

32 비트 곱셈기에 기반한 몽고메리 모듈러 곱셈기 하드웨어 모듈 개발

양인제, 김동규
부산대학교 컴퓨터공학과

Development of Hardware Modules for Montgomery Modular Multipliers based on 32-bit multipliers

In Je Yang⁰ Dong Kyue Kim
Dept. of Computer Engineering, Pusan National University

요약

RSA 등의 공개키 암호화 시스템에서는 매우 큰 정수에 대해서 모듈러 벡승을 수행한다. 그러므로 모듈러 벡승을 효율적으로 구현하기 위하여 많은 연구가 진행되어 왔다. 모듈러 벡승을 소프트웨어적으로 구현할 경우 시간적인 제약을 극복하지 못하므로, 이를 하드웨어로 구현하려는 연구도 많이 이루어지고 있는 추세이다. 몽고메리 곱셈 알고리즘은 비용이 많이 드는 모듈러 연산을 효율적으로 처리하고 있으므로 하드웨어적 구현에 현재 널리 쓰이고 있다. 몽고메리 곱셈 알고리즘은 내부적으로 당연히 곱셈연산을 주로 사용하기 때문에, 어떤 곱셈기를 사용하느냐가 성능에 영향을 미치게 한다. 본 논문에서는 몽고메리 곱셈기를 다양한 32비트 곱셈기를 적용해 보고, 성능 및 면적을 측정하였다. 이러한 측정 결과를 토대로 특정 용용에 알맞은 32비트 곱셈기를 적절히 선택하여 설계할 수 있을 것으로 기대한다.

1. 서론

통신 기술의 발달로 인해 많은 정보들이 인터넷과 같은 네트워크를 통하여 전파되고 있다. 이러한 정보 중에서 인터넷 뱅킹이나 온라인 결제 시스템에서 필요로 하는 것과 같은 중요한 정보는 불법적으로 접근 할 수 없도록 하기 위하여 암호화 과정이 수행되고 있다. 즉 공개키 암호화 알고리즘을 이용하여 신뢰성 있는 시스템을 구축하고 있는 것이다. 그런데 공개키 알고리즘 중에서 모듈러 벡승에 기반한 알고리즘들은 내부적으로 곱셈 연산의 반복을 통해서 데이터를 암호화/복호화 하는데 이를 연산자들의 크기가 매우 크므로(1024비트 이상) 빠른 속도를 요구하는 환경에서는 적합하지 않다. 그러므로 빠른 속도를 요구하는 환경에서는 암호 알고리듬이 하드웨어 모듈에 탑재 되어 동작되는 것이 바람직하다.

본 논문에서는 공개키 알고리즘 중에서 모듈러 벡승에 사용되는 몽고메리 알고리즘[1]을 하드웨어 모듈로 구현하는데, 다양한 32비트 곱셈기를 적용하여 곱셈기의 선택이 전체 성능에 어떤 영향력을 미치는지 알고 보고자 한다.

공개키 암호화 알고리즘에서 다루는 매우 큰 정수가 부호 없는 정수이므로 2장에서는 부호 없는 32비트 곱셈기의 특징과 동작 방식을 설명하고 3장에서는 몽고메리 곱셈 알고리즘의 개념과 32비트 곱셈기를 이용하여 설계한 몽고메리 곱셈기에 대해서 알아본다. 마지막으로 4장에서는 2장에서 언급한 32비트 곱셈기의 성능과 이를 몽고메리 곱셈기에 적용한 몽고메리 곱셈기의 성능에 대해서 알아보고 5장에서 결론을 내린다.

2. 32 비트 곱셈기

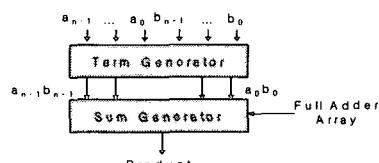
2.1 어레이 곱셈기

n비트 부호 없는 정수 $A = \sum_{i=0}^{n-1} a_i 2^i$, $B = \sum_{i=0}^{n-1} b_i 2^i$ 의 곱 $P = A * B$ 는 아래와 같이 기술 할 수 있다.

	a3	a2	a1	a0
X	b3	b2	b1	b0
	a3b0	a2b0	a1b0	a0b0
	a3b1	a2b1	a1b1	a0b1
	a3b2	a2b2	a1b2	a0b2
	a3b3	a2b3	a1b3	a0b3
p7	p6	p5	p4	p3
				p2
				p1
				p0

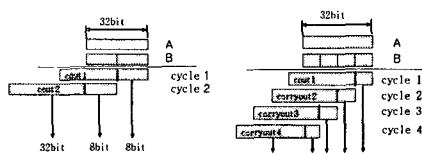
[그림 1. 4비트 x 4비트]

위와 같이 기술 했을 때 합은 위에서 아래로 내려 올 것이며 전파되는 캐리는 대각선 아래 방향으로 내려 오게 된다. 따라서 이러한 로직을 수행하기 위해서는 $a_i b_j (0 < i, j < n-1)$ 를 구하는 로직(Term Generator)과 이들을 더하는 로직(Sum Generator)이 있어야 한다. 이를 그림으로 나타내면 아래와 같다.



[그림 2. n x n 비트 어레이 곱셈기]

또한 이 논문에서는 면적을 줄이기 위해서 어레이 곱셈기를 2, 4클럭[2]에 동작 할 수 있도록 아래와 같이 변형하였다.



[그림 3. 어레이 곱셈기의 변형]

2.2 부분곱 곱셈기

n비트 부호 없는 정수 A,B의 곱 P를 구하기 위해서 A와 B를 각각 작은 부분으로 나누어 곱하고 이들의 결과를 더하는 방식[3]이다.

$P = A * B = (A_H * A_L) * (B_H * B_L) = A_H B_H + A_L B_H + A_H B_L + A_L B_L$

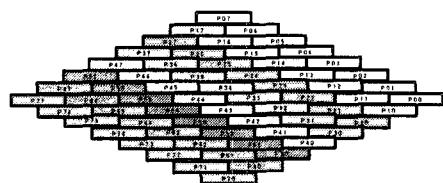
아래 그림은 8비트를 각각 4비트로 나누어서 부분곱을 구해서 더하는 과정을 나타낸 것이다.



[그림 4. 8비트 곱셈기의 배치도]

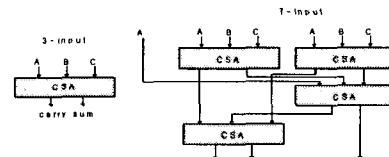
그림의 오른쪽에서 중첩되는 부분은 Carry Saved Adder(3:2)를 사용하여 합과 캐리로 나타낼 수 있고 이를 캐리 전파 덧셈기를 이용하여 최종결과를 더할 수 있다. 또한 각각의 부분합을 구하는 로직은 위에서 설명한 어레이 곱셈기를 사용하였다.

다음 [그림 5]는 위의 방식을 이용하여 32비트 곱셈을 부분곱을 구하고 더하는 구조를 나타낸 것이다.



[그림 5. 32비트 부분곱 곱셈기의 배치도]

또 이를 더하기 위해서 월리스 트리[4]를 사용하였다. 월리스 트리의 기본 소자는 아래와 같다.



[그림 6. 월리스 트리의 기본 소자]

2.3 수정된 부스 곱셈기

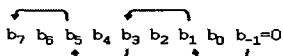
가장 간단한 곱셈 알고리즘인 Shift & Add 곱셈기는 숭수의 비트가 1이면 피승수를 더해주고 0이면 피승수 대신 0을 더해준다. 그러므로 숭수의 비트가 0이 되는 경우에 덧셈과정을 생략한다면 곱셈 수행 속도는 높일 수 있다.

부스 알고리즘[5]은 연속적인 0이나 1을 처리하는데는 우수하지만 피승수에서 0과 1이 번갈아 나오는 경우는 오히려 곱셈 횟수가 증가하는 단점이 있다. 그러므로 숭수의 비트에 의존해서 수행 속도가 달라진다. 이러한 단점을 개선하여 일정한 속도를 제공하는 알고리즘이 수정된 부스 알고리즘이다.

수정된 부스 알고리즘은 숭수의 비트를 한 비트가 중첩되면서 여러 비트(radix4, radix8)로 묶어서 곱셈을 수행하는 것이다. 결과적으로 덧셈의 수는 3비트씩 묶었을 경우 N/2번 정도, 4비트씩 묶었을 경우 N/3번 정도가 되어 덧셈의 횟수를 줄일 수 있게 되는 것이다. 이 논문에서는 3비트씩 묶는 방법 방법인 래디스4를 다른 32비트 곱셈기와 비교하는 것으로 한다.

8비트 피승수에 대한 수정된 래디스4 부스 인코딩

[6]은 다음과 같다.



중첩되는 세그먼트의 비트값에 따라서 다음과 같은 연산을 실시한다.

	b_{i+1}	b_i	b_{i-1}	Partial Product
case1	0	0	0	+ 0A
case2	0	0	1	+ 1A
case3	0	1	0	+ 1A
case4	0	1	1	+ 2A
case5	1	0	0	- 2A
case6	1	0	1	- 1A
case7	1	1	0	- 1A
case8	1	1	1	- 0A

[표 1. 래더스4의 비트별 부스 인코딩 방법]

각각 1A는 A를 더해주고, -1A는 A의 2의 보수를 취한 뒤 더해주고, 2A는 왼쪽으로 쉬프트 된 A를, -2A는 -A를 왼쪽으로 쉬프트 하여 더해 주면 된다.

3. 몽고메리 곱셈기의 설계

공개키 암호 시스템에서 수행되는 모듈러 곱셈 연산 ($AB \bmod N$)은 두 수(A,B)를 곱한 결과를 모듈러스 N 에 대해서 나머지를 취하는 연산이다. 단 여기서 A,B,N 은 k -비트 이진(binary)수이다. RSA 시스템에서는 모듈러 곱셈연산을 반복적으로 수행($R=C^E$)하므로 효율적인 곱셈기를 설계해야만 한다. 이러한 이유로 많은 곱셈기들이 제안되었고 그 중에서 현재 가장 많이 쓰이고 있는 곱셈기는 몽고메리 알고리즘에 기반한 곱셈기이다. 모듈러스 N 이 k -비트 이진수($2^{k-1} \leq N \leq 2^k$)이고 $r=2^k$ 일 때, 이 곱셈 알고리즘은 새로운 나머지 클래스에서의 두 수 A,B 에 대하여 $C=AB \bmod N$ 을 수행하는 것 대신에 $C=ABr^{-1} \bmod N$ 연산을 수행한다.

알고리즘1: 워드 단위 몽고메리 알고리즘

input: $A=(a_{n-1}, \dots, a_0), B=(b_{n-1}, \dots, b_0), N(n_{n-1}, \dots, n_0)$

$\gcd(N,W)=1, n_0'=N^{-1} \bmod W, r=W^s$

$T=0;$

for i to s-1

step 1. $m_i = (T_0 + A_i B_0) * n_0' \bmod W$

step 2. $T = (T + A_i B + m_i N)/W$

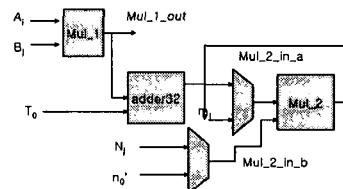
end for

step 3. *if T>N then T-N*

else return T;

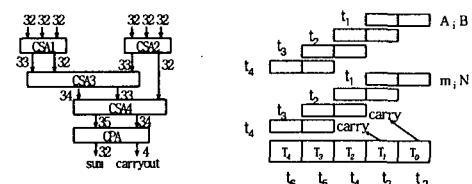
이 논문에서는 32비트 곱셈기를 이용하여 1024비트 몽고메리 알고리즘을 수행하는 것이 목표이므로 $W=2^{32}, s=32$ 가 된다.

step 1에서 m_i 를 구하기 위해서 곱셈 두 번, 덧셈 한번을 해야 한다. m_i 가 구해지면 step 2에서 워드 단위로 곱셈을 실시한다. 그러기 위해서 인덱스 j 를 추가하였다. 아래 [그림 7]은 두개의 곱셈기를 공유하여 step 1의 m_i 와 step2의 $A_i B_j$ 와 $m_i N_j$ 곱셈을 하는 로직이다. m_i 는 Mul_1 , adder32, Mul_2 를 거쳐 연산된다.



[그림 7. m_i 를 구하는 로직]

아래 그림은 4워드 몽고메리 알고리즘의 계산 순서를 도시한 것이다. 그림의 오른쪽은 step 2에서 $A_i B_j$ 와 $m_i N_j$ 를 워드 단위로 곱하는 순서를 나타낸 것이다. 시간 t_j 에 곱셈 연산이 일어나면 그 다음 시간 t_{j+1} 에 왼쪽 그림처럼 하위 워드 덧셈 연산을 수행하는데 피연산자는 오른쪽 그림의 한 컬럼에 있는 5개의 값과 캐리가 된다. 이와 같은 과정을 워드의 길이 만큼인 $s+2$ 번 반복 수행한다.

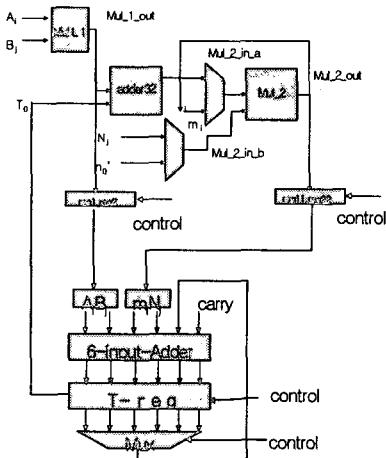


[그림 8. 곱셈과 덧셈 연산 수행 순서]

아래 [그림9]는 몽고메리 곱셈기 코어를 나타낸 것이다.

알고리즘1에서 m_i 를 구하기 위해서 로직은 $Mul_1(A_i B_0), adder32(T_0 + A_i B_0), Mul_2((T_0 + A_i B_0) * n_0')$ 를 순차적으로 수행한다. 이제 m_i 를 이용해서 다시 $Mul_1(A_i B_j), Mul_2(m_i N_j)$ 를 구하고 6-input-Adder를 이용하여 덧셈을 수행한다. 그리고 그 결과를 T-레지스터에 저장한다. 또한 알고리즘1의 step 3을 비교 및 빨셈 연산을 수행하지 않기 위해서 알고리즘을 변형[7]하였다. 이렇게 할 경우 원래 몽고메리 알고리즘의

결과가 달라지는데 이것은 전처리 및 후처리 과정을 통해 보정가능하다.



[그림 9. 몽고메리 곱셈기 코어 로직]

알고리즘1에서 m_i 를 구하기 위해서 로직은 $Mul_1(A; B_0)$, $adder32(T_0 + A_i B_0)$, $Mul_2((T_0 + A_i B_0) * n_0')$ 를 순차적으로 수행한다. 이제 m_i 를 이용해서 다시 $Mul_1(A; B_j)$, $Mul_2(m_i; N_j)$ 를 구하고 6-input-Adder를 이용하여 덧셈을 수행한다. 그리고 그 결과를 T -레지스터에 저장한다. 또한 알고리즘1의 step 3을 비교 및 펠셈 연산을 수행하지 않기 위해서 알고리즘을 변형 [7]하였다. 이렇게 할 경우 원래 몽고메리 알고리즘의 결과가 달라지는데 이것은 전처리 및 후처리 과정을 통해 보정가능하다.

4. 실험 결과

본 논문에서는 각종 32비트 곱셈기와 몽고메리 곱셈기를 하드웨어 설계 언어인 VHDL로 설계하였다. 구현된 VHDL 코드는 Model Technology 사의 ModelSim5.2c를 가지고 시뮬레이션 되었으며, Xilinx 사의 foundation 3.1을 통해 합성되었다.

1. 곱셈기의 성능

[표 2]는 2장에서 설명한 어레이 곱셈기 3개와 수정된 부스 곱셈기 그리고 부분곱 곱셈기를 월리스트리를 이용하여 성능을 비교한 것이다. [표 2]에서 볼 수 있듯이 단일 클럭에서는 수정된 부스 곱셈기가 빠른 속도를 냄을 알 수 있다. 또한 면적을 고려했을 경우는 어레이 곱셈기가 효율적임을 알 수 있다.

	클럭수	속도(ns)	면적(게이트)
어레이 곱셈기(32 x 8)	4	59.35	4758
어레이 곱셈기(32 x 16)	2	73.19	7319
어레이 곱셈기(32 x 32)	1	102.58	12084
부분곱 곱셈기	1	42.14	18069
수정된 부스 곱셈기	1	39.97	14658

[표 2. 32비트 곱셈기의 속도와 면적]

2. 몽고메리 곱셈기의 성능

[표 3]는 위에서 언급한 32비트 곱셈기를 몽고메리 곱셈기에 적용시켜 속도와 면적을 측정한 것이다.

곱셈기의 종류	속도	면적
어레이 곱셈기(32 x 8)		
어레이 곱셈기(32 x 16)		
어레이 곱셈기(32 x 32)		
부분곱 곱셈기		
수정된 부스 곱셈기		

[표 3. 1024비트 몽고메리 곱셈기의 속도와 면적]

5. 결론

본 논문에서는 몽고메리 곱셈기를 32비트로 동작하도록 설계하였고 곱셈기의 선택에 따른 몽고메리 곱셈기의 성능을 비교 분석해 보았다. 몽고메리 곱셈기를 사용하여 모듈러 연산을 수행할 경우 지수의 길이에 의존해서 성능이 변할 것으로 예상되었다.

앞으로는 몽고메리 곱셈기를 파이프라인 하여 좀 더 빠른 환경에서 동작할 수 있도록 할 것이다.

【참고문헌】

- P. L. Montgomery, "Modular multiplication without trial division", Mathematics of Computation, Vol. No. 170, pp. 519~521, Apr. 1995.
- 김무섭, 최용재, 김호원, 정교일, "개선된 몽고메리 알고리즘을 이용한 저면적용 RSA 암호 회로 설계", 정보보호학회논문지, pp. 95~105, 2002, 10.
- C. S. Wallace, "A suggestion for a fast multiplier", IEEE Trans. Electrom. Comp., vol. EC-13, pp. 14~17, Feb, 1964.
- Behrooz Parhami, Computer Arithmetic Algorithms and hardware designs, 1999.
- A. D. Booth. "A signed binary multiplication technique", Q. J. Math., vol. 4. IV, pt.2, 1951.
- K. Hwang. Computer Arithmetic, Principles, Architecture, and Design. New York, NY: John Wiley & Sons, 1979.
- T. Blum and C. Parr. "Montgomery modular exponentiation on reconfigurable hardware," 14th IEEE Symposium on Computer Arithmetic, pp.70~77, 1999.