ISSN : 1226-7244 (Print)
ISSN : 2288-243X (Online)
j.inst.Korean.electr.electron.eng.Vol.25,No.4,625~633,December 2021
https://doi.org/10.7471/ikeee.2021.25.4.625
논문번호 21-04-07
42

확장 가능형 몽고메리 모듈러 곱셈기

A Scalable Montgomery Modular Multiplier

최 준 백*, 신 경 욱**★

Jun-Baek Choi*, Kyung-Wook Shin**★

Abstract

This paper describes a scalable architecture for flexible hardware implementation of Montgomery modular multiplication. Our scalable modular multiplier architecture, which is based on a one-dimensional array of processing elements (PEs), performs word parallel operation and allows us to adjust computational performance and hardware complexity depending on the number of PEs used, N_{PE}. Based on the proposed architecture, we designed a scalable Montgomery modular multiplier (sMM) core supporting eight field sizes defined in SEC2. Synthesized with 180-nm CMOS cell library, our sMM core was implemented with 38,317 gate equivalents (GEs) and 139,390 GEs for $N_{PE}=1$ and $N_{PE}=8$, respectively. When operating with a 100 MHz clock, it was evaluated that 256-bit modular multiplications of 0.57 million times/sec for $N_{PE}=1$ and 3.5 million times/sec for $N_{PE}=8$ can be computed. Our sMM core has the advantage of enabling an optimized implementation by determining the number of PEs to be used in consideration of computational performance and hardware resources required in application fields, and it can be used as an IP (intellectual property) in scalable hardware design of elliptic curve cryptography (ECC).

요 약

몽고메리 모듈러 곱셈의 유연한 하드웨어 구현을 위한 확장 가능형 아키텍처를 기술한다. 처리요소 (processing element; PE)의 1차원 배열을 기반으로 하는 확장 가능형 모듈러 곱셈기 구조는 워드 병렬 연산을 수행하며, 사용되는 PE 개수 N_{PE}에 따라 연산 성능과 하드웨어 복잡도를 조정하여 구현할 수 있다. 제안된 아키텍처를 기반으로 SEC2에 정의된 8가지 필드 크기를 지원하는 확장 가능형 몽고메리 모듈러 곱셈기(scalable Montgomery modular multiplier; sMM) 코어를 설계했다. 180-nm CMOS 셀 라이브러리로 합성한 결과, sMM 코어는 $N_{PE}=1$ 및 $N_{PE}=8$인 경우에 각각 38,317 등가게이트 (GEs) 및 139,390 GEs로 구현되었으며, 100 MHz 클록으로 동작할 때, $N_{PE}=1$인 경우에 57만회/초 및 $N_{PE}=8$인 경우에 350만회/초의 256-비트 모듈러 곱셈을 연산할 수 있는 것으로 평가되었다. sMM 코어는 응용분야에서 요구되는 연산성능과 하드웨어 리소스를 고려하여 사용할 PE 수를 결정함으로써 최적화된 구현이 가능하다는 장점을 가지며, ECC의 확장 가능한 하드웨어 설계에 IP (intellectual property)로 사용될 수 있다.

Key words: Modular multiplication, Montgomery multiplication, ECC, scalable architecture, scalable multiplier

* Ranix Inc.
** 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)
- Authors are thankful to IDEC for supporting EDA software.

Manuscript received Nov. 25, 2021; revised Dec. 10, 2021; accepted Dec. 15, 2021.

Ⅰ. 서론

대표적인 공개키 암호 방식인 타원곡선 암호(elliptic curve cryptography; ECC) [1]는 블록체인(block chain)의 트랜잭션 검증, 자율주행 자동차의 차량간 통신 보안, 드론의 식별인증 등 다양한 분야에서 폭넓게 사용되고 있다[2, 3]. 예를 들어, V2V (Vehicle to Vehicle) 및 V2I (Vehicle to Infrastructure) 무선통신을 지원하는 지능형 교통 시스템(intelligent transport system; ITS) 표준 WAVE(Wireless Access in Vehicular Environment)에서는 ECDSA (Elliptic Curve Digital Signature Algorithm), ECIES(Elliptic Curve Integrated Encryption Scheme) 등 ECC 기반의 전자서명과 공개키 암호 방식을 포함하고 있다[4].

ECC는 응용분야 및 구현되는 보안 기능에 따라 타원곡선이 정의되는 체(field)와 크기, 타원곡선의 종류, 요구되는 연산성능 등이 달라진다. 따라서 ECC의 하드웨어 구현 시 다양한 분야의 요구조건을 충족시킬 수 있도록 유연성(flexibility)과 확장성(scalability)을 고려하는 것이 필요하다. 이를 위해서는 ECC의 점 연산을 구현하는 하위 연산인 유한체 상의 모듈러 곱셈, 모듈러 나눗셈 등의 확장 가능형(scalable) 설계가 필수적이다. 여기서 확장성은 응용분야나 구현되는 보안 기능에 따라 연산성능과 하드웨어 크기를 가변시킬 수 있는 구조를 의미한다.

다양한 체 크기의 ECC 구현을 위한 확장 가능형(scalable) 모듈러 곱셈기에 관한 연구가 발표되고 있으며, 이를 구현하는 방법은 크게 두 가지로 나눌 수 있다. 고정된 크기(예를 들면, 워드 기반)의 연산회로를 이용해서 여러 가지 크기의 모듈러 곱셈을 구현하는 방법 [5]과 워드단위 연산회로(처리요소)의 배열 구조를 통해 연산성능과 하드웨어 요구량을 가변시킬 수 있는 방법 [6-8]이다. 예를 들어, 문헌 [6]은 2차원 시스토릭 배열(systolic array) 구조를 이용한 확장 가능형 몽고메리 모듈러 곱셈기를 제안하였으나, 5종류의 처리요소가 사용되고, 연산에 소요되는 클록 수가 곱셈기 비트 수에 무관하다는 특징을 갖는다.

본 논문에서는 응용분야에서 요구되는 연산성능과 하드웨어 복잡도 사이의 최적화를 고려한 확장 가능형 모듈러 곱셈기 구조를 제안한다. 워드기반 몽고메리 곱셈 알고리듬을 기반으로 처리요소 (processing element; PE)의 1차원 배열 구조를 이용하여 32-비트 워드를 병렬로 연산하는 방법과 곱셈기 구조를 제안한다. 제안된 워드기반 몽고메리 곱셈 알고리듬을 적용하여 확장 가능형 몽고메리 모듈러 곱셈기 코어를 설계했다. SEC2 [9]에 정의된 소수체 상의 8가지 타원곡선을 지원하며, 사용되는 PE 개수 N_{PE}에 따라 연산성능과 면적을 최적화시킬 수 있는 장점을 갖는다. Ⅱ장에서는 워드기반 몽고메리 곱셈 알고리듬을 소개하고, Ⅲ장에서는 확장 가능형 몽고메리 모듈러 곱셈기 설계에 대해 설명한다. Ⅳ장에서는 FPGA 검증과 성능평가 결과를 기술하고, Ⅴ장에서 결론을 맺는다.

Ⅱ. 몽고메리 곱셈 알고리듬

1. 몽고메리 모듈러 곱셈 알고리듬

몽고메리 모듈러 곱셈 알고리듬[10]은 모듈러 합동 특성을 이용하는 유한체 곱셈 알고리듬으로, L 비트의 승수 A와 피승수 B의 곱셈결과를 모듈러스 N으로 축약하여 L 비트의 모듈러 곱셈결과 $S = A \times B \times R^{-1} \bmod N$ (단, $R^{-1} = 2^{-L}$)을 출력한다. 몽고메리 모듈러 곱셈 알고리듬은 그림 1의 슈도코드로 나타낼 수 있다. 단계-2에서 승수 A의 각 비트 a_i (단, $0 \le i \le L-1$)와 피승수 B를 곱하여 부분곱 $a_i \times B$를 생성하고, 이전 루프의 부분곱 가산 결과 S에 더하여 $L+1$ 비트의 결과를 얻는다. 단계-3~단계-4는 축약과정이며, 모듈러 합동 특성을 이용해 부분곱 가산 결과의 최하위 비트 s_0를 0으로 만들고, 단계-4의 시프트 과정을 통해 최하위 비트를 제거하여 L 비트의 연산결과를 만든다. i-루프

```
Input  ;  A = {a_{L-1}, ..., a_1, a_0}_2
          B, N  (0 ≤ A, B < N)
Output;   S = {s_{L-1}, ..., s_1, s_0}_2
          = A × B × R^{-1} mod N

1: for i = 0 to (L − 1) do
2:   S ← S + (a_i × B)
3:   S ← S + (s_0 × N)
4:   S ← S ≫ 1
5: end for
6: if (S ≥ N) then
7:   S ← S − N
8: end if
9: return S
```

Fig. 1. Montgomery multiplication algorithm [10].
그림 1. 몽고메리 곱셈 알고리듬 [10]

가 L회 반복되므로, 모듈러 곱셈결과에 2^{-L}가 포함된다. L 비트 승수에 대한 부분곱 생성 및 가산이 완료되면, 연산결과 S와 모듈러스 N을 비교하여 $S \geq N$인 경우, $S-N$의 모듈러 뺄셈 연산을 거쳐 최종 모듈러 곱셈결과가 얻어진다.

2. 확장 가능형 워드기반 몽고메리 곱셈 알고리듬

본 논문에서 제안하는 확장 가능형 워드기반 몽고메리 곱셈 알고리듬(scalable word-based Montgomery multiplication algorithm; sWMMA)의 슈도코드는 그림 2와 같으며, 워드기반 몽고메리 곱셈 알고리듬의 변형이다[11, 12]. 승수와 피승수 A, B 그리고 모듈러스 N은 w-비트의 워드 m개로 분할되어 연산된다. 워드의 크기 w는 16, 32, 64 비트로 설계에 따라 결정된다. n_0^*는 모듈러스 N의 최하위 워드 n_0로부터 $n_0^* = -n_0^{-1} \bmod 2^w$로 정의되며, 모듈러 합동 특성을 이용하여 부분곱 가산 결과의 최하위 워드 s_{-1}을 0으로 만들어 제거하는 축약 연산에 사용된다.

```
Input ; A = {a_{m-1}, ..., a_1, a_0}_{2^w} (0 ≤ A < N)
        B = {b_{m-1}, ..., b_1, b_0}_{2^w} (0 ≤ B < N)
        N = {n_{m-1}, ..., n_1, n_0}_{2^w}
Output; S = {s_{m-1}, ..., s_1, s_0}_{2^w} = A×B×R^{-1} mod N
Pre_computed ; n_0^* = -n_0^{-1} mod 2^w, r = ⌈m/N_PE⌉, m = ⌈L/w⌉
Number of PE used ; N_PE ,  Word size ; w

1:  for h = 0 to (r − 1) do
2:    concurrent k = 0 to (N_PE − 1) do
3:      PL_q_(h×N_PE+k) ← a_(h×N_PE+k)×b_0 mod 2^w
4:    end concurrent
5:  end for
6:  for i = 0 to (m − 1) do
7:    C_add1 = C_add2 ← 0
8:    c1 = c2 = c3 = c4 ← 0
9:    q ← (s_0 + PL_q_i)×n_0^* mod 2^w
10:   for j = 0 to (r − 1) do
11:     PH1_{-1} ← C_add1, PH2_{-1} ← C_add2
12:     concurrent k = 0 to (N_PE − 1) do
13:       {PH1_k, PL1_k} ← a_i×b_{j×N_PE+k}
14:       {c1, s_k^*} ← PL1_k + PH1_{k-1} + c1
15:       {c2, s_k} ← s_k^* + s_{j×N_PE+k} + c2
16:       {PH2_k, PL2_k} ← q×n_{j×N_PE+k}
17:       {c3, s_{j×N_PE+k-1}} ← PL2_k + PH2_{k-1} + c3
18:       {c4, s_{j×N_PE+k-1}} ← s_k^* + s_{j×N_PE+k-1} + c4
19:     end concurrent
20:     if (j = r − 1) then
21:       C_add1 ← PH1_{m-j×N_PE}, C_add2 ← PH2_{m-j×N_PE}
22:     else
23:       C_add1 ← PH1_{N_PE-1}, C_add2 ← PH2_{N_PE-1}
24:     end if
25:   end for
26:   {c_msb, s_{m-1}} ← c1 + c2 + c3 + c4 + c_msb + C_add1 + C_add2
27: end for
28: if (S ≥ N) then
29:   S ← S − N
30: end if
31: return S
```

Fig. 2. Scalable word-based Montgomery multiplication algorithm (sWMMA).

그림 2. 확장 가능형 워드기반 몽고메리 곱셈 알고리듬 (sWMMA)

그림 2의 슈도코드에서 i-루프와 j-루프는 각각 승수 A와 피승수 B를 워드 단위로 처리하며, i-루프는 승수의 워드 개수인 m회 반복되고 j-루프의 반복 횟수는 $r = \lceil m/N_{PE} \rceil$ 가 된다. j-루프 내부의 concurrent k-루프는 N_{PE}개의 PE를 이용하여 N_{PE}개의 피승수 워드를 병렬로 처리하며, 사용되는 PE의 개수 N_{PE}에 따라 연산에 소요되는 사이클 수가 달라진다. sWMMA의 연산처리 과정은 다음과 같다.

(1) 단계-1~단계-5 : i-루프의 축약연산에 사용되는 q 값을 생성하기 위해 $a_i \times b_0$ 곱셈을 통해 부분곱 하위 워드 PL_q를 계산한다. PE 배열에 의해 N_{PE}개의 워드가 병렬로 연산되며, r회 반복 연산된다.

(2) 단계-7~단계-9 : i-루프 반복이 시작될 때, 캐리 데이터를 초기화하고 축약 연산을 위한 q 값을 생성한다.

(3) 단계-11 : 이전 j-루프에서 생성된 부분곱의 상위워드 C_add1, C_add2가 현재 j-루프에서 가산에 사용되도록 저장한다.

(4) 단계-12~단계-19 : PE 배열에 의해 N_{PE}개의 피승수 워드가 병렬로 연산되며, 곱셈과 가산 연산으로 구성된다. 단계-13~단계-15는 승수 워드 a_i와 피승수 워드 b_j의 곱셈연산을 통해 부분곱이 생성되며, 생성된 부분곱의 상위워드 $PH1_k$는 $(k+1)$-번째 워드의 부분곱 가산에 사용된다. 생성된 부분곱의 하위워드 $PL1_k$는 $(k-1)$-번째 워드의 부분곱 상위워드 $PH1_{k-1}$와 가산되며, 이전 i-루프의 부분곱 가산결과 워드 s_k와 가산된다. 이때, 1회의 i-루프 연산은 1개의 승수 워드 a_i와 m개의 피승수 워드 $b_0 \sim b_{m-1}$ 간의 곱셈이 계산되며, $m+1$개 워드의 부분곱 가산결과가 생성된다. 단계-16~단계-18은 단계-13~단계-15에 의해 생성된 $m+1$개의 부분곱 가산결과 워드에 대한 축약연산 과정이며, 모듈러 합동 특성을 이용해 $s_{m-2} \sim s_{-1}$ 워드 중 최하위 워드 s_{-1}를 0으로 만들어 제거한다. 단계-13~단계-15와 동일한 연산구조 (곱셈 1회, 가산 2회)를 가지며, 단계-9에서 생성된 q 값과 모듈러스 값이 사용된다.

(5) 단계-20~단계-24 : concurrent k-루프에서 생성된 부분곱 상위워드 $PH1_{N_{PE}-1}$와 $PH2_{N_{PE}-1}$을

저장하여 다음 j-루프의 단계-11에서 사용되도록 한다. 마지막 j-루프일 경우, 마지막 워드가 연산된 PE의 부분곱 상위워드 값을 저장하며, 단계-26에서 최상위 워드 s_{m-1}를 생성하는 가산에 사용된다.

(6) 단계-26 : j-루프의 반복이 완료되면, 단계-20~단계-24에 의해 저장된 부분곱 상위워드와 이전 i-루프의 캐리 값 c_{msb}을 가산하여 최상위 워드 s_{m-1}와 캐리 값 c_{msb}을 생성한다.

(7) 단계-28~단계-30 : i-루프의 반복이 완료되면, 곱셈결과 S와 N을 비교하여 $S \geq N$일 경우 $S-N \bmod N$의 축약연산을 한다.

위의 과정을 통해 몽고메리 모듈러 곱셈 결과 $S=A\times B\times R^{-1} \bmod N$ 가 출력된다. 매 i-루프마다 단계-16~단계-18의 축약 과정을 통해 최하위 워드 s_{-1}이 제거되며, m회의 i-루프를 반복을 통해 m개의 워드가 제거되어 $R^{-1} \bmod N$이 포함된 곱셈 결과가 얻어진다.

그림 3은 N_{PE}개의 PE를 사용하여 그림 2의 sWMMA를 구현하는 경우의 연산과정을 나타낸 모식도이다. x-축은 사용되는 PE 개수 $p=N_{PE}$를 나타내며, 그림 2의 슈도코드에서 concurrent k-루프에 의한 p개 워드가 병렬 연산됨을 의미한다. y-축은 모듈러 곱셈에 소요되는 클록 사이클 수를 나타내며, 그림 2의 슈도코드에서 m개의 승수 워드에 대한 연산을 처리하는 i-루프와 그 내부에 피승수 워드에 대한 연산을 처리하는 j-루프로 구성된다. p개의 PE가 사용되는 경우, m개의 피승수 워드에 대한 연산을 위해 j-루프에서 $r=\lceil m/p \rceil$ 클록 사이클이 소요된다. 처음 r 클록 사이클은 슈도코드의 단계-1~단계-5의 q 값 생성에 소요되며, 나머지 $m\times r$ 클록 사이클은 모듈러 곱셈의 부분곱 워드 생성과 가산 그리고 축약연산에 소요된다. 예를 들어, 4개의 PE를 사용해서(즉, $p=4$) 256-비트 모듈러 곱셈을 연산하는 경우 워드의 수는 $m=256/32=8$이고, j-루프에서 $r=\lceil 8/4 \rceil=2$ 클록 사이클이 소요되며, 총 18 클록 사이클이 소요된다. 이와 같이, sWMMA는 사용되는 PE의 개수에 따라 연산성능과 하드웨어 복잡도를 조정할 수 있다는 장점을 갖는다.

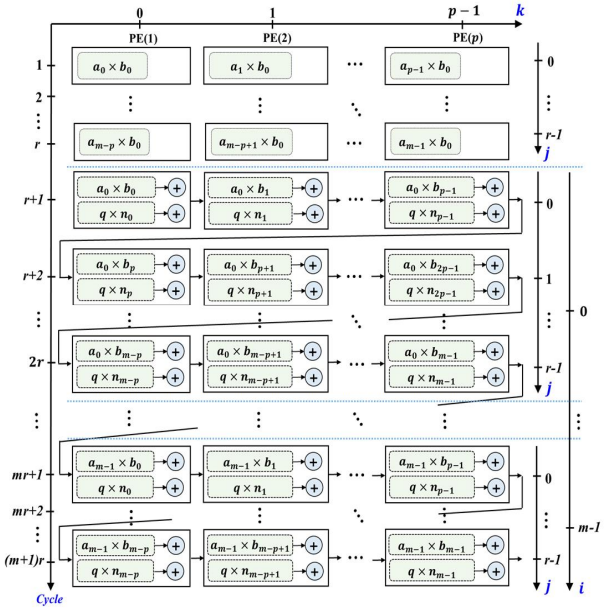


Fig. 3. Space(PEs)-time(# of cycles) diagram of sWMMA ($p=N_{PE}$ and $r=\lceil m/p \rceil$).

그림 3. sWMMA의 공간(PEs)-시간(사이클 수) 모식도 ($p=N_{PE}$ and $r=\lceil m/p \rceil$)

III. 확장 가능형 모듈러 곱셈기 설계

PE의 1차원 배열에 의해 피승수 워드를 병렬로 처리 하는 확장 가능형 몽고메리 모듈러 곱셈기(scalable Montgomery modular multiplier; sMM) 구조를 제안한다. sMM은 SEC2에 정의된 소수체 상의 8가지 타원곡선(P192R, P192K, P224R, P224K, P256R, P256K, P384R, P521R)을 지원하며, PE를 1개~8개까지 사용해 하드웨어를 구현할 수 있다. 그림 4는 sMM의 내부 블록도이며, 데이터를 저장하는 RAM, 파라미터와 연산 중간결과 값을 저장하는 레지스터 블록 Reg_Mod, 산술연산을 수행하는 sALU 블록, 연산과정을 제어하는 Cntl_FSM 블록으로 구성된다. sALU 블록은 내부에 PE의 1차원 배열을 포함하며, 사용되는 PE의 개수 N_{PE}에 따라 연산성능과 하드웨어 복잡도를 조정할 수 있다. RAM은 연산에 필요한 데이터와 연산결과를 저장하며, 연산이 완료되면 연산결과를 외부로 출력한다. 이때, sALU 블록에 사용된 PE의 개수 N_{PE}에 따라 RAM의 입출력 포트 크기가 $w\times N_{PE}$로 설정된다. 본 논문에서는 워드크기를 $w=32$-비트로 설정하여 설계했다. Reg_Mod 블록은 승수 A와 n_0^* 데이터를 저장하여 sALU 블록으로 입력하며, RAM에서 데이터를 읽고, 저장할 수 있다.

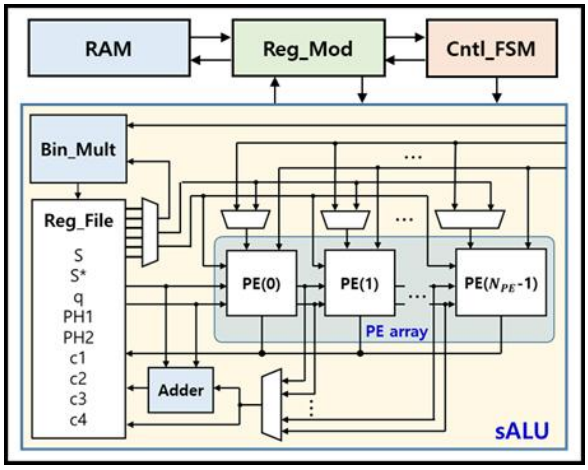


Fig. 4. Architecture of sMM core.
그림 4. sMM 코어의 구조

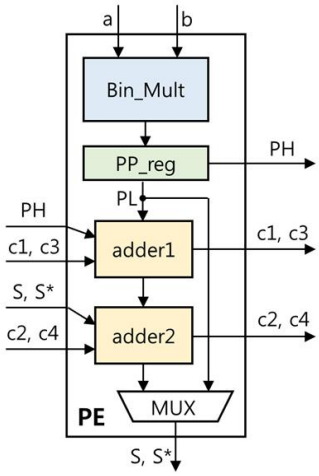


Fig. 5. Block diagram of processing element.
그림 5. 처리요소의 블록도

sALU 블록은 S 데이터, q 값, 캐리 데이터 등의 연산 중간결과 값들을 저장하는 Reg_File, N_{PE}개의 PE 1차원 배열, 32-비트 이진 곱셈기(Bin_Mult), 32-비트 가산기 그리고 MUX 및 DEMUX들로 구성된다. PE 배열을 구성하는 각 PE는 워드단위의 연산을 처리하며, 그림 2 슈도코드의 concurrent k-루프 내부의 연산을 수행한다. 이진 곱셈기 Bin_Mult는 단계-9에서 q를 생성하는 이진 곱셈에 사용되며, MUX에 의해 해당 i-루프의 PL_q_i가 선택된다. 또한, Cntl_FSM 블록에서 출력되는 동작모드 신호와 계수기 신호에 따라 연산에 필요한 워드가 Reg_File에서 출력되며, MUX에 의해 선택되어 PE 배열로 입력된다. PE 배열로 입력되는 데이터의 크기는 사용되는 PE의 개수 N_{PE}에 따라 $32\times N_{PE}$ 비트가 되며, 32-비트의 워드로 분할되어 각 PE에 입력된다. MUX는 외부에서 입력되는 데이터와 Reg_File에서 출력되는 데이터 중 선택하여 PE로 입력하며, PE 배열에서 연산이 완료된 데이터는 Reg_File에 저장된다. 인접한 PE들은 연산에 필요한 캐리 데이터를 주고 받는다. 이때, 마지막 j-루프의 경우에는 가산기에서 단계-26의 최상위 워드 s_{m-1}을 연산하여 Reg_File에 저장한다.

그림 5는 PE의 내부 블록도이며, 32-비트 이진 곱셈기(Bin_Mult)와 32-비트 가산기(adder1, adder2), 선택기(MUX)와 레지스터(PP_reg)로 구성된다. PE는 그림 2 슈도코드의 단계-1~단계-5의 q 데이터를 위한 부분곱 하위워드 PL_q의 생성과, 단계-12~단계-19의 곱셈 및 가산을 수행한다. 그림 2 슈도코드의 concurrent k-루프 내에서 1회의 곱셈과 2회의 가산이 각 PE에 의해 병렬로 2회 반복 연산되며, 이를 위해 PE 내부의 이진 곱셈기와 가산기를 직렬형태로 구현했다. 이진 곱셈기는 그림 2 슈도코드의 단계-13과 단계-16의 곱셈을 수행하며, adder1은 단계-14와 단계-17의 가산을, adder2는 단계-15와 단계-18의 가산을 수행한다. 각 가산기는 1 비트 캐리 값을 가지며, PE 배열에 의한 가산기의 지연을 줄이고자 16-비트 캐리선택 가산기(carry select adder)를 2개씩 사용하여 구현하였다. 이진 곱셈기와 가산기 사이에 파이프라인 레지스터 PP_reg를 삽입하여 최악경로(critical path) 지연이 최소화되도록 하였으며, 이진 곱셈과 가산 연산에 2 사이클이 소요된다. 이때, PL과 PH의 생성에 연산 중간결과 값이 사용되지 않는 점을 이용하여 추가로 소모되는 사이클을 최소화하였다. 곱셈을 수행한 후, 2회의 가산과 다음 연산과정의 곱셈이 동시에 처리되도록 하였다. 그림 2의 슈도코드에서 $i=0$이고, $j=0$인 초기 연산의 경우, 단계-13의 곱셈이 수행되고, $i=m$이고, $j=r-1$인 마지막 연산의 경우, 단계-17과 단계-18의 가산이 수행된다. 이 외의 연산 과정에서는, 단계-14와 단계-15의 가산과 단계-16의 곱셈이 동시에 수행되며, 단계-17과 단계-18의 가산과 단계-13의 곱셈이 동시에 연산되고, 위의 연산이 교대로 반복된다.

Ⅳ. FPGA 구현 및 성능평가

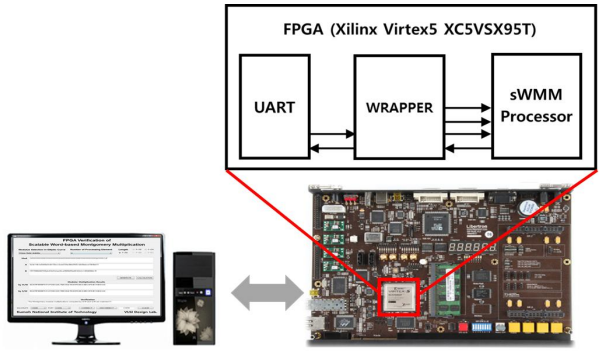
1. FPGA 검증

설계된 sMM 코어를 그림 6-(a)의 FPGA 검증

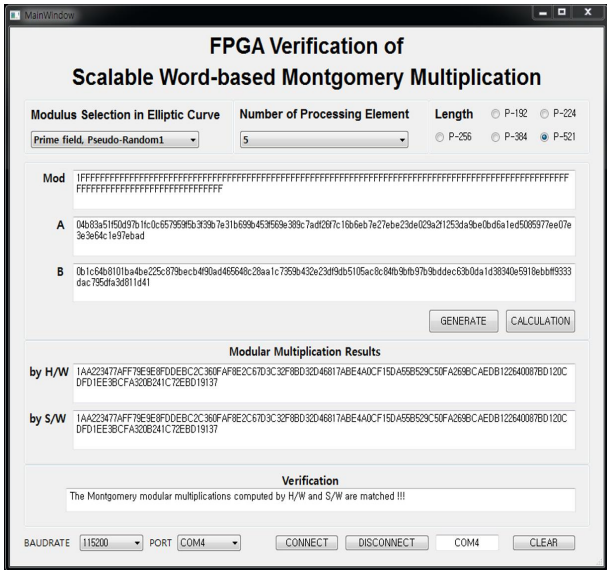
플랫폼에 구현하여 하드웨어 동작을 검증하였으며, GUI를 통해 FPGA에서 연산된 결과와 소프트웨어로 계산된 결과를 비교하여 정상 동작을 확인하였다. 그림 6-(b)는 $N_{PE}=4$로 구현된 sMM 코어에서 256-비트 모듈러 곱셈을 연산한 결과의 화면 캡처이다. PE 개수 N_{PE}에 따른 8가지 하드웨어 구현에 대해 SEC2에 정의된 8가지 소수체 타원곡선 상의 모듈러 곱셈을 검증하여 모두 올바로 동작함을 확인하였다.

2. 성능평가

표 1은 사용되는 PE 개수 N_{PE}에 따른 sMM 코어의 소요 클록 사이클 수, 등가게이트 수, RAM 크기 등을 보인 것이다. sMM 코어의 연산성능은 모듈러 곱셈에 소요되는 클록 사이클 수로 평가할 수 있으며, 사용된 PE의 개수 N_{PE}와 워드 수 m에 따라 $5r+2mr+7$ (단, $r=\lceil m/N_{PE} \rceil$)로 계산된다. N_{PE}가 증가함에 따라 소요 클록 사이클 수는 1/5로 감소하고, 면적은 약 3.6배 증가하는 것으로 분석되었다. 256-비트 모듈러 곱셈의 초당 연산량은 $N_{PE}=1$인 경우 약 57만회, $N_{PE}=4$인 경우에 약 200만회, $N_{PE}=8$인 경우에 약 357만회로 평가되었다. 설계된 sMM 코어를 180-nm CMOS 셀 라이브러리로 합성한 결과, 약 9 nsec의 최대경로 지연을 갖는 것으로 평가되었으며, $N_{PE}=1$인 경우에 38,317 등가게이트 (GEs), $N_{PE}=8$인 경우에 139,320 GEs로 평가되었다.



(a) FPGA verification setup



(b) N_{PE}=5, P521R curve

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

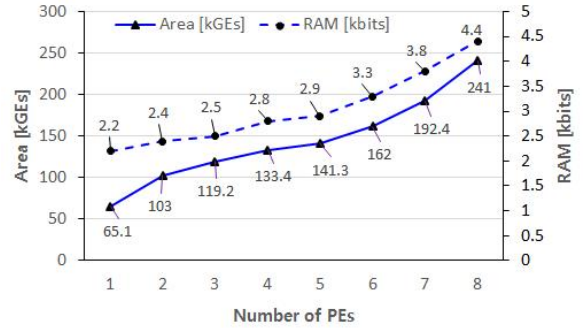
Table. 1. sMM core's performance according to N_{PE}.
표 1. N_{PE}에 따른 sMM 코어의 성능

N_{PE}		1	2	3	4	5	6	7	8
# of clock cycles	192-bit	109	58	41	41	41	24	-	-
	224-bit	140	83	64	45	45	45	26	-
	256-bit	175	91	70	49	49	49	49	28
	384-bit	355	181	123	94	94	65	65	65
	521-bit	670	358	241	202	163	124	124	124
Area (kGEs)*		38.3	65.8	79.4	91.0	113.6	105.0	137.5	139.3
RAM (kbits)		2.2	2.4	2.5	2.8	2.9	3.3	3.8	4.4
256-b Mult. per second ($\times 10^6$)		0.57	1.09	1.42	2.04	2.04	2.04	2.04	3.57

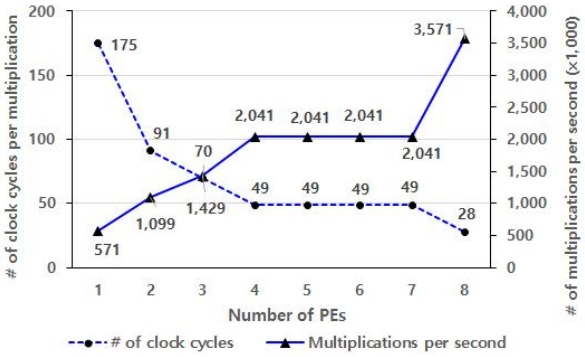
(*) 180-nm CMOS cell library

그림 7은 N_{PE}에 따른 sMM 코어의 등가게이트 수와 메모리 크기, 그리고 256-비트 모듈러 곱셈 연산에 소요되는 클록 사이클 수 및 초당 곱셈 횟수 등의 성능을 분석한 결과이다. 그림 7로부터, 사용되는 PE 개수 N_{PE}에 따라 sMM 코어의 하드웨어 복잡도와 연산성능의 조정이 가능하며, 따라서 응용분야에서 요구되는 연산성능과 리소스 제한을 고려한 최적화된 모듈러 곱셈기 구현이 가능함을 알 수 있다. 그림 8은 N_{PE}에 따른 sMM 코어의 등가게이트 당 256-비트 곱셈 횟수를 분석한 결과이며, N_{PE}=4, 5, 8인 경우에 최적의 효율이 얻어짐을 알 수 있다.

표 2는 문헌에 발표된 모듈러 곱셈기와 본 논문의 sMM 코어(L=256-비트, N_{PE}=8인 경우)의 성능을 비교한 결과이다. sMM 코어의 하드웨어 복잡도와 동작주파수는 사용되는 PE의 개수 N_{PE}에 따



(a) Hardware complexity



(b) Computational performance

Fig. 7. Performance of sMM core according to N_{PE}.
그림 7. N_{PE}에 따른 sMM 코어의 성능

라 영향을 받으며, N_{PE}=2인 경우의 동작주파수는 63 MHz로 예측되었다. DSP 블록과 LUT의 상대적인 하드웨어 크기에 관한 정보는 알려져 있지 않으므로, 하드웨어 복잡도의 간접적인 비교만 가능하다. 본 논문의 sMM 코어는 문헌 [13]에 비해 28%의 연산시간이 더 소요되지만 LUT와 DSP 블록을 각각 약 50%와 65% 적게 사용하며, 문헌 [14]와 비슷한 LUT를 사용하지만 연산시간이 약 50% 작다. 간접적인 비교를 위해, 1,024-비트 곱셈기인 문헌 [16]의 사례는 sMM 코어에 비해 약 10

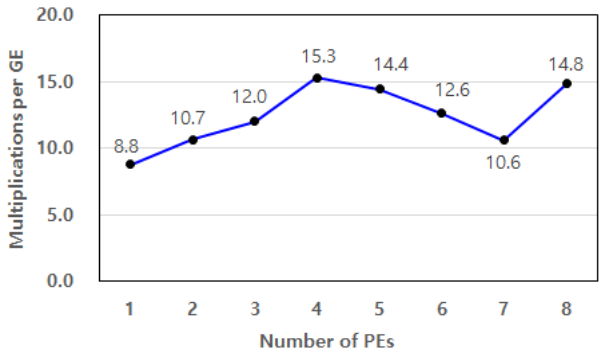


Fig. 8. Multiplications-per-GE performance of sMM core (for 256-bit).
그림 8. sMM 코어의 GE 당 곱셈 성능 (256-비트의 경우)

배의 LUT가 사용되며 연산시간은 약 1.4배 소요된다. 표 2에 제시된 성능 이외에 본 논문의 sMM 코어는 워드크기와 사용되는 PE 개수를 조정하여 임의의 크기의 모듈러 곱셈기를 쉽게 구현할 수 있다는 구조적 유연성의 장점을 갖는다.

Table 2. Comparison of modular multipliers.
표 2. 모듈러 곱셈기 비교

	This paper*	[13]	[14]	[15]	[16]**
Size of modulus [bit]	192, 224, 256, 384, 521	256	256	256	1,024
FPGA device	Vertex-5	Vertex-5	Vertex-6	Vertex-7	Vertex-7
DSP block	41	120	-	-	-
LUT [slice]	3.76k	7.32k	3.9k	605	39.9k
Frequency [MHz]	44.1	58.37	96	152.7	181
Latency [nsec]	635	497	1,300	1,683	910

(*) L=256-bit and $N_{PE}=8$.
(**) The word sizes of multiplier and multiplicand are 8-bit and 32-bit, respectively.

V. 결론

워드 단위의 연산 병렬성을 조정하여 연산성능과 하드웨어 복잡도를 최적화할 수 있는 확장 가능형 몽고메리 모듈러 곱셈 알고리듬(sWMMA)과 하드웨어 구조를 제안했다. 부분곱 생성과 가산 연산의 병렬성은 사용되는 PE의 개수에 의해 결정된다. 제안된 구조를 적용하여 워드 크기는 32-비트이고, PE를 1개에서 최대 8개까지 병렬로 사용할 수 있는 확장 가능형 몽고메리 모듈러 곱셈기 코어를 설계했다. $N_{PE}=1$ 및 $N_{PE}=8$인 경우에 각각 38,317 GEs 및 139,390 GEs로 구현되었으며, 100 MHz 클록으로 동작할 때, $N_{PE}=1$인 경우에 초당 57만회 및 $N_{PE}=8$인 경우에 초당 350만회의 256-비트 모듈러 곱셈을 연산할 수 있는 것으로 평가되어 사용되는 PE 개수 N_{PE}에 따라 연산성능과 하드웨어 복잡도 사이에 교환조건이 성립한다. 본 논문의 확장 가능형 모듈러 곱셈기 구조는 응용분야에서 요구되는 연산성능과 하드웨어 리소스를 고려하여 사용되는 PE 개수를 결정함으로써 최적화된 곱셈기 구현이 가능하다는 장점을 가지며, ECC의 확장 가능한 하드웨어 설계에 핵심 IP로 사용될 수 있다.

References

[1] N. Koblitz, “Elliptic curve cryptosystems,” *Mathematics of Computation*, vol.48, no.177, pp.203-209, 1987.

[2] S. Sugiyama, H. Awano and M. Ikeda, “Low Latency 256-bit Fp ECDSA Signature Generation Crypto Processor,” *IEICE Transaction on Fundamentals*, vol.E101-A, no.12, pp.2290-2296, 2018. DOI: 10.1587/transfun.E101.A.2290

[3] M. Kneževic, V. Nikov, and P. Rombouts, “Low-latency ECDSA signature verification-A road toward safer traffic,” *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, vol.24, no.11, pp.3257-3267, 2016.
DOI: 10.1109/TVLSI.2016.2557965

[4] ITS Committee, “IEEE standard for wireless access in vehicular environments-security services for applications and management messages,” *IEEE Vehicular Technology Society,* Vol.1609, No.2, 2013. DOI: 10.1109/IEEESTD.2016.7426684

[5] D. S. Kim and K. Y. Shin, “Montgomery Multiplier supporting Dual-Field Modular Multiplication,” *Journal of the Korea Institute of Information and Communication Engineering*, vol.24, no.6, pp.736-743, 2020.
DOI: 10.6109/jkiice.2020.24.6.736

[6] M. Amine, E. M. Nadia, L. Ronan, J. B. Rigaud, B. Belgacem, M. Sihem and M. Mohsen, “A Scalable and Systolic Architectures of Montgomery Modular Multiplication for Public Key Cryptosystems Based on DSPs,” *Journal of Hardware and Systems Security*, vol.1, issue3, pp.219-236, 2017. DOI: 10.1007/s41635-017-0018-x.

[7] M. Shieh and W. Lin, “Word-Based Montgomery Modular Multiplication Algorithm for Low-Latency Scalable Architectures,” *in IEEE Transactions on Computers,* vol.59, no.8, pp.1145-1151, 2010. DOI: 10.1109/TC.2010.72

[8] S. Kuang, C. Liang and C. Chen, “An Efficient Radix-4 Scalable Architecture for Montgomery Modular Multiplication,” *in IEEE Transactions on Circuits and Systems II: Express Briefs*, vol.63, no.6, pp.568-572, 2016.
DOI: 10.1109/TCSII.2016.2530801

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

[10] P. L. Montgomery, “Modular multiplication without trial division,” *Math. of Computation*, vol.44, no.170, pp.519-521, 1985.

[11] A. F. Tenca and C. K. Koc, “A Scalable Architecture for Montgomery Multiplication,” International Workshop on Cryptographic Hardware and Embedded Systems, Springer, Heidelberg, vol.1717, pp.94-108, 1999.

[12] J. B. Choi, “A Scalable ECC Processor Supporting Prime Field Elliptic Curves,” *Master Thesis, Kumoh National Institute of Technology,* 2021.

[13] K. Safiullah, J. Khalid and S. Y. Ali, “High-speed FPGA implementation of full-word Montgomery multiplier for ECC applications,” *Microprocessor and Microsystems*, vol.62, pp.91-101, 2018.
DOI: 10.1016/j.micpro.2018.07.005.

[14] K. Javeed, X. Wang and M. Scott, “Serial and parallel interleaved modular multipliers on FPGA platform,” *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, London, pp.1-4, 2015.
DOI: 10.1109/FPL.2015.7293986.

[15] M. Selim Hossain and Y. Kong, “FPGA-based efficient modular multiplication for Elliptic Curve Cryptography,” *2015 International Telecommunication Networks and Applications Conference (ITNAC)*, Sydney, NSW, pp.191-195, 2015,
DOI: 10.1109/ATNAC.2015.7366811.

[16] B. Zhang, Z. Cheng and M. Pedram, “High-Radix Design of a Scalable Montgomery Modular Multiplier with Low Latency,” *IEEE Transactions on Computers*, Accepted for publication, 2021.
DOI: 10.1109/TC.2021.3052999.

BIOGRAPHY

Jun-Baek Choi (Member)

2019: BS degree in Electronic Engineering, Medical IT Convergence Eng., Kumoh National Institute of Technology.
2019~ : Graduate student, Kumoh National Institute of Technology
2021~ : Design Engineer, Ranix Inc.

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~ : Professor at 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)