개선된 다정도 CSA에 기반한 모듈라 곱셈기 설계
(A Design of Modular Multiplier Based on Improved Multi-Precision Carry Save Adder)

김 대 영 † 이 준 용 ††
(Dae Young Kim) (Jun Yong Lee)

요 약 가산기를 이용하여 몽고메리 곱셈을 수행하는 모듈라 곱셈기를 구현하는 방법은 선택한 가산기의 종류에 따라 달라진다. 가산기로 CPA를 사용하는 경우는 캐리 전파 문제가 발생되며, CSA를 사용하는 경우는 최종 결과 보정이 요구된다. 다정도 CSA는 CSA와 CPA를 접목함으로써 이 두 문제를 동시에 해결한 방식이다. 본 논문에서는 기존의 다정도 CSA의 캐리 체인 구조를 변경함으로써, 하드웨어 자원과 수행시간을 동시에 감소시킨 새로운 방식을 제안하였다. 결과적으로, 모듈라 곱셈기를 반복 사용하여 큰 정수의 곱셈과 멱승을 수행하는 모듈을 기존의 방식보다 더 빠르고 더 작게 구현할 수 있다.
키워드 : 모듈라 곱셈기, 몽고메리 알고리즘, 다정도 CSA

Abstract The method of implementing a modular multiplier for Montgomery multiplication by using an adder depends on a selected adder. When using a CPA, there is a carry propagation problem. When using a CSA, it needs an additional calculation for a final result. The Multiplier using a Multi-precision CSA can solve both problems simultaneously by combining a CSA and a CPA. This paper presents an improved MP-CSA which reduces hardware resources and operation time by changing a MP-CSA's carry chain structure. Consequently, the proposed multiplier is more suitable for the module of long bit multiplication and exponentiation using a modular multiplier repeatedly.
Key words : Modular Multiplier, Montgomery Algorithm, Multi-precision CSA

1. 서 론

보안 서비스를 제공하는 공개키 암호 알고리즘에서, 가장 핵심적임 수학 연산은 모듈라 지수승이다[1]. 모듈라 지수승 A^E mod N은 AB mod N과 같은 모듈라 곱셈 연산이 수백, 수천번 필요하다. 특히, 지수값이 E가 512비트 이상의 큰 수일 때 매우 긴 계산시간을 필요로 한다. 모듈라 곱셈의 경우, 고전적인 방법으로 Barret 알고리즘과 몽고메리(Montgomery) 알고리즘[2]이 있으며, 더 빠른 구현은 몽고메리 알고리즘이 적합하다[3]. 몽고메리 알고리즘은 곱셈기의 내부 연산이 규칙적이고 데이타 흐름이 일정한 구조를 가지고 있어 현재까지 많은 연구가 되어 왔으며, 다양한 형태의 변형된 알고리즘이 존재하여 하드웨어 구현 방법 또한 다양하게 연구되어 왔다[4-7]. 몽고메리 곱셈기의 구현 방법은 크게 시스토릭 어레이(systolic array)[4]를 사용한 방법과 가산기를 사용한 방법으로 나뉜다. 시스토릭 어레이 기반의 모듈라 곱셈기는 처리 속도가 빠른 반면 많은 하드웨어 자원이 소모되며, 가산기 기반의 모듈라 곱셈기는 시스토릭 어레이 방법에 비해 적은 자원을 사용하는 대신, 캐리 전파 문제를 지니고 있다[]. Arazi가 제안한 DSD (Digital Signature Device)[8]는 가산기, 레지스터, 멀티플레서(MUX)만을 사용한 간단한 회로 구조로서 몽고메리 모듈라 곱셈을 수행할 수 있다. 이외에도, 구현에 많은 하드웨어 자원을 필요로 하지만 고속 연산이 가능한 radix 기반의 모듈러 곱셈기[9]도 있으며, 현재까지의 구현 방식중 가장 빠른 수행속도를 가진 RNS (Residue Number System) 기반의 모듈러 곱셈기[10]도 있다.

한편, 가산기를 기반으로 모듈라 곱셈기를 설계할 경우, 일반적으로 캐리 전파 문제를 해결하기 위해 CSA(Carry Save Adder)를 사용하는데 이 경우에는 곱셈 결과가 캐리(carry)와 합(sum)의 형태로 나누어지게 되므로 최종 결과를 얻기 위한 회로 및 추가 클럭이 필요하다. [11,12]에서 제시된 MP-CSA(Multi-Preci-

---

· 이 논문은 2003학년도 홍익대학교 교내연구비에 의하여 지원되었음
† 학생회원 : 홍익대학교 컴퓨터공학과
hansol@cs.hongik.ac.kr
†† 종신회원 : 홍익대학교 컴퓨터공학과 교수
jlee@cs.hongik.ac.kr
논문접수 : 2005년 3월 22일
심사완료 : 2005년 12월 29일

sion CSA, 다정도 CSA)는 CSA와 CPA를 접목한 방식으로써 부가적인 하드웨어를 사용하지 않고 추가 클럭을 통해 최종 결과 보정 문제를 해결한 방법이다.

본 논문에서는 기존의 MP-CSA의 개선된 구현 방법을 제안하고자 한다. 제안된 방식은 기존의 방법에 비해 하드웨어 자원과 수행시간을 동시에 줄인 장점을 지닌다. MP-CSA(Multi-Precision)는 n 비트의 모듈라 곱셈을 수행하기 위해 m(= n / b)개의 CPA를 사용하는 가산기이다. 여기서 b는 하나의 CPA를 구성하는 FA의 개수이다. 이 곱셈기에서는 중간 결과가 합과 캐리의 형태로 표현되며 다음 사이클에 다시 입력으로 보내진다. 최종 결과를 얻기 위해서는 합과 캐리를 보정해야 하기 때문에 추가 사이클이 필요하다. 따라서 한 번의 모듈라 곱셈을 수행하는 데 n+m 클럭이 소요된다. 개선된 방법에서는 효과적인 구현을 통해 클럭을 n+0.5m로 줄이면서 동시에 하드웨어 자원과 복잡도를 감소시켰다.

본 논문의 2장은 관련 연구로서 비트 단위의 몽고메리 알고리즘을 먼저 소개하고, 몽고메리 알고리즘을 이용한 모듈라 곱셈기의 기본구조 기술한 후, MP-CSA를 이용한 구현 방법을 살펴본다. 3장에서 새로 제안한 개선된 MP-CSA를 설명하고 결론을 맺는다.

2. 관련 연구

2.1 몽고메리 곱셈 알고리즘

공개키 암호 시스템에 사용하는 모듈라 멱승 A^E mod N은 모듈라 곱셈의 반복으로 이루어진다. 일반적인 모듈라 곱셈 AB mod N은 두수 A, B를 곱한 결과를 N으로 나눈 나머지를 취하는 연산이다. 몽고메리 모듈라 곱셈은 AB mod N가 아닌 ABR^{-1} mod N을 수행하는데 여기서 R은 N과 서로소인 N보다 큰 정수이다. 논문에서는 A, B 및 N을 모두 n비트라고 가정한다. 또, 각 정수를 기저(base) r로 표현하면 $A=\sum_{i=0}^{k-1}A[i]r^i$, $B=\sum_{i=0}^{k-1}B[i]r^i$, $N=\sum_{i=0}^{k-1}M[i]r^i$와 같이 k자리로 나타낼 수 있으며 r=2인 경우에는 k자리는 n비트와 같아진다.

몽고메리 알고리즘은 먼저 N보다 크고 N과 서로소인 R을 선택하여 div R이나 mod R을 간단히 계산할 수 있도록 $R=r^n$으로 하는 것이 일반적이다. 본 논문에서는 r=2이라 가정하며 몽고메리 곱셈은 $AB2^{-n}$ mod N으로 구체화 한다. 따라서 몽고메리 모듈라 곱셈 알고리즘은 r=2이고, N이 홀수인 경우 그림 1과 같이 간단히 나타낼 수 있다.

그림 1에서 단계 2.2는 A와 B를 서로 곱하기 위한 반복 과정이며 단계 2.3은 모듈라 감소를 위한 반복 단계이다. 또, 단계 2.3 및 2.4는 M[i]=0일 경우에는 T를

```
1  : T=0
2  : for i=0 to n−1 do {
2.1:     M[i] = T[0] ⊕ (A[i]B[0])
2.2:     T = T + A[i]B
2.3:     T = T + M[i]N
2.4:     T = T/2}
3  : return T
```

그림 1 몽고메리 알고리즘

단순히 오른쪽으로 쉬프트시키고, M[i]=1일 경우에는 T에 N을 더한 후 오른쪽으로 쉬프트시키는 것과 동일하다. 그림 1의 알고리즘은 n번의 루프를 돌면서 각 루프마다 2개의 n비트 가산기를 사용하고, 이 출력을 오른쪽으로 쉬프트 시킴으로써 하드웨어로 구현이 가능하다. 이 알고리즘에서 n번째 루프 후의 최종 계산 값은 $T=AB2^{-n}$ mod N이 됨을 확인할 수 있다. 경우에 따라서 n번의 루프가 끝나고 출력되는 결과 T가 N보다 클 수도 있다. 이 때는 간단한 뺄셈기를 이용하거나, 하드웨어를 추가하지 않으려면 루프를 2번 더 실행하여 최종 결과가 N미만이 되도록 조정한다[13].

2.2 모듈라 곱셈기의 기본 구조

그림 2는 n비트의 A, B, N을 입력으로 받아 $AB2^{-n}$ mod N을 출력하는 모듈라 곱셈기의 기본 구조이다. 그림에서 보듯이 이 모듈은 2개의 n비트 가산기와 중간 결과값을 저장하기 위한 레지스터로 구성된다. 그림 1의 단계 2.2를 수행하는 첫 번째 가산기는 중간 결과값을 저장하는 레지스터 출력 T와 A[i]B를 더한다. 그림 1의

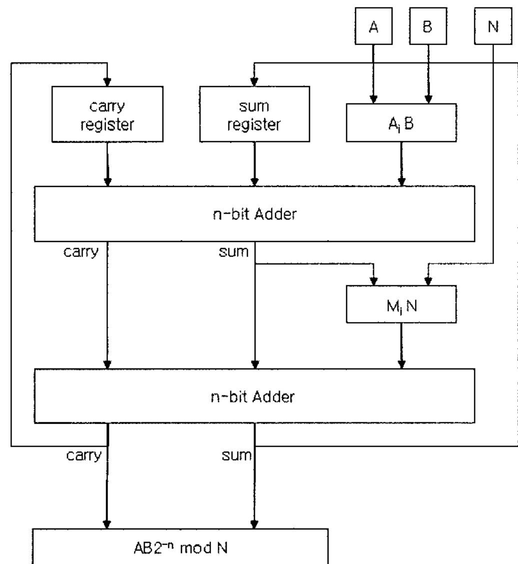


그림 2 모듈라 곱셈기의 기본 구조

단계 2.3을 수행하는 두 번째 가산기는 첫 번째 가산기의 결과값과 M[i]N을 더한다. 두 번째 가산기의 출력 중에서 최하위 비트를 제외한 나머지 비트들을 레지스터에 궤환(feedback)시킴으로써 그림 1의 단계 2.4에 해당하는 오른쪽으로 한 비트 쉬프트시키는 연산을 수행하게 된다. 따라서, 이 과정을 피승수 A의 각 비트 A[i]에 따라 1번 수행하면 $A[i]B2^{-1}$ mod N이 되고 n번 수행하면 $AB2^{-n}$ mod N의 결과를 얻게 된다.

이 곱셈기의 입력은 n비트의 B, N이 병렬로 입력되고, A는 1비트씩 직렬로 입력된다. 출력은 가산기의 선택에 따라 달라질 수 있으며, CPA를 사용할 경우 최종 결과를 곧바로 얻을 수 있지만, n의 값이 커서 캐리 전파 시간의 길어지기 때문에 현실적으로 어렵다. 대안으로 CSA(Carry Save Adder)를 사용할 수 있으며 이때는 곱셈기의 출력이 합과 캐리의 형태이기 때문에 최종 결과를 얻기 위해 보정을 해주어야 한다. 추가 회로 없를 사용하지 않으려면, n 클럭 이후 A[i], M[i]를 0으로 두고 n 클럭을 더 인가하여 2n만에 최종 결과를 얻을 수 있다.

2.3 MP-CSA를 이용한 모듈라 곱셈기

CSA를 사용한 방법은 CPA를 사용한 방법에 비해 캐리 전파 지연을 하나의 FA 지연으로 줄인 장점이 있는 반면, 다음 곱셈에 필요한 최종 결과를 얻기 위해서 합과 캐리의 보정이 필요하고, 결과적으로 추가 회로를 사용하지 않으려면 n 클럭을 더 인가해 주어야 한다. MP-CSA(다정도 CSA)는 캐리 전파와 결과 보정 문제를 해결하면서 동시에 추가 클럭 수를 줄인 방식이다. 이 방식에서는 하나의 가산기가 전체적으로는 CSA의 구조를 가지며, n비트를 m(m=n/b)개의 블록으로 나누어 내부는 b비트 CPA로 구성하는 방법이다. 그림 3은 n = 6, m = 2, b = 3일 경우의 회로 구성을 보여주고 있다.

6비트의 가산기를 2블록으로 나누었기 때문에 각 블록은 3비트 CPA가 처리하게 된다. 각 CPA는 한 클럭마다 캐리와 3비트의 합을 출력하고, 중간 결과들은 캐리 저장 형태로 레지스터에 저장된다. 따라서, 합을 저장하기 위한 n비트 레지스터와, 상단과 하단의 캐리를 저장하기 위한 2개의 b비트 레지스터(D-F/F)가 필요하다. 현재 클럭의 출력이 다음 클럭의 입력으로 들어가기 전에 오른쪽으로 한 비트 쉬프트되어야 하기 때문에((그림 1)의 2.4단계), 합은 레지스터 입출력 배핑을 통해 해결하고, 캐리는 블록사이에 존재하는 AX(AND/XOR) 유닛을 통해 처리한다. CSA를 사용했을 때와 마찬가지로 최종 결과를 얻기 위해서는 캐리와 합을 보정해야 하는데, 추가 회로 없이 해결하기 위해, A[i], M[i]의 입력에 0을 준 상태에서 m 클럭만 더 인가하면 된다. 따라서 전체 클럭은 CSA의 2n보다 적은 n+m이다. 최종 결과 값은 Tout 포트로 나오는 m개의 연속적인 비트들과, 합을 저장하는 레지스터의 n-m개 비트들(T[0] - T[n-m-1])이 된다.

이 방식의 핵심은 한 클럭 동안 하나의 CPA에서의 캐리 전파 시간을 최대화하는 것이다. 일반적으로 하나의 CPA를 구성하는 FA 개수는 한 클럭과 캐리 전파 시간을 고려해서 결정한다. 예를 들어, 곱셈기의 클럭 사이클이 5MHz라면, 한 클럭은 200 ns이다. 논리 게이트의 지연 시간을 4 ns라고 가정했을 때 FA의 전파 깊

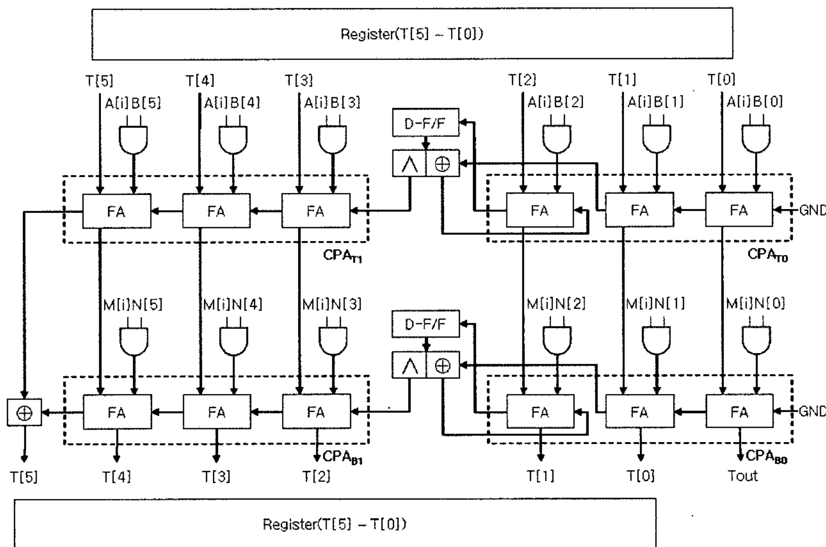


그림 3 MP-CSA 기반의 모듈라 곱셈기

이는 3이므로 전체 지연시간은 12ns이다. 따라서, 하나의 CPA에 들어가는 FA의 개수는 최대 ⌊200/12⌋ = 16개가 된다.

3. 개선된 MP-CSA를 이용한 모듈라 곱셈기

3.1 개선된 MP-CSA의 구조

캐리 저장 형태의 가산기를 사용한 모듈라 곱셈기는 모듈라 곱셈을 수행하는 기본 n 클럭 이외에, 합과 캐리를 보정한 최종 결과를 얻기 위한 추가 클럭이 필요하다. 즉, CSA의 경우는 n 클럭, MP-CSA의 경우는 m 클럭이 더 인가되어야 한다. 그런데, MP-CSA의 경우에 이 m 클럭 동안은 실제로 상단의 CPA만 사용되고, 하단의 CPA는 사용되지 않는다. 본 논문에서는 추가 클럭 동안 하단의 CPA를 활용할 수 있는 구조를 도입함으로써 추가 클럭을 절반으로 줄이고, 동시에 하드웨어 자원와 복잡도를 감소시킨 개선된 MP-CSA(이하 IMP-CSA, Improved Multi-Precision CSA)를 제안하고자 한다.

그림 4는 IMP-CSA의 구조를 보여주고 있다. 그림에서 보듯이 상단 가산기(CPA_{T0})에서 발생된 캐리아웃을 다음 블록의 하단 가산기(CPA_{B1})의 캐리인으로, 하단 가산기(CPA_{B0})의 캐리 아웃을 다음 블록의 상단 가산기(CPA_{T1})의 캐리인으로 교차해서 입력하는 것이 제안된 방식의 핵심이다. 이로 인해 얻게되는 이점은 첫 번째, 상단의 캐리를 따로 저장할 필요가 없으므로 캐리 저장용 레지스터를 절반으로 줄일 수 있고, 그림 1의 2.4 단계에 해당하는 쉬프트 연산을 처리하는 AX(and/xor) 모듈도 상단의 블록에는 필요 없고 하단에만 필요하게 되므로 이 회로 또한 절반으로 줄일 수 있다. 두 번째, 캐리 합 보정을 위한 추가 클럭 수행시, 첫 번째 블록에서 발생한 캐리아웃을 두 번째 블록의 하단에서 처리하게 되므로 한 클럭에 두 블록의 결과를 얻을 수 있어 추가 클럭을 절반으로 줄일 수 있다. 또한, 기존의 방식은 하나의 CPA에 두 비트의 캐리 아웃이 필요하기 때문에 독특한 구조를 갖고 있는 반면, 제안된 방식은 일반적인 CPA의 구조를 갖고 있어 구현이 더 용이하다.

3.1 IMP-CSA의 캐리 체인

본 절에서는 추가 클럭동안 한 클럭에 두 블록을 결과를 올바로 얻을 수 있는지 증명하고자 한다. 설명을 용이하기 위해 회로를 구성하는 기본 모듈들과 관련된 용어들을 그림 5와 표 1에 기술하였다.

하나의 CPA 모듈은 b비트의 두 입력(X,Y)과 캐리인(Cin)을 받아 b비트의 출력(Z)과 캐리아웃(Cout)을 발생시킨다. 추가 클럭 동안은 Y의 입력이 0이기 때문에 캐리아웃 발생조건은 X가 모두 1이고 캐리인이 1일 경우이다. AX 모듈은 두 비트(L, R)를 입력 받아서 and한 결과(A)와 xor한 결과(X)를 출력한다. 즉, AX 모듈은 반가산기(Half Adder)와 동일한 동작을 한다. 하단 가산기의 캐리아웃과 최하위 비트가 이 모듈의 입력으로 들어가서 쉬프트와 캐리 전파를 처리하게 된다.

그림 6은 4개의 블록으로 구성된 MP-CSA의 캐리 체인을 단순화하여 보여주고 있다. 점선으로된 화살표가

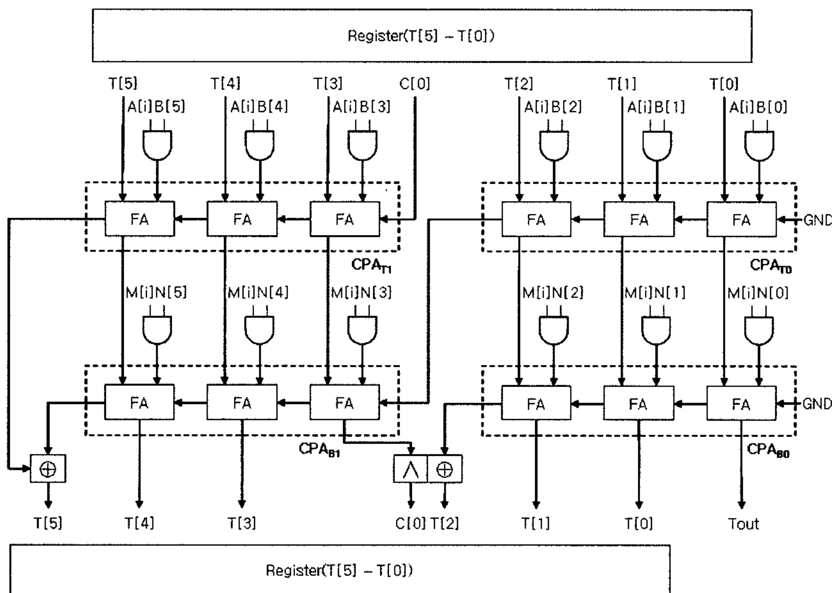


그림 4 IMP-CSA기반의 모듈라 곱셈기

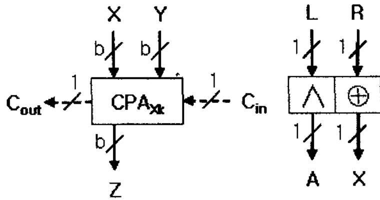


그림 5 CPA와 AX 모듈의 구조

표 1 CP

표기	의미
CPA_{Tk}	k번째 상단 CPA
CPA_{Bk}	k번째 하단 CPA
$CPA_{Tk}.X$	k번째 상단 CPA의 b비트 입력 X
$CPA_{Tk}.Y$	k번째 상단 CPA의 b비트 입력 Y
$CPA_{Tk}.Z$	k번째 상단 CPA의 b비트 출력 Z
$CPA_{Tk}.Ci$	k번째 상단 CPA의 캐리인(Cin)
$CPA_{Tk}.Co$	k번째 상단 CPA의 캐리아웃(Cout)
$CPA_{Bk}.Z[i]$	k번째 하단 CPA 출력 Z의 i번째 비트
AXk	k번째 and/xor 유닛
AXk.L	k번째 AX의 좌측 1비트 입력
AXk.R	k번째 AX의 우측 1비트 입력
AXk.A	두 입력 비트의 and 결과 출력 A
AXk.X	두 입력 비트의 xor 결과 출력 X

캐리 체인을 나타낸다. 보시다시피 상하단 각각에서 발생하는 캐리를 레지스터에 저장했다가 다음 블록에 전달하기 때문에 한 클럭에 한 블록의 결과만을 얻을 수 있다. 즉, 최종 결과를 얻기 위해서는 반드시 m 클럭을 더 인가해야 한다.

그림 7은 그림 6을 제안한 IMP-CSA로 구현했을 때의 캐리 체인의 흐름도이다. 블록과 블록 사이에는 두 개의 캐리 체인이 존재하는데 하나는 현재 블록의 상단에서 다음 블록의 하단으로 흘러가는 C_{TBx}, 다른 하나는 현재 블록의 하단에서 다음 블록의 상단과 흘러가는 C_{BTx}이다. 두 캐리의 값은 다음 식에 의해 결정된다.

- $C_{TBk} = CPA_{Tk}.Co$
- $C_{BTk} = CPA_{Bk}.Co$ and $CPA_{Bk+1}.Z[0]$

C_{BTk}는 그림 1의 2.4 단계에 해당하는 쉬프트 연산을 처리하기 위한 AX 모듈을 통과하기 때문에 CPA_{Bk}에서 발생한 캐리아웃(Co) 값 그대로가 아니라 CPA_{Bk+1} 출력 값의 최하위 비트(Z[0])와 and한 값이 된다. 또한, 그림 6에는 캐리 체인을 단순화하기 위해 표시되지 않았지만, 그림 4에서 보듯이 C_{TBx} 캐리는 레지스터에 저장하지 않고 곧바로 전달되며, C_{BTx} 캐리는 레지스터에 저장되었다가 다음 클럭에 전달된다. 이상과 같은 조건에서 추

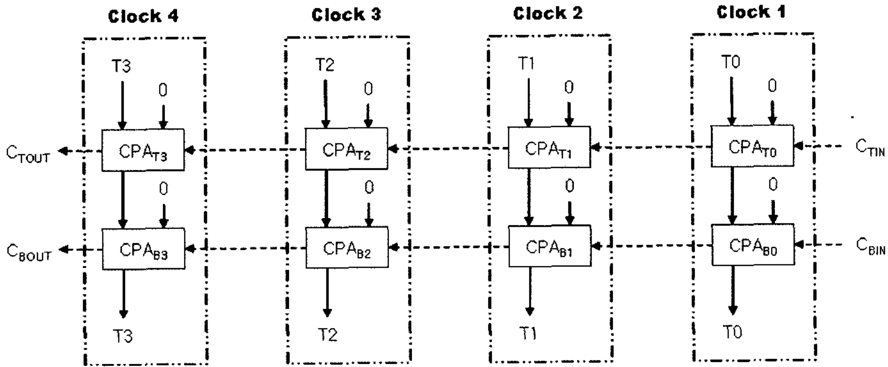


그림 6 MP-CSA의 캐리 체인

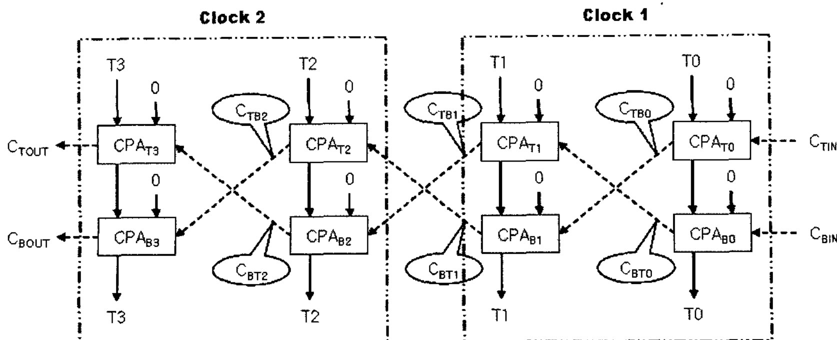


그림 7 제안한 IMP-CSA의 캐리 체인

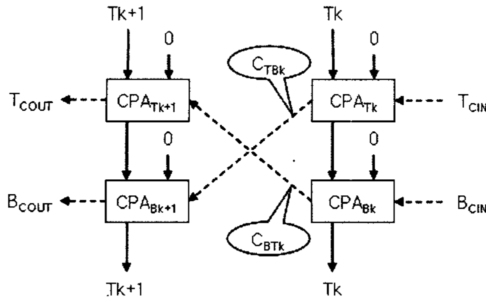


그림 8 추가 클럭동안 한 클럭에 처리해야하는 두 블록

가 클럭동안 한 클럭에 두 블록의 결과를 올바르게 얻을 수 있는지 검증해보도록 하겠다.

그림 8은 제안된 방식에서 한 클럭에 처리해야 하는 두 블록을 따로 떼어놓은 그림이다. 한 클럭 안에 상단과 하단의 두 가산기가 차례대로 수행되기 때문에, 두 블록 사이의 중간 캐리중에서 C_{TBk}는 문제가 되지 않지만, 레지스터에 저장되었다가 다음 클럭에 전달되는 C_{BTk}가 1이 되면, 이 캐리를 처리하기 위해 한 클럭이 더 인가되어야 하므로, 두 블록을 처리하는 데 두 클럭을 사용해야 한다. 하지만 이 추가 클럭 동안은 가산기의 입력이 특수하기 때문에 한 클럭에 두 블록을 처리하는 것이 가능하다. 표 2는 그림 8의 두 블록에 입력되는 캐리인에 따른 최악의 상황을 고려한 캐리아웃 값과 소요 클럭을 계산해 놓은 표이다.

표 2 캐리 입력에 따라 2 블록을 처리하는데 소요되는 Worst Case Clock

TCin	BCin	TCout	BCout	Worst Case Clock
0	0	0	0	1 clock
0	1	1	0	2 clock
1	0	0	1	1 clock
1	1	1	0	2 clock

추가 클럭 동안에는 모든 CPA 모듈의 입력 Y가 0이기 때문에 C_{TBk}와 C_{BTk}가 1이 되는 조건은 다음과 같다.

- C_{TBk}

$CPA_{Tk}.X$가 모두 1이고, $CPA_{Tk}.Ci = 1$ 일 때

- C_{BTk}

$CPA_{Bk+1}.Z[0] = 1$이고 $CPA_{Bk}.Co = 1$ 일 때

C_{TBk}와 C_{BTk}가 1이 되는 조건이 다른 이유는, 앞서 언급한 바와 같이, 상단 가산기의 출력은 쉬프트를 고려하지 않지만 하단 가산기의 출력은 쉬프트를 처리하기 위해 AX 모듈을 거쳐야 하기 때문이다. 이해를 돕기 위해 T_{CIN}과 B_{CIN}이 둘다 1인 경우를 예로 들어 설명하겠다. 최악의 상황은 Tk=110, Tk+1=111이다. 이 경우, C_{TBk}는 $CPA_{Tk}.X$가 모두 1이 아니므로 0이 된다. $CPA_{Tk}.Z$ 값인 111이 $CPA_{Bk}.X$에 전달되어 $CPA_{Bk}.Co$가 1이 되고, $CPA_{Bk+1}.Z[0]$도 1이므로 C_{BTk}는 1이 된다. 결과적으로 C_{BTk} 캐리아웃은 다음 클럭에 처리할 수 밖에 없다. 다음 클럭에, 캐리인(C_{BTk})이 1이므로 T_{COUT}은 1이 되고, B_{COUT}은 이전 클럭에 캐리 전파가 끝났으므로 0이 출력된다. 따라서, 세 번째 클럭에 그 다음 두 블록에 T_{CIN}으로 1이, B_{CIN}으로 0이 입력되어, 표 2에 명시된 것처럼 한 클럭 후에 $T_{COUT}=0$, $B_{COUT}=1$이 출력된다. 결과적으로 초기 캐리인이 모두 1인 경우에는 4 블록을 처리하는데 총 3 클럭이 필요하게 된다. 이런식으로 두 블록을 하나의 단위로 묶어 캐리 전파를 고려했을 때 각각의 소요 클럭은 표 2에 명시된대로 입력되는 캐리인에 따라 다르게 된다.

표 3에 총 m(4의 배수로 가정)개의 블록을 가진 모듈라 곱셈기의 초기 캐리인에 따른 총 소요 클럭을 정리해 놓았다.

표 3 m 블록 캐리 전파에 필요한 클럭수

CTIN	CBIN	4 블록에 소요되는 클럭수	m 블록에 소요되는 클럭수
0	0	0.5× 4 = 2	m/4 * 2 = 0.5m
0	1	0.5× 2 + 1× 2 = 3	m/4 * 3 = 0.75m
1	0	0.5× 2 + 1× 2 = 3	m/4 * 3 = 0.75m
1	1	1× 2 + 0.5× 2 = 3	m/4 * 3 = 0.75m

그림 4에서 보듯이 맨 처음의 두 캐리 입력은 항상 0(GND)이므로 표 3에서 C_{TIN}, C_{BIN}이 둘다 0인 경우에 해당한다. 따라서, 항상 0.5m 추가 클럭 만에 합과 캐리가 보정된 최종 결과를 얻을 수 있음을 확인할 수 있다.

4. 성능 평가

IMP-CSA를 사용한 몽고메리 곱셈기의 성능을 평가하기 위해 곱셈기 회로를 Xilinx FPGA Virtex-4[16]을 사용하여 설계하였고 타겟 디바이스는 xc2s100-5pq208으로 설정하였다.

먼저, 표 4는 MP-CSA와 IMP-CSA의 하드웨어 자원과 시간을 이론적으로 비교한 결과이다. 레지스터(D 플립플롭)와 FA(전가산기)는 5개의 게이트로 구성되는 경우를 가정하였다.

표 4 MP-CSA와 IMP-CSA 방식의 비교

구분		MP-CSA	IMP-CSA
수행시간		n+m	n+0.5m
복잡도	Register(5)	n+2m	n+m
	FA(5)	2n	2n
	AND(1)	2n+2m	2n+m
	XOR(1)	2m	m
	합계	17n+14m	17n+7m

우선 수행시간을 보면 기본 클럭은 동일하게 n이며, 추가 클럭에서 제안된 방식이 하단의 가산기를 활용하므로 클럭 타임이 절반(0.5m)으로 줄어들게 된다. 자원의 경우, 상단의 캐리를 저장하지 않고 즉시 다음 블록의 하단으로 보내어 처리하기 때문에, m개의 D 플립플롭, AND, XOR 유닛을 절약할 수 있다. 즉, 제안된 방식은 기존의 방식에서 필요한 2m개의 자원을 절반인 m으로 줄인 효과가 있다. 그림 9와 그림 10은 각각 32개의 블록으로 구성한 1024비트 크기의 MP-CSA와 IMP-CSA를 Xilinx 툴을 이용하여 구현하였을 때의 시간과 자원 사용량을 비교한 그림이다. 타겟 디바이스의 FA의 지연시간은 대략 0.6ns 정도이고, 합성 과정에서 레지스터는 FlipFlop으로 FA와 AX 모듈은 LUT로 변환되어 표시되었다. LUT3로 변환되는 FA의 개수는 두 방식이 동일하므로 그림에서와 같이 동일하며 LUT2로 변환되는 AX모듈과 레지스터의 사용량은 m(m=32) 정도 만큼 줄었음을 확인할 수 있다.

그림 11은 CPA를 구성하는 FA의 지연시간을 1ns로 가정한 상태에서, 두 방식의 모듈라 곱셈기를 사용하여 1024비트 곱셈을 수행할 때의 총 클럭과 총시간을 비교한 결과이다.

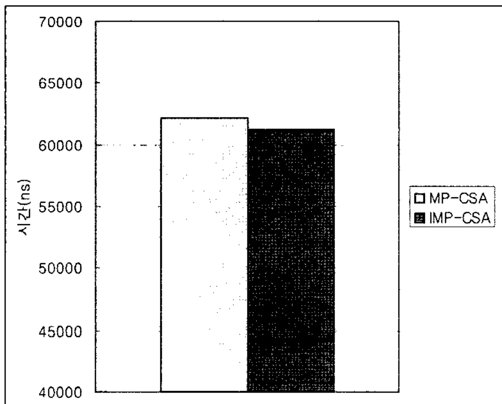


그림 9 1024비트 모듈의 시간 비교

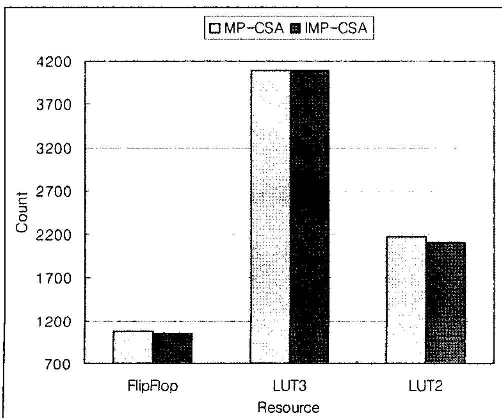


그림 10 1024비트 모듈의 자원 비교

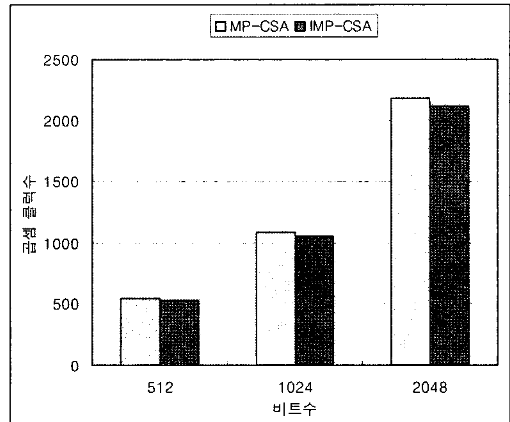


그림 11 1024 비트의 곱셈 수행 결과 비교

MP-CSA는 클럭 주파수를 올릴수록, 하나의 CPA를 구성하는 FA의 개수가 줄어들기 때문에, 전체 CPA의 개수가 늘어나지만 낮은 주파수보다 빠른 성능을 보이게 된다. 하지만 CPA의 개수가 늘어나면 추가 클럭도 늘어나는 단점이 존재한다. 이러한 점을 고려할 때, 제안된 IMP-CSA 방식은 다른 조건이 동일한 상태에서 추가 클럭을 절반으로 줄였기 때문에, 높은 주파수를 사용할 때 MP-CSA비해 더 이점을 얻을 수 있다.

하나의 모듈라 곱셈기에서 추가 클럭을 절반으로 줄이는 것은 성능 개선에 있어서 큰 이점이 없어 보이지만, 이러한 클럭 곱셈기를 수백, 수천번 반복 사용하는 멱승기에서는 수행시간에서 큰 이점을 얻게 된다. 그림 12는 주파수가 50MHz일 때, 두 방식의 모듈라 곱셈기

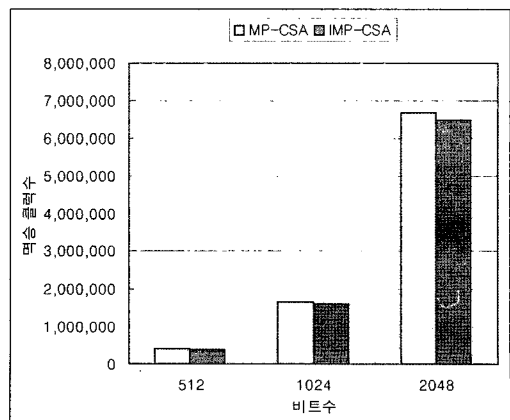


그림 12 50HMz에서 RSA 암호화 수행에 필요한 클럭수 비교

를 LR 방식[14]으로 반복 사용하여 512, 1024, 2048 비트의 RSA 암호화를 수행할 때 필요한 평균 클럭수(1.5n)를 비교한 결과이다.

RSA 암호 모듈과 같이 큰 키 값을 갖는(512-2048) 모듈에 적용했을 때, 동일한 주파수 하에서 n의 값이 커질수록 CPA 블록의 개수 m이 증가하므로, 제안된 방식의 이점이 더욱 두드러진다. 1024 비트인 경우에 암호화를 수행하는 데 필요한 멱승 클럭이 49,152 클럭 줄어들었고, 2048 비트인 경우에는 196,608 클럭 줄어들었다. 이 결과만 보더라도 비트수가 2배 증가하였을 때 클럭수가 단순히 2배만큼 줄어드는 것이 아니라 그 이상 줄어드는 것을 확인할 수 있다.

5. 결 론

MP-CSA 기반의 모듈라 곱셈기는 m개의 b비트 CPA로 구성되어 n 비트의 모듈라 곱셈을 수행하는 모듈로, 캐리 전파 깊이를 최대화하면서 하나의 CPA가 한 클럭에 수행되도록 하는 것이 핵심이다. 기존의 방법은 최종 결과를 얻기 위해 블록의 개수(m)에 해당하는 만큼의 추가 클럭이 필요한데, 본 논문에서는 상하단의 캐리 교차 입력을 통하여 추가 클럭 동안 하단 가산기의 활용하는 구조로 개선함으로써, 기존의 방식보다 추가 클럭 타임을 절반(0.5m)으로 줄이고 CPA의 캐리 결과를 저장하는 레지스터를 하단에만 둠으로써 하드웨어 자원 또한 줄일 수 있는 방법을 제시하였다. 또한, 기존의 방식처럼 특수한 CPA 구조를 사용하지 않고, 일반적인 CPA 구조를 사용하여 하드웨어 설계시 복잡도를 감소시켰다. 따라서, 제안된 모듈은 IC 카드와 같이 적은 하드웨어 자원과 빠른 연산 시간을 필요로하는 암호 모듈에 설계하는데 기존의 방식보다 더 적합하다고 할 수 있다.

참 고 문 헌

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signature and public-key cryptosystems," Comm. of ACM, Vol. 21, pp. 120-126, 1978.

[2] P. L. Montgomery, "Modular multiplication without trial division," Math. Computation, Vol. 44, pp. 519-521, 1985.

[3] Antoon Bosselaers, Rene Govaerts, and Joos Vandewalle, "Comparison of Three Modular Reduction Functions," Advances in Cryptology-CRYPTO '93, pp. 175-186, August, 1993.

[4] C. D. Walter, "Systolic Modular Multiplication," IEEE Trans. on Computers, Vol. 42, pp. 376-378, 1993.

[5] S. E. Eldridge and C. D. Walter, "Hardware implementation of Montgomery's modular multiplication algorithm," IEEE Trans. on Computers, Vol. 42, No. 6, pp. 693-699. 1993.

[6] S. E. Eldridge, "A Faster Modular Multiplication Algorithm," Intern. J. Computer Math, Vol. 40, pp. 63-68, 1991.

[7] C. Y. Su, S. A. Hwang, P. S. Chen, and C. W. Wu, "An improved Montgomery's algorithm for high-speed RSA public-key cryptosystem," IEEE Trans. on Very Large Scale Integration(VLSI) Systems, Vol. 7, No. 2, 1999.

[8] B. Arazi, Digital Signature Device, US Patent #5448639, 1995.

[9] N. Takagi, "A radix-4 modular multiplication hardware algorithm for modular exponentiation," IEEE Trans. Computers, Vol, 41, pp. 949-956, 1992.

[10] A. A. Hiasat, "New efficient structure for a modular multiplier for RNS," IEEE Trans, on Computers, Vol. 49, pp. 170-174, 2000.

[11] 허준희, 하재철, 문상재, "다정도 CSA를 이용한 고속 모듈라 곱셈기", 한국 통신정보보호학회 학술대회, Vol. 9, No. 1, pp. 541-55, 1999. 4.

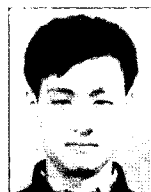
[12] J. C. Ha and S. J. Moon., "A Desing of Modular Multiplier Based on Multi-Precision Carry Save Adder," Joint Workshop on Information Security and Cryptology(JWISC'2000), pp. 45-51, Jan. 2000.

[13] T. Blum and C. Paar, "Montgomery modular exponentiation on reconfigurable hardware," in Proc. 14th IEEE Symp. on Computer Arithmetic, pp. 70-77, 1999.

[14] D. E. Knuth, "The Art of Computer Programming, Vol. 2: Seminumerical Algorithms," Addison-Wesley, 2nd Edn. 1981.

[15] William Stallings, "Cryptography and network security :principles and practice," Prentice Hall, 3nd Edn, 2002.

[16] Xilinx, Inc. url http://www.xilinx.com/



김 대 영

1998년 홍익대학교 컴퓨터공학과(학사)
2001년 홍익대학교 전자계산학과(석사)
2001년~현재 홍익대학교 컴퓨터공학과 박사과정

이 준 용

1986년 서울대학교 공과대학 컴퓨터공학과(학사). 1988년 미국 미네소타 주립대(석사). 1996년 미국 미네소타 주립대(박사). 1996년~1997년 미국 IBM 연구원. 1997년~현재 홍익대학교 컴퓨터공학과 교수