큰 수인 경우에는 많은 연산량이 요구되므로, 대부분의 모듈러 역원 연산은 최대 공약수를 이용하는 알고리듬을 사용한다. 최대 공약수를 이용한 방법은 유클리드 호제법(Euclidean algorithm)을 이용하여 큰 수의 최대 공약수를 구해 역원을 구한다. 대표적인 알고리듬으로는 binary right shift 알고리듬[2, 3], binary left shift 알고리듬[2, 4], 유클리디안 알고리듬[2, 3], 몽고메리 역원 알고리듬(Montgomery Modular Inverse Algorithm : MMIA)[5, 6], 스테인 알고리듬[7] 등이 있으며, 알고리듬에 따라 연산 과정과 연산량에 차이가 있다. MMIA는 타 역원 연산 알고리듬에 비해 연산량이 적고, 연산 결과가 몽고메리 도메인 상에 존재하는 특징이 있다. MMIA를 몽고메리 모듈러 곱셈 알고리듬과 함께 ECC의 점 연산 구현에 적용하면, 몽고메리 도메인에서 연산을 처리할 수 있어 효율적인 구현이 가능하므로, 본 논문에서는 모듈러 역원 연산을 위해 MMIA를 선택하였다.

본 논문에서는 MMIA를 가변 성능의 하드웨어로 구현하기 위한 확장 가능형(scalable) 구조를 제안하고, 하드웨어로 구현하였다. 워드 단위(32-비트)로 연산을 수행하는 처리요소 (processing element : PE)의 1차원 배열을 기반으로 하며, 사용되는 PE의 개수에 따라 워드 단위 연산의 병렬성을 조정하여 연산성능과 하드웨어 복잡도를 고려한 최적의 모듈러 역원 연산기를 구현할 수 있는 방법을 제안한다. Ⅱ장에서는 MMIA에 대해 소개하고, Ⅲ장에서는 모듈러 역원 연산기의 확장 가능형 하드웨어 구조와 회로설계에 대해 설명한다. Ⅳ장에서는 설계된 확장 가능형 역원기의 기능 검증 및 FPGA 검증, 그리고 사용되는 PE 개수에 따른 연산성능과 하드웨어 복잡도 평가를 기술하고, Ⅴ장에서 결론을 맺는다.

Ⅱ. 몽고메리 역원 알고리듬

1. 몽고메리 모듈러 역원 알고리듬[5, 6]

MMIA는 Kaliski [5]에 의해 제안되었으며, Savas와 Koc에 의해 정형화되었다. MMIA는 연산결과가 몽고메리 도메인 상에 존재하는 특징이 있다. 그림 1은 Savas와 Koc에 의해 정형화된 MMIA[6]의 연산과정을 나타낸 슈도코드이며 Phase Ⅰ과 Phase Ⅱ 두 과정으로 구성된다. Phase Ⅰ은 Almost Montgomery Inverse(AlmMonInv)과정이며, 연산결과 R이 유사 몽고메리 도메인 상의 값으로 출력된다. Phase Ⅰ은 L 비트의 정수 A와 모듈러 값 N을 입력받아, L 비트의 역원 연산결과 $R = A^{-1} \times 2^k(\bmod N)$ (단, $L \le k \le 2L$)와 반복루프 횟수 k를 출력한다. 이진 나눗셈의 반복 연산에 의해 역원 연산 결과에 $2^k(\bmod N)$가 곱해져 출력되며, 입력되는 정수 A의 값에 영향을 받는다. 반복루프 횟수 k는 평균적으로 약 $1.4L$의 값을 가지며, Phase Ⅰ의 소요 클록 사이클 수는 반복루프 횟수 k와 비례관계를 갖는다. Phase Ⅰ의 연산

```
Phase Ⅰ
Input  ; A = {a_{L-1}, ···, a_1, a_0}_2
         N  (1 ≤ A < N)
Output; R = A^{-1} × 2^k (mod N), k (L ≤ k ≤ 2L)
  U ← N, V ← A, R ← 0, S ← 1, k ← 0
  while (V > 0)
     if (u_0 = 0) then
        U ← U/2, S ← 2 × S
     else if (v_0 = 0) then
        V ← V/2, R ← 2 × R
     else if (U > V) then
        U ← (U − V)/2, R ← R + S, S ← 2 × S
     else
        V ← (V − U)/2, S ← R + S, R ← 2 × R
     end if
     k ← k + 1
  end while
  if (R ≥ N) then
     R ← R − N
  end if
  return (N − R), k
```

```
Phase Ⅱ
Input  ; R = {r_{L-1}, ···, r_1, r_0}_2 = A^{-1} × 2^k (mod N)
         N, L, k  (1 ≤ R < N)
Output; R = A^{-1} × 2^L (mod N)
  for i = 1 to (k − L) do
     if (r_0 = 0) then
        R ← R/2
     else
        R ← (R + N)/2
     end if
  end for
  return R
```

Fig. 1. Pseudocode for MMIA [6].
그림 1. MMIA의 슈도코드 [6]

은 이진 곱셈, 이진 나눗셈, 모듈러 가산 및 감산을 통해 진행되며, 이진 곱셈과 이진 나눗셈은 비트 시프트 동작으로 처리된다.

PhaseⅡ는 Correction Phase(CorPh) 과정으로, PhaseⅠ의 연산결과를 몽고메리 도메인으로 변환시킨다. PhaseⅠ의 결과 $R=A^{-1}\times 2^k(\mathrm{mod}N)$, 반복루프 횟수 k와 모듈러 값 N, 비트 수 L을 입력받아 $R=A^{-1}\times 2^L(\mathrm{mod}N)$을 출력하며, 연산 결과가몽고메리 도메인 상에 존재한다.

몽고메리 역원 연산의 소요 사이클 수를 줄이기 위해, 한 번에 여러 비트를 시프트시켜 반복루프 횟수를 줄이는 방법이 제안되었다[8～10]. 한 번에 다수의 하위 비트를 스캔하여 시프트 동작을 연속으로 수행하는 경우에 한 사이클에 여러 비트를 시프트하고, k 값에 비트 수만큼 가산하여 소요 사이클 수를 줄일 수 있다. 3 비트씩 스캔하는 경우, 실험적으로 소요 사이클 수를 약 22% 줄일 수 있으며, 평균 반복루프는 $1.1L$회가 된다.

2. 확장 가능형 몽고메리 모듈러 역원 알고리듬

그림 2는 본 논문에서 제안하는 확장 가능형 몽고메리 모듈러 역원 알고리듬의 연산과정을 나타낸 슈도코드이다. L 비트의 정수 데이터 A, 모듈러 값 N을 입력받아, L 비트의 역원 연산결과 $R=A^{-1}\times 2^k(\mathrm{mod}N)$ (단, $L\le k\le 2L$)와 반복루프 횟수 k 값을 출력한다. 입력된 데이터 A, N은 w 비트 크기의 워드 m개로 분할되어 PE 배열에 의해 워드 단위 병렬로 연산된다. 본 논문에서는 워드 크기를 w=32로 적용하였다.

그림 2의 슈도코드에서 확장 가능형 몽고메리 역원 알고리듬의 모든 연산은 반복루프 내에서 이루어지며, 매 반복루프마다 내부의 U와 V의 현재 데이터에 의해 연산 동작모드가 결정된다. 연산 동작모드에 따라 가산 및 감산 연산과 시프트 연산을 위한 데이터가 선택되어 연산이 수행되며, 하위 3 비트를 스캔하여 연속 시프트 동작으로 소요 사이클 수를 줄였다. 그림 2의 슈도코드에서, $RS(x,y)$는 x 데이터 워드를 y 비트만큼 오른쪽으로 시프트시키는 연산을 의미하며, $LS(x,y)$는 x 데이터 워드를 y 비트만큼 왼쪽으로 시프트시키는 연산을 의미한다. 그림 2의 슈도코드에서 j-루프는 N_{PE}개의 PE에 의해 N_{PE}개의 워드가 병렬로 연산되는 동작을 의미하고, i-루프는 PE 배열을 통한 반복 연산을 의미한다. i-루프의 반복 연산 횟수는 워드 개수 m과 사용되는 PE 개수 N_{PE}에 의해 $ite=\lceil m/N_{PE}\rceil$로 결정된다. 그림 2의 슈도코드에 의한 연산과정은 다음과 같다.

(1) 단계-1 : 연산에서 사용되는 데이터를 초기화 시키는 과정이며, L 비트의 정수 데이터 A, 모듈러 값 N을 입력받는다.

(2) 단계-2～단계-57 : 역원 연산을 진행하는 반복루프로, U와 V의 현재 값에 따라 반복루프의 동작모드가 달라지며, V=0이 될 때까지 반복 수행된다. 반복루프는 조건문에 따라 단계-4, 단계-19, 단계-34, 단계-45 중 하나의 연산이 수행되며, 단계-4와 단계-19는 하위 3 비트를 스캔하여 수행되는 연속 시프트 동작이다.

(3) 단계-4～단계-18 : U의 현재 데이터가 짝수인 경우의 연산과정이며, 단계-5～단계-11은 하위 3 비트를 스캔하여 연속 시프트 동작을 위한 k 데이터 가산 및 SN 값을 설정해 준다. 단계-1

2~단계-18은 워드 단위의 시프트 연산으로 구성된다. i-루프는 $ite = \lceil m/N_{PE} \rceil$ 회 반복 수행되며, j-루프는 N_{PE}개 워드를 병렬 연산한다. U와 S 데이터의 시프트 연산이 수행되며, LS의 경우 캐리 데이터가 다음 루프에서 사용되고, RS의 경우 단계-15와 같이 캐리 데이터 C_{RS}를 이전 워드의 알맞은 위치에 저장한다.

(4) 단계-19~단계-33 : U의 현재 데이터가 홀수이고 V의 현재 데이터가 짝수인 경우의 연산과정이며, 단계-4~단계-18과 유사한 연산과정이다. 단계-27~단계-33은 단계-12~단계-18과 동일한 연산 구조를 가지며, 데이터 V, R의 워드 단위 시프트 연산이 수행된다.

(5) 단계-34~단계-44 : U와 V의 현재 데이터가 홀수이고 $U>V$인 경우의 연산과정이며, 가산 연산이 포함되므로, 여러 비트를 스캔하지 않는다. 따라서 SN 값은 1로 고정되며, k 값 또한 1만큼 증가된다. 두 개의 가산기를 이용하여 가산과 감산 연산이 동시에 수행된다. 가산기에서 단계-38, 단계-39의 감산 U-V와 가산 S+R이 수행되며, 감산 연산의 결과와 S 데이터의 시프트 연산도 수행된다.

(6) 단계-45~단계-56 : U와 V의 현재 데이터가 홀수이고, $U \le V$인 경우의 연산과정이며, 단계-34~단계-44와 유사한 연산이다. 단계-49, 단계-50의 감산 V-U와 가산 S+R이 수행되며, 단계-36~단계-44와 동일한 연산구조를 갖는다.

(7) 단계-58 : 축약과정이 포함된 연산 출력과정이며, N-R 연산과 2N-R 연산이 수행되고, $N-R>0$이면 N-R을 출력하고, $N-R<0$이면 2N-R을 출력한다.

위의 연산과정을 통해 반복루프 횟수를 나타내는 k 값과 역원 연산결과 $R=A^{-1}\times 2^{k}(\mathrm{mod}N)$가 출력된다. 반복루프 1회에 소요되는 사이클 수는 $ite = \lceil m/N_{PE} \rceil$ 로 표현되며, 반복루프는 약 $1.1L$회 반복되어 역원 연산에 소요되는 평균 사이클 수는 $1.1L \times ite$로 나타낼 수 있다.

III. 확장 가능형 모듈러 역원 연산기 코어

그림 2의 슈도코드를 하드웨어로 구현하여 확장 가능형 몽고메리 모듈러 역원 연산기 코어를 설계

```
Input  ; A = {a_(L-1), ..., a_1, a_0}_2 = {a^(m-1), ..., a^1, a^0}_(2^w)
         N  (1 ≤ A < N)
Output; R = A^(-1) × 2^k (mod N), k  (L ≤ k ≤ 2L)
Pre_computed ; ite = ⌈m/N_PE⌉, m = ⌈L/w⌉
Number of PE ; N_PE
Word size ; w

1 :  U ← N, V ← A, R ← 0, S ← 1, k ← 0
2 :  while (V > 0)
3 :     C_RS ← 0, C_LS ← 0
4 :     if (u_0 = 0) then
5 :        if (u_2 u_1 u_0 = 000) then
6 :           SN ← 3, k ← k + 3
7 :        else if (u_2 u_1 u_0 = 100) then
8 :           SN ← 2, k ← k + 2
9 :        else if (u_2 u_1 u_0 = X10) then
10:           SN ← 1, k ← k + 1
11:        end if
12:        for i = 0 to (ite − 1) do
13:           concurrent j = 0 to (N_PE − 1) do
14:              {u^(j+i*N_PE), C_RS} ← {0, u^(j+i*N_PE)};  RS(u^(j+i*N_PE), SN)
15:              u^((j+i*N_PE)−1)_[31:32−SN] ← C_RS
16:              {C_LS, s^(j+i*N_PE)} ← {s^(j+i*N_PE), C_LS};  LS(s^(j+i*N_PE), SN)
17:           end concurrent
18:        end for
19:     else if (v_0 = 0) then
20:        if (v_2 v_1 v_0 = 000) then
21:           SN ← 3, k ← k + 3
22:        else if (v_2 v_1 v_0 = 100) then
23:           SN ← 2, k ← k + 2
24:        else if (v_2 v_1 v_0 = X10) then
25:           SN ← 1, k ← k + 1
26:        end if
27:        for i = 0 to (ite − 1) do
28:           concurren j = 0 to (N_PE − 1) do
29:              {v^(j+i*N_PE), C_RS} ← {0, v^(j+i*N_PE)};  RS(v^(j+i*N_PE), SN)
30:              v^((j+i*N_PE)−1)_[31:32−SN] ← C_RS
31:              {C_LS, r^(j+i*N_PE)} ← {r^(j+i*N_PE), C_LS};  LS(r^(j+i*N_PE), SN)
32:           end concurrent
33:        end for
34:     else if (U > V) then
35:        k ← k + 1, C_A1 ← 1, C_A2 ← 0, SN ← 1
36:        for i = 0 to (ite − 1) do
37:           concurrent j = 0 to (N_PE − 1) do
38:              {C_A1, u^(j+i*N_PE)} ← u^(j+i*N_PE) + bit_inv(v^(j+i*N_PE)) + C_A1
39:              {C_A2, r^(j+i*N_PE)} ← s^(j+i*N_PE) + r^(j+i*N_PE) + C_A2
40:              {u^(j+i*N_PE), C_RS} ← {0, u^(j+i*N_PE)};  RS(u^(j+i*N_PE), SN)
41:              u^((j+i*N_PE)−1)_[31:32−SN] ← C_RS
42:              {C_LS, s^(j+i*N_PE)} ← {s^(j+i*N_PE), C_LS};  LS(s^(j+i*N_PE), SN)
43:           end concurrent
44:        end for
45:     else
46:        k ← k + 1, C_A1 ← 1, C_A2 ← 0, SN ← 1
47:        for i = 0 to (ite − 1) do
48:           concurrent j = 0 to (N_PE − 1) do
49:              {C_A1, v^(j+i*N_PE)} ← v^(j+i*N_PE) + bit_inv(u^(j+i*N_PE)) + C_A1
50:              {C_A2, s^(j+i*N_PE)} ← s^(j+i*N_PE) + r^(j+i*N_PE) + C_A2
51:              {v^(j+i*N_PE), C_RS} ← {0, v^(j+i*N_PE)};  RS(v^(j+i*N_PE), SN)
52:              v^((j+i*N_PE)−1)_[31:32−SN] ← C_RS
53:              {C_LS, r^(j+i*N_PE)} ← {r^(j+i*N_PE), C_LS};  LS(r^(j+i*N_PE), SN)
54:           end concurrent
55:        end for
56:     end if
57: end while
58: S1 ← Sub(N, R), S2 ← Sub(2N, R)
59: if (S1 ≥ 0) then
60:    return S1, k
61: else
62:    return S2, k
63: end if
```

Fig. 2. Pseudocode for scalable MMIA proposed in this paper.
그림 2. 본 논문에서 제안하는 확장 가능형 MMIA의 슈도코드

하였다. 설계된 역원 연산기 코어 SModInv는 SEC2 [11]에 정의된 소수체 상의 5가지 키 길이(192, 224, 256, 384, 521 비트)를 지원하며, 그림 3과 같이 SInvCal 블록, CNTL 블록, 레지스터 파일 Reg_File 블록, 계수기 K-counter 블록 등으로 구성된다. SInvCal 블록은 모듈러 역원을 연산하는 회로이며, PE의 1차원 배열과 다수개의 MUX로 구성되어 사용된 PE의 개수 N_{PE}에 따라 연산속도와 하드웨어 복잡도가 달라진다. SModInv 코어는 CNTL 블록 내부의 OP 레지스터 설정에 따라 동작이 제어된다. Reg_File 블록은 544 비트 크기의 U_reg, V_reg, R_reg, S_reg, N_reg 레지스터와 캐리 데이터를 저장하는 레지스터들로 구성되며, U, V, R, S 데이터 및 캐리 데이터를 저장한다. K-counter 블록은 내부에 가산기를 포함하며, 반복루프 마다 Int_CNTL 블록에서 생성되는 SN 값을 가산하여 k 값을 결정하며, 역원 연산이 완료되면 최종 k 값을 출력한다. Int_CNTL 블록은 U와 V의 현재 값에 따라 SN 값과 동작모드를 결정한다. SN 값은 연속 시프트 되는 비트 수를 나타내며, 본 논문에서는 3 비트 스캔을 적용하므로 1~3 범위의 값을 갖는다. 그림 2의 슈도코드에서 단계-4, 단계-19, 단계-34, 단계-45의 조건과 축약 및 출력 과정의 결과에 의해 동작모드가 결정되며, 이에 따라 PE에 입력되는 데이터와 PE의 연산 동작이 결정된다.

동작모드 신호와 선택기 MUX_I1~MUX_I4에 의해 Reg_File의 데이터 중 연산에 필요한 데이터가 선택되며, CNTL 블록의 신호에 따라 각 사이클에

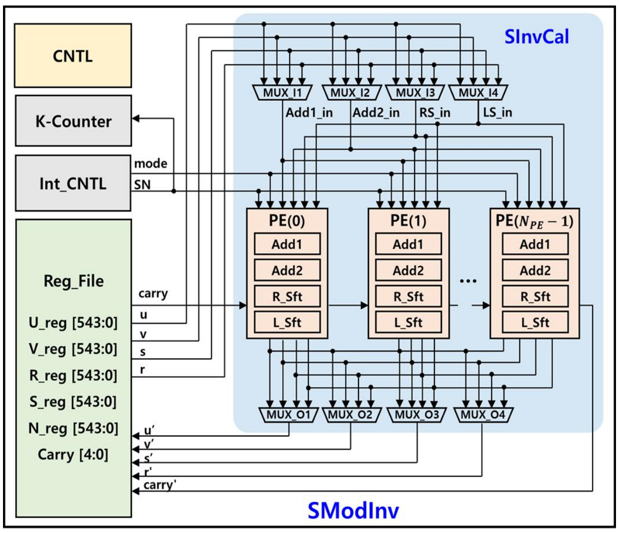


Fig. 3. Scalable core for computing modular inverse.
그림 3. 확장 가능형 모듈러 역원 연산기 코어

연산되는 N_{PE}개의 워드가 결정된다. 결정된 N_{PE}개의 워드는 PE 배열의 가산기 및 시프트기에 입력되어 동시에 연산되며, 슈도코드 상의 j-루프를 의미한다. 이때, PE의 가산기 및 시프트기의 캐리 데이터는 인접한 PE로 전달되어 사용되며, 최상위 PE의 캐리 데이터는 저장되어 다음 루프에서 사용된다. PE 배열에 의해 연산이 완료된 데이터는 MUX_O1~MUX_O4에 의해 레지스터에 저장된다. 사용되는 PE의 개수 N_{PE}에 의해 병렬 연산되는 워드 수와 반복루프 1회에 소요되는 사이클 수가 결정되므로, 역원 연산기의 응용분야에서 요구되는 성능에 맞춰 사용되는 PE의 개수를 조정하여 구현할 수 있다.

그림 4는 확장 가능형 몽고메리 역원기의 PE 내부 블록도이며, 32 비트 가산기 두 개(Add1, Add2), 오른쪽 시프트기(R_sft), 왼쪽 시프트기(L_sft)와 선택기 두 개(MUX1, MUX2)로 구성된다. R_sft와 L_sft는 데이터를 입력받아 SN 값만큼 시프트 연산을 수행한다. PE의 Add1은 U-V, V-U의 감산과 N-R의 연산을 수행하며, Add2는 R+S의 가산과 2N-R의 감산 연산을 수행한다. Add1이 단계-34, 단계-45의 조건에 의해 단계-38, 단계-49의 감산을 수행할 경우, MUX1에 의해 Add1의 감산 연산의 결과가 R_sft로 입력되어 단계-40 및 단계-51의 시프트 연산이 수행된다. 단계-4, 단계-19의 조건에 의해 시프트 연산만 수행되는 경우, MUX1에 의해 외부 입력 데이터가 선택되어 시프트 연산이 수행된다.

그림 2의 슈도코드에서 단계-58은 축약 과정을 포함한 데이터 출력 과정이며, N-R의 연산과 2N-R의 연산이 PE 배열을 통해 동시에 수행된다. Add1을 통해 N-R의 연산이 수행되며, L_sft를 통해 N을

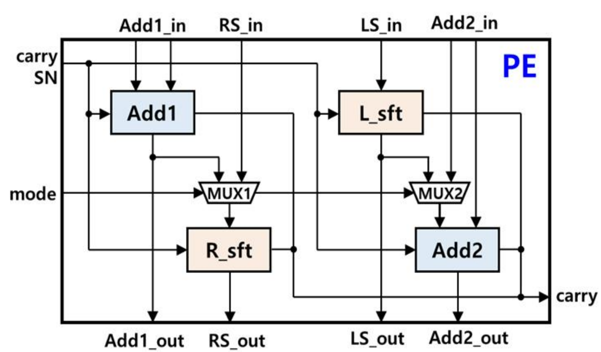


Fig. 4. Block diagram of processing element.
그림 4. 처리요소의 내부 블록도

2N으로 연산하고, Add2를 통해 2N-R 연산이 수행된다. 선택기 MUX2는 축약 연산을 위해 L_sft의 결과를 Add2의 입력으로 선택하고, 반복루프를 실행할 때에는 외부의 입력을 Add2의 입력으로 선택한다.

PE의 시프트기 2개는 3 비트의 캐리 데이터를 가지며, 가산기 2개는 1 비트의 캐리 데이터를 갖는다. 각 PE의 캐리 데이터는 인접한 PE로 전달된다. PE의 1차원 배열로 인한 최악경로(critical path) 지연을 줄이고자 가산기를 캐리선택 가산기(carry select adder)로 구현하였다. R_sft의 캐리 데이터는 연산과정 중 알맞은 위치에 즉시 저장된다. 마지막 j-루프에서 생성된 L_sft의 캐리 데이터는 저장되어 다음 루프의 연산에서 사용된다.

Ⅳ. 검증 및 성능 평가

1. 기능검증 및 FPGA 검증

SModInv 코어는 Verilog HDL 기반으로 설계되었으며, SEC2 표준에 정의된 소수체 상의 8가지 타원곡선(P192R, P224R, P256R, P384R, P521R, P192K, P224K, P256K) 파라미터에 대해 RTL 기능검증을 수행하였다. N_{PE}=2, L=192 비트인 경우의 RTL 기능검증 결과는 그림 5와 같다. 모듈러 값 N은 P192K 곡선의 “ffffffff ffffffff ffffffff ffffffff fffffffe ffffee37”를 사용했으며, 정수 A는 “06c76a69 efa03369 e98af190 6a813fdf 217314c8 b117a5a7”을 사용했다. SModInv 코어에서 연산된 정수 A의 모듈러 역원 R은 “7bc2b4f0 61d5b8c9 b8d3f602 94cad467 5a7c0461 5ad569d6”이고, k 값은 281이 출력되었다. RTL 시뮬레이션 결과가 Python 소프트웨어로 계산된 결과와 일치했으며, 이를 통해 설계된 역원 연산기 코어가 올바로 동작함을 확인하였다.

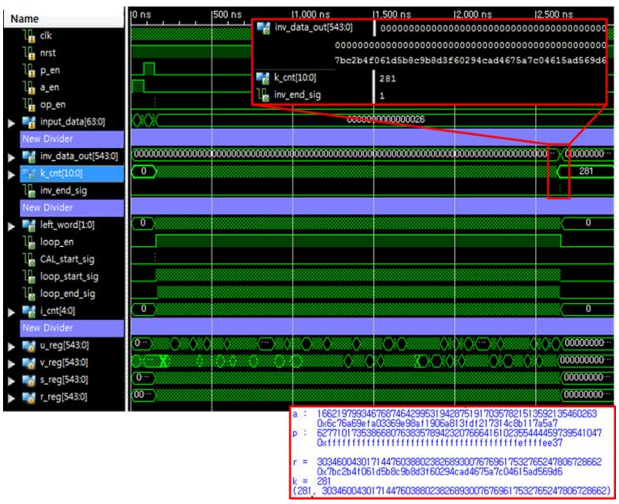
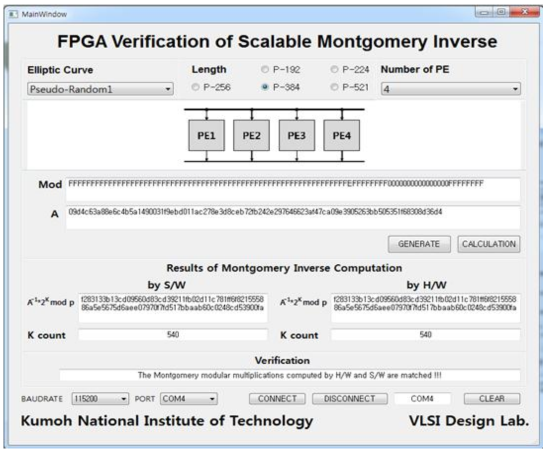
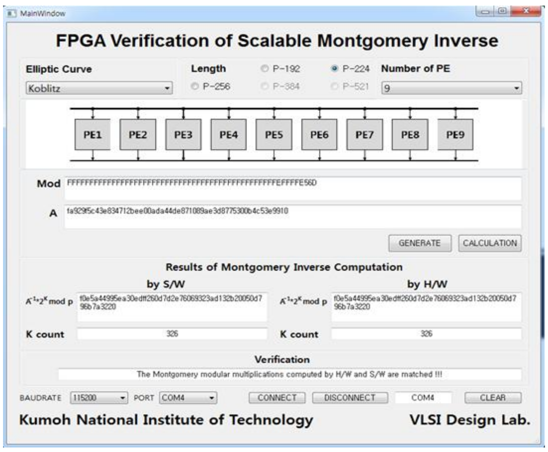


Fig. 5. Simulation results of SModInv core.
그림 5. SModInv 코어의 시뮬레이션 결과

설계된 SModInv 코어를 Spartan-6 FPGA 디바이스에 구현하여 하드웨어 동작을 검증하였다. Uart 인터페이스, Wrapper, GUI로 FPGA 검증 플랫폼을 구성하였다. 그림 6은 설계된 SModInv 코어의 FPGA 검증 결과이다. 그림 6-(a)는 N_{PE}=4, L=384 비트인 경우, 6-(b)는 N_{PE}=9, L=224 비트인 경우의 동작을 보인 화면 캡처이다. 모듈러 값 N과 정수 A를 FPGA로 전송하고, SModInv 코어에서 연산된 결과를 GUI 화면에 출력하였다. 소프트웨어로 계산된 역원 결과와 FPGA에서 계산된 결과가 일치함을 확인하였다. SEC2에 정의된 소수체 상의 8가지 타원곡선 파라미터에 대해 PE가 1개 사용된



(a) N_{PE}=4, L=384-bit



(b) N_{PE}=9, L=224-bit

Fig. 6. FPGA verification results of SModInv core.
그림 6. SModInv 코어의 FPGA 검증 결과

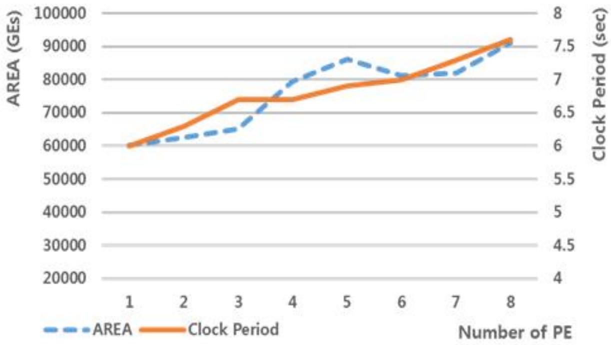


Fig. 7. Gate counts and critical path delays of SModInv core according to the number of PEs used.
그림 7. 사용된 PE 개수에 따른 SModInv 코어의 게이트 수와 최악경로 지연

경우에서부터 PE가 10개 사용되는 경우까지 FPGA 검증을 실행한 결과, 모든 경우에 대해 정상 동작함을 확인하였다.

2. 성능평가

설계된 SModInv 코어를 180-nm CMOS 표준 셀 라이브러리로 합성한 결과는 그림 7과 같으며, SModInv 코어에 사용된 PE 개수 N_{PE}에 따른 회로 복잡도 (등가 게이트 수)와 최악경로 지연을 그래프로 나타내었다. $N_{PE}=1$인 경우와 $N_{PE}=8$인 경우의 최악경로 지연은 각각 6 nsec, 7.6 nsec로 평가되었고, 등가 게이트(gate equivalent) 수는 각각 60,218 GEs와 90,960 GEs로 분석되었다. SModInv 코어에 PE가 많이 사용될수록 당연히 회로 복잡도가 증가하며, $N_{PE}=8$인 경우의 SModInv 코어는 $N_{PE}=1$인 경우에 비해 약 1.5배 많은 게이트를 필요로 한다. 또한, $N_{PE}=1$인 경우에 비해 $N_{PE}=8$인 경우의 지연이 약 30% 증가하는 것으로 분석되었으며, 이는 N_{PE}가 클수록 PE 간의 캐리 체인이 길어져 최악경로 지연도 증가하는 것에 기인한다.

표 1은 사용된 PE 개수 N_{PE}에 따른 연산성능과 면적 대비 연산성능을 나타낸 것이다. SModInv 코어는 L=256의 경우에 284회의 반복루프가 수행되며, 반복루프 1회에 $\lceil 8/N_{PE} \rceil$ 사이클이 소요된다. L=256의 경우, SModInv 코어의 평균 연산성능은 18.7~118.2 Mbps로 평가되었다. 표 1을 통해, N_{PE}가 커짐에 따라 등가 게이트 수와 클록 주기가 증가하는 추세를 가지며, 전체 연산에 소요되는 클록 사이클 수는 감소하여 SModInv 코어의 회로 복잡도와 연산성능 사이에 교환조건이 존재함을 확인할 수 있다. 따라서 본 논문의 확장 가능형 모듈러 역원 연산기 코어 SModInv는 응용분야의 요구 조건에 따라 저면적을 필요로 하는 경우에는 PE 개수 $N_{PE}=1\sim2$로 적게 사용하여 구현하고, 고성능이 요구되는 경우에는 PE 개수를 $N_{PE}=6\sim8$로 사용하여 구현할 수 있다.

Table 1. Performance of SModInv core according to the number of PEs used N_{PE} (L=256-bit)
표 1. 사용된 PE 개수 N_{PE}에 따른 SModInv 코어의 성능 (L=256-bit)

N_{PE}	Clock Period	Area	Cycle for 1 loop (P256)	Throughput AVG (P256)	Throughput /Area
1	6 nsec	60,218 GEs	8 cycles	18.7 Mbps	0.31 kbps/GE
2	6.3 nsec	62,541 GEs	4 cycles	35.6 Mbps	0.57 kbps/GE
3	6.7 nsec	65,228 GEs	3 cycles	44.7 Mbps	0.69 kbps/GE
4	6.7 nsec	79,297 GEs	2 cycles	67.0 Mbps	0.85 kbps/GE
5	7 nsec	80,112 GEs	2 cycles	64.1 Mbps	0.80 kbps/GE
6	7 nsec	81,312 GEs	2 cycles	64.1 Mbps	0.79 kbps/GE
7	7.3 nsec	82,039 GEs	2 cycles	61.5 Mbps	0.75 kbps/GE
8	7.6 nsec	90,960 GEs	1 cycle	118.2 Mbps	1.30 kbps/GE

V. 결론

몽고메리 역원 연산을 확장 가능형 하드웨어로 구현하기 위한 방법을 제안하였으며, 이를 적용한 확장 가능형 모듈러 역원 연산기 코어 SModInv를 설계하고, Spartan-6 FPGA 디바이스에 구현하여 하드웨어 동작을 검증하였다. 사용되는 PE 개수 N_{PE}에 따라 연산성능과 면적 사이에 교환조건이 성립하며, 응용분야에서 요구되는 성능조건에 따라 사용되는 PE의 개수를 결정함으로써 면적과 연산성능을 고려한 최적의 모듈러 역원 연산기 구현이 가능하다. 향후, 본 논문에서 설계된 확장 가능형 모듈러 역원 연산기 코어를 확장 가능형 ECC 프로세서 설계에 적용하는 후속 연구가 진행될 예정이다.

References

[1] Y. Kim, “Efficient Algorithm for Multi-Bit Montgomery Inverse Using Refined Multiplicative

Inverse Modular 2^K," *IEEE Access*, vol.7, pp. 906-918, 2018. DOI: 10.1109/ACCESS.2018.2885989

[2] L. Hars, "Modular Inverse Algorithms Without Multiplications for Cryptographic Applications," *EURASIP Journal on Embedded Systems 2006*, pp.1-13, 2006. DOI: 10.1155/ES/2006/32192

[3] D. E. Knuth, "The Art of Computer Programming, Volume 2: Seminumerical Algorithms," Addison-Wesley, Reading, Mass, USA, 3rd edition, 1997.

[4] R. Lórencz, "New algorithm for classical modular inverse," *in Cryptographic Hardware and Embedded Systems, ser. LNCS*, vol.2523. London, UK: Springer-Verlag, pp.57-70, 2002. DOI: 10.1007/3-540-36400-5_6

[5] B. S. Kaliski, "The Montgomery inverse and its applications," *IEEE Transactions on Computers*, vol.44, no.8, pp.1064-1065, 1995. DOI: 10.1109/12.403725

[6] E. Savas and C. K. Koc, "The Montgomery modular inverse-revisited," *IEEE Transactions on Computers*, vol.49, no.7, pp.763-766, 2000. DOI: 10.1109/12.863048

[7] H. Zhang, R. Li, L. Li and Y. Dong, "Improved speed Digital Signature Algorithm based on modular inverse," *Proceedings of 2013 2nd International Conference on Measurement, Information and Control*, Harbin, pp.706-710, 2013. DOI: 10.1109/MIC.2013.6758059

[8] A. A. A. Gutub and A. F. Tenca, "Efficient scalable VLSI architecture for montgomery inversion in GF(p)," *Integration*, vol.37, no.2, pp.103-120, 2004. DOI: 10.1016/j.vlsi.2003.12.001

[9] A. A. A. Gutub, A. F. Tenca and C. K. Koc, "Scalable VLSI architecture for GF(p) Montgomery modular inverse computation," *Proceedings IEEE Computer Society Annual Symposium on VLSI*, Pittsburgh, PA, USA, pp.53-58, 2002. DOI: 10.1109/ISVLSI.2002.1016874

[10] P. J. Choi and D. K. Kim, "Efficient Hardware Montgomery Modular Inverse Module for Elliptic Curve Cryptosystem in GF (p)," *Journal of Korea Multimedia Society*, vol.20, no.2 pp.289-297, 2017. DOI: 10.9717/kmms.2017.20.2.289

[11] Certicom, Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0, 2000.

BIOGRAPHY

Jun-Baek Choi (Member)

2019 : BS degree in Electronic Engineering, medical IT convergence engineering, Kumoh National Institute of Technology.
2019~ : Graduate student, Kumoh National Institute of Technology

Kyung-Wook Shin (Member)

1984 : BS degree in Electronic Engineering, Korea Aerospace University
1986 : MS degree in Electronic Engineering, Yonsei University
1990 : Ph.D. degree in Electronic Engineering, Yonsei University
1990~1991 : Senior Researcher, Semiconductor Research Center, Electronics and Telecommunications Research Institute (ETRI)
1991~Present : Professor in School of Electronic Engineering, Kumoh National Institute of Technology
1995~1996 : University of Illinois at Urbana- Champaign (Visiting Professor)
2003~2004 : University of California at San Diego (Visiting Professor)
2013~2014 : Georgia Institute of Technology (Visiting Professor)