내장형 시스템을 위한 최적화된 RSA 암호화 프로세서 설계

준회원 허 석 원, 김 문 경, 정회원 이 용 석

Design of an Optimal RSA Crypto-processor for Embedded Systems

Seok-Won Heo, Moon-Gyung Kim *Associate Members*
Yong-Surk Lee *Regular Member*

요 약

본 논문에서는 RSA 암호화 알고리즘을 지원하기 위한 암호화 프로세서의 구조를 제안한다. 본 논문의 RSA 암호화 프로세서는 빅 몽고메리 알고리즘(FIOS)을 기반으로 제안되었으며, 다양한 비트 길이(128~2048 비트)를 지원한다. RSA 암호화 프로세서의 구조는 RSA 제어 신호 발생기, 빅 몽고메리 프로세서(가산기, 승산기)의 모듈로 구성된다. 빅 몽고메리 프로세서의 가산기와 승산기는 다양한 알고리즘을 이용하여 구현하였다. 내장형 시스템에 적합하게 설계하기 위하여 여러 가지 연산기를 합성한 결과 중에서 ARM 코프로세서와 연동할 수 있는 동작 주파수를 갖는 연산기 중에서 가장 작은 연산기를 선택하였다. RSA 암호화 프로세서는 Verilog-HDL을 이용하여 하향식 설계 방법으로 구현되었으며, C언어와 Cadence의 Verilog-XL을 이용하여 검증하였다. 검증된 모델은 하이닉스 0.25um CMOS standard cell 라이브러리를 이용하여 합성되었으며, 2.3V, 100°C 최악 조건에서 동작한다. 본 논문에서 제안한 RSA 암호화 프로세서는 약 51MHz의 주파수에서 동작하며, 게이트 수는 nand2 게이트 기준으로 약 36,639gates의 면적을 가진다.

ABSTRACT

This paper proposes a RSA crypto-processor for embedded systems. The architecture of the RSA crypto-processor should be used relying on Big Montgomery algorithm, and is supported by configurable bit size. The RSA crypto-processor includes a RSA control signal generator, an optimal Big Montgomery processor(adder, multiplier). We use diverse arithmetic unit(adder, multiplier) algorithm. After we compared the various results, we selected the optimal arithmetic unit which can be connected with ARM core-processor. The RSA crypto-processor was implemented with Verilog HDL with top-down methodology, and it was verified by C language and Cadence Verilog-XL. The verified models were synthesized with a Hynix 0.25um, CMOS standard cell library while using Synopsys Design Compiler. The RSA crypto-processor can operate at a clock speed of 51 MHz in this worst case conditions of 2.7V, 100°C and has about 36,639 gates.

Key Word : RSA, 암호화 프로세서, 빅 몽고메리, 내장형 시스템, 가산기, 승산기, HDL

I. 서 론

네트워크 기술의 발달로 현대 사회는 유무선 네트워크와 같은 거대한 공통매체로 정보를 공유하게 되었다. 필요에 따라 이러한 정보들은 암호화되어 보호되어야 하므로 개인의 정보를 인증해 줄 수 있는 스마트카드와 같은 정보 보호 기술이 반드시 필요하게 되었다[1].

*연세대학교 전기전자공학과 프로세서연구실(comace@dubiki.yonsei.ac.kr)
논문번호 : 030546, 접수일자 : 2003년 12월 11일
※ 본 연구는 하이닉스반도체의 "Flexible Cryptographic Engine" 프로젝트 지원으로 수행되었음

데이터의 보호를 위한 암호 알고리즘은 보통 구현의 용이성, 이식성 및 개발 비용의 저렴성 등으로 인하여 주로 소프트웨어로 구현한다. 그러나 소프트웨어로 구현하는 것은 컴퓨터, 통신기기 및 네트워크 등의 발전에 따르는 처리 속도에 적합하지 못하다. 따라서 이러한 처리 속도 보완과 더 높은 보안성을 제공하기 위해서 하드웨어적으로 암호화 프로세서가 필요하게 되었다.

따라서 본 논문에서는 내장형 시스템(Embedded system)에 응용할 수 있는 RSA 암호화 프로세서를 구현하였다. 구현한 암호화 프로세서는 ARM 마이크로프로세서의 코프로세서(co-processor) 형태로 동작이 된다. 그리고 ARM 마이크로프로세서에 존재하는 명령어를 사용하여 데이터 전달 및 암호화 명령어 수행을 통한 RSA 암호화 작업을 수행한다.

본 논문에서 구현한 RSA 암호화 프로세서는 빅 몽고메리 알고리즘을 기반으로 하여 RSA 제어 신호 발생기(RSA control signal generator)와 승산기(multiplier), 가산기(adder)로 구성되었다. RSA 암호화 프로세서를 최적화시키기 위하여, 여러 가지 형태의 승산기와 가산기를 설계하였다. 설계된 승산기와 가산기의 면적 및 지연시간을 비교하여 지연시간이 일정 시간보다 작은 연산기 중에서 면적이 가장 작은 최적화된 승산기와 가산기를 선택하였다.

본 논문에서는 제 2장에서 RSA 암호 알고리즘의 특징에 관하여 살펴보고, 제 3장에서 빅 몽고메리 알고리즘 및 RSA 제어 신호 발생기의 구조에 관하여 살펴본다. 제 4장과 제 5장에서는 가산기와 승산기의 여러 구조를 살펴보고, 구현된 연산기들의 성능 평가를 통하여 최적화된 연산기를 선택한다. 제 6장에서는 위에서 언급한 RSA 제어 신호 발생기, 승산기와 가산기를 통합한 RSA 암호화 프로세서를 구현하여 합성 결과를 평가하며, 제 7장에서 결론을 맺는다.

Ⅱ. RSA 암호 알고리즘

RSA 공개키 암호 시스템은 1977년 Rivest, Shamir와 Adleman에 의하여 개발되었다. RSA 암호 알고리즘은 약 200자리 십진 정수의 소인수분해의 어려움을 갖는 것을 기반으로 하여 보안성과 전자서명을 제공한다.

RSA 암호화 알고리즘을 이용하여 공개키와 비밀키를 생성하는 순서는 다음과 같다.

첫째, 개략적으로 비슷한 크기의 큰 임의의 소수 p, q를 생성하고 n과 ϕ를 계산한다.

$$n = pq \tag{1}$$

$$\phi = (p-1)(q-1) \tag{2}$$

둘째로 식 (3)을 만족하는 임의의 정수 e를 선택하여 공개 자물쇠로 설정한다.

$$\gcd(e, \phi) = 1, \ (1 < e < \phi) \tag{3}$$

셋째로, 식 (4)의 Extended Euclidean algorithm을 사용하여 특별한 정수 d를 계산한다.

$$ed \equiv 1 (\bmod \phi), \ (1 < d < \phi) \tag{4}$$

마지막으로 선택한 e 값과 계산한 d 값을 이용하여, 공개키 (n, e), 비밀키 (p, q, d)를 생성한다.

RSA 암호 시스템을 다른 공개키 암호 시스템과 비교할 때 장점은 다음과 같다.

첫째로 비교적 구현이 간단한 알고리즘이다.

둘째로 1978년 이후 20년 이상 검증된 암호 시스템이다.

셋째로 RSA 암호 시스템은 현재 상용화된 암호 시스템 중에서 가장 큰 시장 점유율을 가지며, 마지막으로 타원 곡선 암호 시스템(Elliptic Curve Crypto-system)과 달리 system parameter 생성이 불필요하다.

Ⅲ. RSA 제어 신호 발생기

RSA 암호 알고리즘과 같은 공개키 암호 알고리즘에서 가장 많이 사용되는 연산인 modular multiplication 연산을 구현하는 방법에 대해서 알아보자. Modular multiplication을 구현하는 알고리즘은 매우 많지만, 그 중 가장 구현하기 쉽고 가장 많이 사용되는 알고리즘이 바로 몽고메리 알고리즘이다. 본 논문에서는 메모리 접근(Memory access) 시간을 줄이기 위하여 word 단위로 연산을 수행하는 빅 몽고메리 알고리즘을 사용하였다.

1. 빅 몽고메리 알고리즘(Big Montgomery)

일반적으로 큰 수에 대한 연산은 그 수를 word 단위로 쪼개어 수행한다. 만약 w가 컴퓨터의 word

크기라고 한다면 그 수는 radix W = 2^w로 표현되는 수들의 연속으로 생각할 수 있다. 만약 그 수가 radix W로 표현되는 s개의 word가 필요하다면 r=2^{sw}로 r값을 정할 수 있다.

빅 몽고메리 알고리즘은 크게 5가지로 나누어 볼 수 있다. 5가지 빅 몽고메리 알고리즘의 연산 및 메모리 접근 횟수는 표 1과 같다.

표 1. 빅 몽고메리 비교
Table 1. Comparison of Big Montgomery Algorithms

방법	Multiplications	Additions	Reads	Writes	Space
SOS	$2s^2+s$	$4s^2+2s+1$	$5s^2+5s$	$2s^2+4s$	2s+2
CIOS	$2s^2+s$	$4s^2+3s+1$	$5s^2+4s$	$2s^2+4s$	s+3
FIOS	$2s^2+s$	$4s^2+s+1$	$5s^2+3s$	$2s^2+2s$	s+3
FIPS	$2s^2+s$	$4s^2+s+1$	$7s^2+7s$	$3s^2+7s$	s+3
CIHS	$2s^2+s$	$4s^2+4s+1$	$5s^2+7s$	$2s^2+7s$	s+3

이 표의 결과는 Koç[2]가 발표한 내용과는 약간 차이가 있는데, 이는 알고리즘들에 대하여 다시 최적화를 수행하였고 실제 하드웨어에서의 구현을 고려하여 연산을 축소시켰기 때문에 일어난 결과이다. 표를 비교하여 보면 FIOS 알고리즘이 연산 횟수나 메모리 제어가 비교적 적은 것으로 나타났다. 따라서 본 논문에서는 연산 횟수와 메모리 제어수가 적어 전체 실행시간의 감소를 얻을 수 있는 FIOS 알고리즘을 선택하였다.

표 2는 수정된 빅 몽고메리 FIOS 알고리즘이다.

2. RSA 제어 신호 발생기의 구현

RSA 제어 신호 발생기에서 제어 신호를 발생시켜서 Smart MAC(Multiply and Accumulator)으로 보낸다. Smart MAC에서는 제어 신호를 받아서 승산, 가산, MAC 연산을 수행한 후에 연산 결과를 주소 선택기(Address Selector)에 의하여 지정된 레지스터 파일에 쓰게 된다. Smart MAC은 최적화된 승산기와 가산기를 내장하여 구현하였다.

그림 1은 RSA 제어 신호 발생기와 빅 몽고메리 프로세서의 구조를 보여준다.

표 2. 수정된 빅 몽고메리 알고리즘(FIOS)
Table 2. Modified Big Montgomery Algorithm(FIOS)

Product & Reduction & Division	(C,S) := a[0] * b[0]
	t[1] := C
	m := S * n'[0] mod W
	(C,S) := S + m * n[0]
	for j = 1 to s - 1
	  (C,S) := t[j] + a[j] * b[0] + C
	  t[j + 1] := C
	  (C,S) := S + m * n[j]
	  t[j - 1] := S
	(C,S) := t[s] + C
	t[s - 1] := S
	t[s] := C

	for i = 1 to s - 1
	  (C,S) := t[0] + a[0] * b[i]
	  (C',S') := t[1] + C
	  t[1] := S'
	  m := S * n'[0] mod W
	  (C,S) := S + m * n[0]
	  for j = 1 to s - 1
	  (C,S) := t[j] + a[j] * b[i] + C
	  (C',S') := t[j + 1] + C + C'
	  t[j + 1] := S'
	  (C,S) := S + m * n[j]
	  t[j - 1] := S
	  (C,S) := t[s] + C
	  t[s - 1] := S
	  (C,S) := C + C'
	  t[s] := S

Compensation	if (t[0]&1) t = t + n t = t >> 1
	u = t - n if u > 0, then return u[0], ... , u[s - 1] else return t[0], ... , t[s - 1]

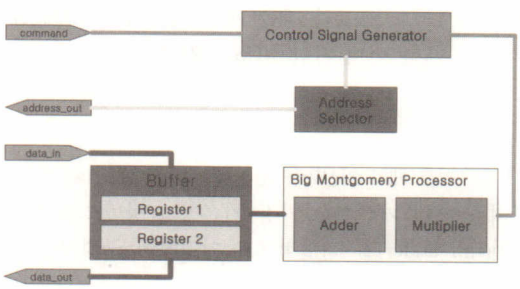


그림 1. RSA 제어 신호 발생기와 빅 몽고메리 프로세서 구조
Fig. 1. Structure of RSA Control Signal Generator and Big Montgomery Processor

Ⅳ. 최적화된 가산기(Adder)

1. 가산기의 구조

가산기는 직렬 구조(serial), 트리 구조(tree), 하이브리드 구조(hybrid)로 크게 3가지로 분류할 수 있다.

직렬 구조 가산기는 리플 캐리 가산기와 같이 최악 조건 지연 시간이 O(n)일 경우에 면적이 O(n)이 되는 특징을 가진다. 즉, 최악 조건 지연 시간과 면적이 정비례하는 특징을 가진다.

트리 구조 가산기는 캐리 예측 가산기와 같이 최악 조건 지연 시간이 크게 감소되는 고성능 가산기이다. 이 구조의 가산기는 O(logn)의 지연 시간을 가지는 장점을 가지고 있지만, O(n logn)의 큰 면적을 갖는 단점을 가지고 있다.

하이브리드 구조의 가산기는 직렬 구조와 트리 구조의 중간 성능을 갖는다. 하이브리드 구조는 n비트 블락으로 나누어서 병렬적으로 연산한다. 하이브리드 구조의 가산기는 면적이 O(n)이지만, 지연 시간은 $O(\sqrt{n})$이 된다.[3] 면적은 직렬 가산기와 동일하지만, 지연시간이 더 빠른 장점을 가진다.

본 논문에서 아래와 같은 7가지 구조의 가산기를 제안하였다.

- 직렬 구조의 리플 캐리 가산기(RCA)
- 트리 구조의 캐리 예측 가산기(CLA)
- 하이브리드 구조의 캐리 예측 가산기를 기반으로 하는 리플 캐리 가산기(RCA_CLA)
- 하이브리드 구조의 리플 캐리 가산기를 기반으로 하는 캐리 예측 가산기(CLA_RCA)
- 하이브리드 구조의 캐리 예측 가산기를 기반으로 하는 캐리 선택 가산기(CSA_CLA)
- 하이브리드 구조의 리플 캐리 가산기를 기반으로 하는 캐리 선택 가산기(CSA_RCA)

1) 직렬 구조 가산기

리플 캐리 가산기(Ripple Carry Adder)

리플 캐리 가산기는 가장 기본적인 방법으로 전가산기(full adder)를 직렬적으로 연결하여 구성하는 것이다[4].

2) 트리 구조 가산기

캐리 예측 가산기(Carry Look-ahead Adder)

캐리 예측 가산기는 하위 비트의 캐리 출력을 기다릴 필요 없이 입력이 결정되면 빠른 시간에 연산의 결과를 얻을 수 있다는 장점을 가지고 있다. 그러나 입력의 비트가 클 경우 fan-in이 많아지고 면적이 커지는 단점을 가지고 있다. 이러한 단점을 극복하기 위하여, 가산기를 여러 그룹으로 나눈 후에 그 그룹이 더 작은 그룹을 갖는 멀티 레벨 구조를 갖는 가산기가 많이 이용되고 있다[5][6]

3) 하이브리드 구조 가산기

캐리 선택 가산기(Carry Select Adder)

캐리 선택 가산기는 입력되는 캐리의 값이 '0' 또는 '1'의 값을 가지므로 이 두 가지 경우에 대해 덧셈을 미리 수행한다. 그리고 최하위 블락의 캐리 출력이 결정되면, 이 결과를 이용하여 미리 계산한 결과들 중의 하나를 선택한다. 캐리 선택 가산기는 병렬적으로 연산을 하기 때문에, 리플 캐리 가산기보다 약 2배의 면적을 사용하지만, 약 4배의 속도를 향상시킬 수 있다[7]. 캐리 선택 가산기는 멀티플렉서를 통과하는 시간을 고려하여 마지막 그룹으로 갈수록 많은 비트를 할당할 수 있다. 본 논문에서는 그룹별 비트가 다를 경우 너무 많은 경우의 수가 발생할 수 있으므로 각 그룹별 비트수를 동일하게 처리하였다.

하이브리드 구조 가산기 구현

하이브리드 구조 가산기는 소그룹의 캐리 출력을 캐리 전달 유닛으로 연결하여 연산한 후 캐리 전달 유닛의 출력을 다음 소그룹의 캐리 입력으로 연결하는 구조이다. 소그룹은 리플 캐리 가산기, 캐리 예측 가산기의 두 가지 구조이며, 캐리 전달 유닛은 리플 캐리 가산기, 캐리 예측 가산기, 캐리 선택 가산기 등 세 가지 구조를 가지고 있다. 소그룹은 4, 8, 16 비트로 나누어서 연산을 할 수 있다. 그림 2는 하이브리드 구조의 4개 소그룹으로 나누어진 16비트 가산기의 구조이다.

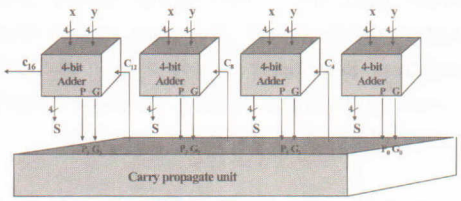


그림 2. 16비트 하이브리드 구조 가산기
Fig. 2. Hybrid Adder Structure

2. 가산기의 합성 결과

가산기를 두가지 방법으로 합성하였다. 첫 번째 방법은 하위 모듈들을 group으로 합성을 하였고, 두 번째 방법은 ungroup으로 합성을 하였다. 일반적으로 ungroup으로 합성할 경우, 지연 시간은 감소되고 면적은 증가하는 현상이 발생한다. 그러나 본 논문에서 제안한 가산기는 지연 시간에만 최적화되어서는 안되므로 group과 ungroup을 이용하여 합성하였다. 내장형 시스템은 일정한 클락 주파수 이상 동작하는 것도 중요하지만, 면적을 최소화시키는 것이 더욱 중요하다.

표 3과 4는 가산기의 지연시간과 면적을 비교한 것이다.

3. 최적화된 가산기의 성능 평가

대부분의 내장형 시스템에서 사용하는 ARM 마이크로프로세서는 40MHz의 동작 주파수를 가진다. 따라서 본 논문에서 제안한 RSA 암호화 프로세서도 같은 속도의 동작 주파수를 가져야 한다. ARM 마이크로프로세서가 한 번 실행될 때, 빅 몽고메리 프로세서 안의 가산기는 네 번 실행이 된다. 그러므로 최적화된 가산기는 160MHz 이상의 동작 주파수 즉, 지연시간이 6.25ns 이하로 동작을 수행해야 한다. 동작 주파수의 조건을 갖춘 가산기 중에서 제조 원가를 줄이기 위하여 면적이 가장 작은 가산기를 선택해야 한다.

그 결과 RSA 암호화 프로세서 안에서 사용될 최적화된 가산기는 64, 128비트를 기준으로 할 때, ungroup으로 합성된 8비트 캐리 예측 가산기를 기반으로 하는 리플 캐리 가산기(RCA_CLA)이며, 256비트를 기준으로 할 때, ungroup으로 합성된 16비트 캐리 예측 가산기를 기반으로 하는 리플 캐리 가산기(RCA_CLA)이다. 그러나 본 논문에서 제안한 RSA 암호화 프로세서는 입력 비트가 최대 2048비트이므로 연산 처리 비트가 클수록 전체 실행 시간이 줄어드는 장점을 가진다.

따라서 본 논문에서는 ungroup으로 합성된 16비트 캐리 예측 가산기를 기반으로 하는 256비트 리플 캐리 가산기를 선택하였다. 이 최적화된 가산기는 200.4MHz의 속도로 동작하며, nand2 게이트 기준으로 약 4331의 면적을 갖는다.

표 3. 가산기의 지연 시간과 면적 비교 (group으로 합성)
Table 3. Comparison of Adders(Group)

adder	group size	64 (bit)		128 (bit)		256 (bit)	
		time	area	time	area	time	area
RCA	·	13.38	811	26.69	1565	53.27	3208
CLA	·	2.26	1431	2.75	2437	3.17	4989
HDL	·	2.18	1382	2.37	2902	2.82	5400
RCA_CLA	4	6.85	1012	13.01	2153	27.80	4237
	8	4.12	986	7.29	2054	14.19	4145
	16	3.03	1108	4.55	2158	7.62	4174
CLA_RCA	4	12.29	856	24.00	1784	28.30	3256
	8	12.91	821	25.81	1680	51.19	3299
	16	12.65	822	26.06	1620	52.65	3202
CSA_CLA	4	4.78	2141	9.47	4507	23.79	9337
	8	2.97	2059	6.67	4430	12.43	9047
	16	2.45	2150	4.21	4542	7.09	9484
CSA_RCA	4	4.79	1848	12.26	3592	21.43	7340
	8	4.53	1651	5.80	3399	10.74	6720
	16	4.41	1765	5.50	3582	7.57	6825

표 4. 가산기의 지연 시간과 면적 비교 (ungroup으로 합성)
Table 4. Comparison of Adders(Unroup)

adder	group size	64 (bit)		128 (bit)		256 (bit)	
		time	area	time	area	time	area
RCA	·	2.16	1711	2.69	3268	3.74	6438
CLA	·	2.37	1342	2.57	2515	3.37	5192
HDL	·	2.18	1382	2.37	2902	2.82	5400
RCA_CLA	4	2.29	1596	2.73	3029	3.49	5531
	8	2.71	1290	4.26	2140	6.62	4404
	16	2.42	1420	3.32	2574	4.99	4331
CLA_RCA	4	2.12	1838	2.79	3329	2.92	5804
	8	2.15	1604	2.95	3149	3.54	6020
	16	2.67	1504	2.84	3268	3.32	6648
CSA_CLA	4	2.24	1256	2.74	2351	3.09	4779
	8	2.30	1374	2.74	2448	3.17	5207
	16	2.90	1568	3.18	2956	3.27	5935
CSA_RCA	4	4.05	1635	9.85	3063	22.52	5607
	8	2.84	1832	4.15	3343	8.72	6486
	16	3.09	1879	3.75	3694	4.96	4534

RCA 리플 캐리 가산기
CLA 캐리 예측 가산기
RCA_CLA 캐리 예측 가산기 기반의 리플 캐리 가산기
CLA_RCA 리플 캐리 가산기 기반의 캐리 예측 가산기
CSA_CLA 캐리 예측 가산기 기반의 캐리 선택 가산기
CSA_RCA 리플 캐리 가산기 기반의 캐리 선택 가산기

V. 최적화된 승산기(Multiplier)

1. 승산기의 구조

RSA 암호화 프로세서는 최대 2048비트의 연산을 수행해야 하므로 많은 비트의 연산을 처리할 수 있으면서 일정한 속도 이상의 저면적 승산기가 필요하다. 본 논문에서는 승산기를 다음과 같이 6가지 종류로 제안하였다.

- Radix-4 수정 Booth 알고리즘과 3 : 2 캐리 저장 가산기를 이용한 32 * 32비트 승산기
- Radix-8 수정 Booth 알고리즘과 4 : 2 캐리 저장 가산기를 이용한 32 * 32비트 승산기
- Verilog HDL의 곱셈 연산자를 이용한 32 * 32 비트 승산기
- Radix-4 수정 Booth 알고리즘과 3:2 캐리 저장 가산기를 이용한 62 * 32비트 승산기
- Verilog HDL의 곱셈 연산자를 이용한 64 * 32 비트 승산기
- Verilog HDL의 곱셈 연산자를 이용한 64 * 64 비트 승산기

Radix-4 수정 Booth 알고리즘을 사용한 32 * 32 비트 승산기는 16개의 부분곱이 필요하다. 그러나 Radix-8 수정 Booth 알고리즘을 사용한 32 * 32비트 승산기는 11개의 곱이 필요하므로 Radix-4 수정 Booth 알고리즘을 사용한 승산기보다 지연시간을 적게 가질 수 있다. 그러나 Radix-8 수정 Booth 알고리즘을 사용한 승산기는 booth encoder에서 3X를 쉬프트와 덧셈을 이용하여 연산해야 하므로 많은 지연시간이 소모된다.

웰레스 트리에서 4 : 2 캐리 저장 가산기를 이용한 승산기는 3 : 2 캐리 저장 가산기를 이용한 승산기보다 규칙적인 형태를 갖는다. 따라서 파이프라인의 효율을 높여주고 면적을 줄여줄 수 있다.[8]

Booth 알고리즘을 사용한 승산기는 웰레스 트리에서 면적을 줄이기 위하여 부호 확장(Sign Extension) 대신 부호 생성 방법(Sign Generate Method)[9]를 이용하였다. 그리고 최종 가산기는 자리올림 전파시간(carry propagate delay)를 감소시키기 위하여 HDL의 연산자를 이용하여 구현하였다. HDL 연산자는 본 논문 제 4장에서 가장 빠른 가산기로 결과가 제시되어 있다.

표 5는 Radix-4 수정 Booth 알고리즘의 부분곱과 동작을 설명한 것이다. 그림 3은 Booth Encoder와 웰러스 트리 및 최종 가산기를 이용한 곱셈기의 구조이다.

1) 승수와 피승수의 부호 확장

본 논문에서 제안한 승산기는 승수와 피승수를 입력받아 1비트 확장하여 연산한다. 부호 확장을 하면 signed 곱셈이 되고, '0'을 채워 넣으면 입력 값이 항상 양수로 인식되므로 unsigned 곱셈이 된다.[10]

표 5. 수정 Booth 알고리즘의 승수에 따른 동작(Radix-4)
Table 5. Operations by Partial Products of Modified Booth Algorithm(Radix-4)

Y_{i+1} Y_i Y_{i-1}	부분곱	동 작 설 명
0 0 0	0	연속된 '0' no operation
0 0 1	+X	string의 끝 X를 덧셈
0 1 0	+X	X를 덧셈
0 1 1	+2X	string의 끝 2X를 덧셈
1 0 0	-2X	string의 시작 -2X를 덧셈
1 0 1	-X	string의 끝과 다른 string의 시작 -X를 덧셈
1 1 0	-X	string의 시작 -X를 덧셈
1 1 1	0	연속된 '1', string의 중간 no operation

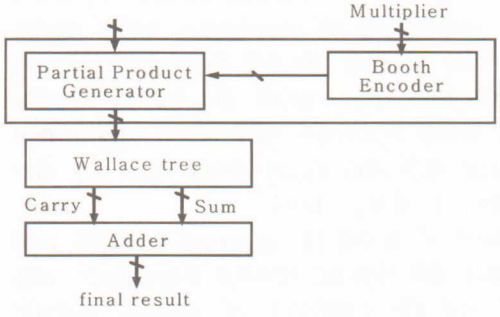


그림 3. 승산기 구조
Fig. 3. Multiplier Structure

피승수를 확장을 하지 않으면 연산 오류가 일어날 가능성이 있다. 이진수 표현에서 2의 보수법(2's complement)에서는 음수가 양수보다 하나 더 많다.

즉, 2의 보수법 N비트로 표현할 수 있는 범위는 식 (5)와 같다.

$$-2^{N-1} \le A \le 2^{N-1}-1 \qquad (5)$$

만약 피승수에 최소값, 즉 -2^{N-1}이 입력된 경우 승산기가 -X나 -2X와 같은 동작을 해야 한다면 부호 생성 알고리즘을 사용한다 하더라도, 피승수를 N+1비트로 확장을 하지 않고서는 올바른 결과를 얻을 수 없다.[11]

2) 웰러스 트리의 구조

월레스 트리에서는 부분곱 간의 부호 확장(sign extension)을 미리 삽입한 후 연산을 수행하게 된다. 부호 확장하는 방법은 부호 생성 알고리즘이다. 부분곱들은 기준 자리수에 따라 두 비트씩 왼쪽 쉬프트 되어 더해지게 되는데, 이때 MSB에서 부호 확장이 일어나야 한다. 모든 부분곱의 MSB에서 부호 확장을 하면, 확장된 부분을 모두 더해야 하므로, 하드웨어로 구성을 하는 것에 있어서 면적이 커진다. 따라서 각 부분곱에 다음과 같은 과정을 거치면 작은 크기의 면적으로 부호 확장을 대신할 수 있다.

1. 각 부분곱의 부호비트를 반전하여 MSB에 추가한다.
2. 각 부분곱의 MSB에 '1'을 추가한다.
3. 곱셈 결과에 2N을 더한다.(N은 피승수의 비트수)

그림 4는 8비트 곱셈에서의 부호 생성 알고리즘이다.

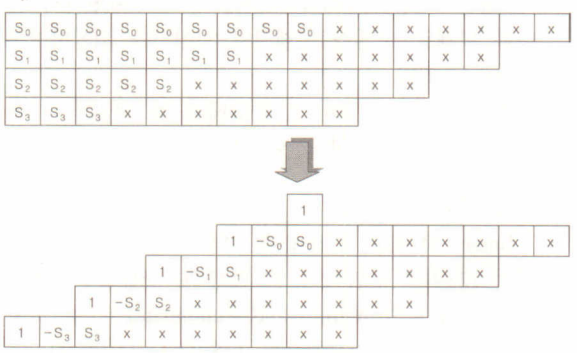


그림 4. 부호 생성 알고리즘
Fig. 4. Sign Generation Algorithm

월레스 트리는 자리올림 전파를 가산기의 덧셈 지연시간과 겹치도록 함으로써 덧셈에 걸리는 시간을 줄이는 방법이다.

월레스 트리에서는 3 : 2 캐리 저장 가산기(Carry Save Adder)와 4 : 2 캐리 저장 가산기를 많이 사용한다. 4 : 2 캐리 저장 가산기는 3 : 2 캐리 저장 가산기보다 규칙적인 형태를 갖는다. 따라서 면적을 감소시킬 수 있다.

2. 승산기의 합성 결과

합성한 결과를 보면 승산기의 크기에 비례하여 면적이 증가하는 특징을 보였다. 그러나 월레스 트리를 이용한 승산기는 승산기의 크기가 증가해도, stage의 개수 차이는 거의 비슷하므로, 지연시간은 거의 증가(최대 5.80% 증가)하지 않는다.

32 * 32비트 승산기에서 Radix-4 수정 Booth 알고리즘을 이용한 승산기는 Radix-8 수정 Booth 알고리즘을 이용한 승산기보다 지연시간이 빠르게 측정(16.77% 지연시간 감소)되었다. 이 결과는 Radix-8 수정 Booth 알고리즘을 이용한 승산기가 +3X, -3X를 연산을 수행하기 때문이다. +3X, -3X는 쉬프트후 연산을 처리해주어야 하므로 지연시간이 오래 걸린다.

4 : 2 캐리 저장 가산기를 이용한 승산기는 3 : 2 캐리 저장 가산기를 이용한 승산기보다 면적이 작게 측정(2.55% 면적 감소)되었다. 4 : 2 캐리 저장 가산기는 규칙적인 형태를 갖기 때문에 합성시 면적을 감소시키는 것이 용이하다.

Booth 알고리즘과 월레스 트리를 이용하지 않고, '*' operator를 이용한 승산기는 Booth 알고리즘과 월레스 트리를 이용한 승산기보다 작은 면적(최대 33.83% 감소)을 보였지만 지연시간 이 크게 증가(최대 210% 증가) 하였다.

표 6은 승산기 종류에 따른 지연시간과 면적이다.

표 6. 구현된 승산기의 비교
Table 6. Comparison of Multipliers

multiplier	area (nand2)	time (ns)
32 * 32 multiplier (Radix 4, (3, 2) CSA)	11250	6.55
32 * 32 multiplier (Radix 8, (4, 2) CSA)	10963	7.87
32 * 32 multiplier ("*" operator)	7746	20.65
64 * 32 multiplier (Radix 4, (3, 2) CSA)	21279	6.93
64 * 32 multiplier ("*" operator)	14621	21.49
64 * 64 multiplier ("*" operator)	27914	40.78

3. 최적화된 가산기의 성능 평가

RSA 암호화 프로세서에 내장되는 승산기는 가산기와 연동되어서 동작되어야 한다. RSA 암호화 프로세서가 연산할 입력 비트가 최대 2048비트이므로 가산기가 큰 것이 연산에 유리하다. 그러나 승산기는 가산기보다 면적이 매우 크기 때문에 가산기와 같은 크기로 설계하는 것은 어렵다. 특히 본 논문에서 제시한 RSA 암호화 프로세서는 내장형 시스템에 적합하도록 구현될 것이므로, 면적의 크기가 작아야 하는 것이 가장 중요하다.

승산기와 가산기의 크기가 다르기 때문에 곱셈 누적 연산을 할 경우에는 곱셈이 몇 번의 연산을 수행한 후에 덧셈을 수행하는 구조를 갖는다. 따라서 곱셈의 지연시간이 느리다면, 곱셈 연산 뿐만 아니라 곱셈 누적 연산까지 느려지게 된다. 따라서 웰러스 트리를 사용하지 않은 승산기, 즉 '*' operator를 사용한 승산기는 지연 시간이 너무 크므로 사용할 수 없다.

입력 비트가 최대 2048비트 이므로 승산기의 크기가 클수록 곱셈 연산이나 곱셈 누적 연산의 클락 사이클 수가 적어지게 된다. 64 * 32비트 승산기가 32 * 32비트 승산기보다 더 큰 면적을 갖게 되지만 (89.15% 증가), 64 * 32비트 승산기를 사용하면 32 * 32비트 승산기보다 절반의 클럭 사이클 수를 갖게 된다.

따라서 본 논문에서는 면적과 지연시간을 최적화하기 위하여 Radix-4 수정 Booth 알고리즘과 3 : 2 캐리 저장 가산기를 이용한 웰레스 트리로 구성된 승산기를 선택하기로 한다. 이 최적화된 승산기는 144.3 MHz의 속도로 동작하며, nand2 게이트 기준으로 약 21279의 면적을 갖는다.

Ⅵ. RSA 암호화 프로세서의 구현

1. RSA 암호화 프로세서의 구조

빅 몽고메리 알고리즘의 RSA 암호화 연산을 수행하면서, 128, 256비트 덧셈, 128비트 곱셈, 128비트 곱셈 누적 연산이 필요하다. 곱셈 누적 연산기(Multiply Accumulate Unit)를 설계하면, 덧셈, 혹은 곱셈 연산만 수행할 경우에 다른 연산을 수행하지 않으면 된다. 그러나 덧셈과 곱셈 연산을 모두 수행하지 않음에도, 두 연산의 기능을 모두 가지고 있으므로, 면적이 상대적으로 크게 된다. 따라서 본 논문에서는 RSA 암호화 프로세서의 면적을 최소화시키기 위하여, 승산기와 가산기를 별도로 설계하였다. 그림 5는 코어프로세서(core-processor)와 암호화 코프로세서(co-processor)의 인터페이스를 보여준다. 코어프로세서와 암호화 코프로세서는 AMBA 버스를 통하여 연결된다.

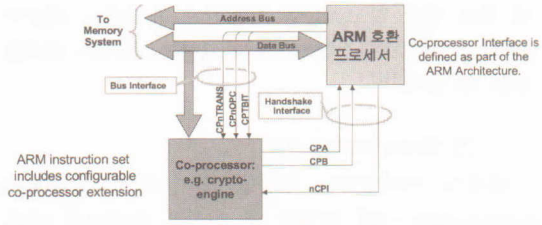


그림 5. ARM 코어프로세서와 RSA 코프로세서의 인터페이스
Fig. 5. Interface between ARM Core-processor and RSA Co-processor

1) 제어기

제어기는 Smart MAC에 해당 단계에 필요한 제어 신호를 보내는 것과 메모리를 제어하여 필요한 값들을 읽어오도록 하는 기능을 가지게 하였다. 실제 연산에 있어서는 Smart MAC을 이용하여 해당 RSA 키 스페이스를 위한 덧셈, 뺄셈, 쉬프트 연산 및 빅 몽고메리 알고리즘을 이용한 모듈라 곱셈을 지원하도록 전체 수행 단계를 구분지어 제어 신호를 공급하며, 이에 필요한 레지스터 파일 제어를 위하여 해당 주소와 읽기 / 쓰기 신호를 생성하여 공급한다. 구현된 Smart MAC이나 레지스터 파일의 속도가 응용 분야에 맞는 구조와 속도로 변형될 수 있기 때문에 연산 / 읽기 / 쓰기의 수행 완료를 확인하여 그 다음 단계로 수행을 진행시킬 것인지를 결정하도록 구현되어 있다.

표 7은 제어기에서 빅 몽고메리 알고리즘을 수행하기 위해 각 단계별로 생성하는 제어신호이다.

2) Smart MAC

Smart MAC은 128비트의 워드 단위를 기본으로 하여 128비트 단위의 곱셈, 덧셈 및 256비트 단위의 승산 누적 연산(multiply-and-accumulate)을 지원하도록 설계되었다. 기본적으로 128비트 크기의 레지스터 6개를 내장하며, 이 레지스터를 이용하여 필요한 연산을 수행한 후 저장하도록 하고 있다.

가산기와 승산기를 이용하여, 연산에 따라 입력 값이나 레지스터 값을 이용하여 연산을 수행하도록 하고 있다.

Smart MAC 연산을 위하여 64 * 32비트 승산기c

는 8번의 연산이 필요하며, 128비트의 결과에는 6번의 연산이 필요하다. 그러나 32 * 32비트 승산기와 256비트 가산기를 이용할 경우, 256비트 결과에는 16번의 연산이 필요하며, 128비트의 결과에는 10번의 연산이 필요하다.

표 8은 Smart MAC에서 지원하는 연산들을 나열한 것이다.

표 8. Smart MAC에서 지원하는 연산
Table 8. Operations of Smart MAC

smart_ command	operation
00_001	(REG1, REG0) = (REG3) + (REG1)
00_010	(REG1, REG0) = (REG1, REG0) + (RIN0)
00_011	(REG1, REG0) = (RIN0) + (REG1)
00_100	(REG3, REG2) = (RIN0) + (REG3)
00_101	(REG3, REG2) = (REG3, REG2) + (REG1)
01_000	(REG5) = (REG0) * (REG4)
01_001	(REG1, REG0) = ((REG0) + (REG5) * (RIN1))
01_010	(REG1, REG0) = ((REG1) + (RIN0) * (RIN1))
01_011	(REG1, REG0) = ((REG0) + (RIN0) * (RIN1))
01_100	(REG1, REG0) = ((RIN0) * (RIN1))
10_000	(REG0, carry) = (carry, RIN0)
10_001	(carry, REG0) = (RIN0) + (RIN1) + carry
10_010	(carry, REG0) = (RIN0) + (~(RIN1)) + carry
10_100	set output REG0
10_101	set output REG1
10_110	set output REG2
11_000	(REG0) = (RIN0)
11_001	(REG1) = (RIN1)
11_010	(REG2) = (RIN0)
11_011	(REG3) = (RIN1)
11_100	(REG4) = (RIN0)
11_110	carry = 0
11_111	carry = 1

2. RSA 암호화 프로세서의 합성 결과

제안된 RSA 암호화 프로세서는 Verilog-HDL을 이용하여 하향식 설계 방식(top-down methodology)으로 설계되었다. Cadence의 Verilog-XL을 이용하여 테스트 벡터 100,000개로 검증하였다. 하이닉스 0.25um CMOS standard cell 라이브러리를 이용하여 합성을 하였으며, 2.3(V), 100(°C) 최악 조건에서 동작한다. 면적은 nand2 gate 기준으로 report하였다.

표 9는 RSA 제어 신호 발생기와 승산기, 가산기를 포함하는 빅 몽고메리 프로세서가 내장된 RSA 암호화 프로세서의 합성 결과이다

표 9. RSA 암호화 프로세서의 합성 결과
Table 9. Synthesis Result of RSA Crypto-Processor

	RSA 암호화 프로세서
면 적	36,639 gates
지연 시간	19.49 ns
동작 주파수	51.3 Mhz
전력 소모)	17.63 mW

3. RSA 암호화 프로세서의 성능 평가

표 10은 제안된 RSA 암호화 프로세서와 다른 RSA 암호화 프로세서를 비교한 것이다. 본 논문에서 제안한 RSA 암호화 프로세서는 다른 RSA 암호화 프로세서보다 더 빠른 동작 주파수를 가지며, 상용화된 RSA 암호화 프로세서보다 면적이 작게 측정되었다. 빅 몽고메리 알고리즘의 FIOS를 사용하여 연산과 메모리 제어의 횟수를 줄인 것이 동작 주파수가 빨라진 원인이라고 생각된다.

그리고 본 논문에서 제안한 RSA 암호화 프로세서는 다른 RSA 암호화 프로세서보다 다양한 비트 길이를 지원한다. Fudan Univ.의 RSA 암호화 프로세서보다 다양한 비트 길이를 지원하므로, 제어 부분의 면적 증가에 따라서 면적이 증가한 것으로 생각된다.

본 논문에서는 내장형 시스템에 응용하기 위하여, 지연 시간과 면적 한 가지 요소에만 최적화를 한 것이 아니라, 지연 시간과 면적 모두 만족할 수 있는 최적화된 RSA 암호화 프로세서를 제안하였다.

표 10. RSA 암호화 프로세서의 비교
Table 10. Comparison of RSA Crypto-processors

저자	Commercial [12]	Fudan Univ. [13]	Ours
면적	55(K)	14(K)	38(K)
비트 길이	1024 비트	configurable (512~1024)	configurable (128~2048)
동작 주파수	5(Mhz)	40(Mhz)	51(Mhz)
공정	0.6(um)	0.5(um)	0.25(um)
알고리즘	몽고메리 알고리즘	Yang's 수정 몽고메리 알고리즘[14]	빅 몽고메리 알고리즘 (FIOS)

Ⅳ. 결 론

본 논문에서는 내장형 시스템에 응용하기 위한 RSA 암호화 프로세서를 구현하였다. 다른 RSA 암호화 프로세서와 다르게 일정한 지연 시간 안에 포함되는 연산기 중에서 최소한 면적을 갖는 연산기를 선택하여 구현을 하였다.

기존의 상용화된 RSA 암호화 프로세서는 비트 길이가 고정되거나, 면적은 작지만 지연 시간이 느린 단점을 가지고 있다. 그러나 본 논문에서는 기존의 RSA 암호화 프로세서를 보완하여, 다양한 비트 길이를 지원하면서 지연 시간이 빠른 RSA 암호화 프로세서를 제안하였다.

RSA 암호화 프로세서의 포함되는 승산기와 가산기를 여러 가지 알고리즘을 사용하여 합성된 결과를 기준으로 선택하였다. 다양한 형태로 설계한 승산기와 가산기는 합성 결과를 이용하여 RSA 암호화 프로세서 외에 다른 응응 분야에도 이용할 수 있을 것이다.

제안된 RSA 암호화 프로세서가 모바일 환경에 적용되기 위해서는 동작을 보장하는 지연시간 내에서 면적이 중요한 요소가 된다. 따라서 ARM 코어 프로세서를 포함한 전체 시스템의 동작 속도를 40Mhz로 예상하므로, 51MHz의 동작 주파수는 전체 시스템의 동작속도에 영향을 줄 만큼 임계 경로 지연시간이 되지 못한다.

본 논문에서 제안한 RSA 암호화 프로세서 외에 다음과 같은 추가 연구가 필요할 것이다. 본 논문에서는 빅 몽고메리 FIOS 알고리즘으로 구현을 하였다. 그러나 Koç[2]가 발표한 다섯 가지 빅 몽고메리 알고리즘에는 서로간의 장단점을 가지므로, 구현을 하지 않은 네 가지 빅 몽고메리 알고리즘을 이용하여 구현을 해야 할 것이다. 나머지 알고리즘을 구현하여 결과를 비교해야 한다. 또한 승산기와 가산기도 다른 알고리즘을 이용하여 좀 더 최적화를 할 필요가 있다.

본 논문에서 제안된 RSA 암호화 프로세서는 다양한 모바일 환경에서 응용될 수 있을 것이다.

참 고 문 헌

[1] 김철, 암호학의 이해, 영풍문고, 1996

[2] C. K. Koç, T. Acar and S. Burton, K. Jr, "Analyzing and Comparing Montgomery Multiplication Algorithm", *IEEE Micro, vol. 16, no. 3*, page(s) 26 ~ 33. June 1996.

[3] J. L. Hennessy and D.A. Patterson, *"Computer Architecture : A Quantitative Approach, third edition"*, Morgan Kaufmann Publishers, CA, 2003.

[4] Israel Koren, *"Computer Arithmetic Algorithms"*, A. K. Peters, Natick, MA, 2002, 2nd Edition

[5] John P. Hayes, *"Introduction to Digital Logic Design"*, Addison Wesley Publishing Company, 1993.

[6] 경종민, 박인철 외 공저, *"고성능 마이크로프로세서 구조 및 설계 방법"*, 대영사, pp. 316-338

[7] Peter M. Kogge, *"The Architecture of Pipelined Computers"*, New York : Hemisphere, 1981.

[8] M.R. Santoro and M.A. Horowitz, SPIM : "A pipelined 64 x 64 iterative multiplier", *IEEE Journal of Solid-State Circuits 24* (Apr. 1989), pp. 487~493

[9] Gensuke Goto, Atsuki Inoue, Ryoichi Ohe, Shoichiro Kashiwakura, Shin Mitarai, Takayuki Tsuru, and Tetsuo Izawa, "A 4.1-ns Compact 54x54-b Multiplier Utilizing Sign-Select Booth Encoders", *IEEE Journal of Solid-State Circuits, Vol.32, No.11*, November, 1997

[10] 박종환, "32비트 RISC/DSP 프로세서를 위한 17비트 x 17비트 곱셈의 설계", *연세대학교 석사학위 졸업 논문*, pp. 11~12, 16~18, 27~28, 1999

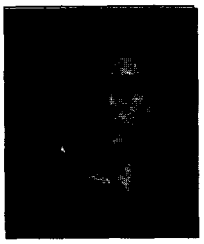
[11] 홍인표, "멀티미디어 데이터 처리에 적합한 SIMD 곱셈누적 연산기의 설계", *연세대학교 석사학위 졸업 논문*, pp. 23, 2001

[12] H. Handshuh, P. Paillier, *Proc. Of 3rd International Conference on CARDIS*, 1998., pp. 372

[13] Zhu Kejia, Xu Ke, Wang Yang, Min Hao, "A Novel ASIC Implementation of RSA Algorithm", *the 5th International Conference in ASIC*, Oct. 2003, pp. 1300-1303

[14] Ching-Chao Yang, Tian-Sheuan Chang, and Chein-Wei Jen, "A new RSA cryptosystem hardware design based on Montgomery's algorithm", *IEEE Trans. Circuits and Systems II: Analog and digital Signal Processing., vol. 45, No 7*, pp. 908-913, Jul. 1998.

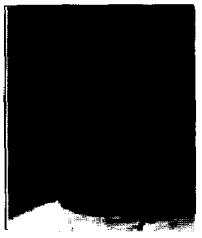
허 석 원(Seok-Won Heo) 준회원



2002년 2월 : 한양대학교
전자컴퓨터공학부 학사
2004년 2월 : 연세대학교
전기전자공학과 석사
2004년 2월~현재 : 삼성전자
반도체총괄 연구원

<관심분야> Microprocessor, Embedded System, Crypto-processor, Arithmetic Unit.

김 문 경(Moon-Gyung Kim) 준회원



1997년 2월 : 연세대학교
전자공학과 학사
1999년 2월 : 연세대학교
전자공학과 석사
1999년 3월~현재 : 연세대학교
전기전자공학과 박사과정

<관심분야> Microprocessor, SMT Microprocessor, Crypto-processor

이 용 석(Yong-Surk Lee) 정회원

1973년 2월 : 연세대학교
전기공학과 학사
1977년 2월 : University of Michigan, Ann Arbor 석사
1981년 2월 : University of Michigan, Ann Arbor 박사
1993년~현재 : 연세대학교
전기전자공학과 교수

<관심분야> Microprocessor, SMT Microprocessor, Network-processor, Crypto-processor