제22회 한국정보처리학회 추계학술발표대회 논문집 제11권 제2호 (2004. 11)

암호화 알고리즘의 효율적인 HW/SW Codesign 기법

이정락, 송문빈, 정연모
경희대학교 전자공학과
e-mail:lager7@khu.ac.kr

Efficient HW/SW Codesign Techniques of Cipher Algorithms

Jounglak Yie, Moonvin Song, Yunmo Chung
Dept. of Electronic Engineering, Kyung Hee University

요 약

본 논문은 SoC 환경에서 암호화 알고리즘의 처리 성능을 향상시키기 위해 각 노드의 실행 시간을 비교하여 하드웨어와 소프트웨어로 codesign 하였다. 암호화 알고리즘으로서는 DES와 SHA-1을 통합 설계하여 적용하였다. 본 논문에서의 codesign 방법을 altera의 excalibur에서 구현하여 실행 시간 및 메모리 크기 그리고 회로의 게이트 크기를 비교 대상으로 하였다. 수행 결과에 따른 분석에 의하면 세 가지 비교 대상에 최적화하여 codesign 성능을 찾을 수 있었다.

I. 서론

반도체 산업의 빠른 발전은 프로세서, 기억장치, 입출력 장치 등과 같이 서로 다른 기능을 수행하는 여러 소자들을 통합하여 하나의 칩 자체가 시스템 역할을 하는 SoC(System on a Chip)의 구현을 가능하게 하였다. 반도체 공정 기술의 발전은 단일 칩에 구현할 수 있는 SoC의 집적도를 매년 50% 정도 증가 시키고 있다. 반면에 이를 구현하기 위한 설계 기술은 약 20% 향상되고 있다[2]. 따라서 SoC를 설계할 때 해결해야 할 문제인 복잡성과 생산성 문제를 해결하기 위해서 새로운 설계 개념과 방법론이 절실하게 요구된다.

본 논문에서는 SoC 설계 생산성을 향상시키기 위하여 하드웨어와 소프트웨어로 시스템을 분할한 후 성능 평가를 하였다. 효율적으로 하드웨어와 소프트웨어의 분할을 위해서는 보다 큰 응용 시스템을 적용해야 하지만, 본 논문에서는 비록 규모가 작지만 다양한 분석이 가능한 암호화 알고리즘을 적용하였다. 여기에 적용한 응용 암호화 알고리즘으로서는 DES와 SHA-1을 통합 설계 하였고, 설계 환경으로서는 altera의 excalibur 상에서 구현 및 검증하였다.

II. HW/SW 분할

하드웨어와 소프트웨어의 분할은, 설계하고자 하는 시스템을 설계 사양에 부합하도록 하드웨어와 소프트웨어로 나누는 과정을 말한다.

하드웨어와 소프트웨어의 분할 설계 과정은 먼저 설계 하고자 하는 시스템 성능의 요구사항을 분석한다. 시스템 요구사항에는 기능, 성능, 저전력, 비용, 그리고 시스템 개발 시간 등이 있다[3,4,5]. 요구된 시스템 디자인을 위해서 각 단계의 세부과정은 수정할 수 있다[6]. 그리고 시뮬레이션 모델링은 설계 오류를 줄이기 위하여 분석 과정을 거친 후 이루어진다. 이러한 상위 레벨 모델링 후 효율적으로 성능의 요구사항을 만족시키기 위해 하드웨어와 소프트웨어로 기능을 분할한다.

[그림 1]은 일반적인 하드웨어와 소프트웨어를 분할하는 방법을 나타낸다[7].

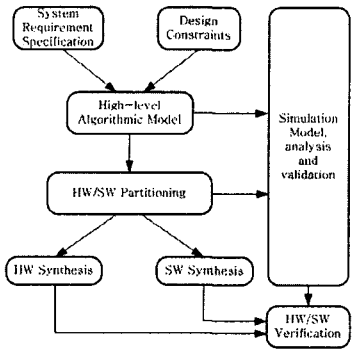

그림 1. 일반적인 하드웨어 소프트웨어 분할 설계 방법

그림과 같이 일반적인 하드웨어와 소프트웨어의 분할은 설계자가 사용하는 PC상에서 각 노드를 부하를 측정하여 그 기준으로 나눈다. 그러나 이 방법으로 설계 할 경우 일반적인 PC 시스템과 임베디드 시스템의 서로 다른 시스템 구조로 인한 부하 노드의 역전문제가 발생할 수 있다.

본 논문에서 하드웨어와 소프트웨어 분할은 상위 레벨 모델링을 적용하여 각 노드의 실행 시간을 대상 시스템에서 직접 실행하여 실행 시간을 구한다. 그리고 이를 이용하여 각 노드별 실행 시간을 비교하여 부하가 많은 노드를 하드웨어 부분으로 분할하고 그렇지 않은 노드는 소프트웨어로 분할한다.

이러한 방법은 부하 노드를 찾아 하드웨어와 소프트웨어를 codesign할 때 발생할 수 있는 데이터 처리 지연 문제를 최소화 할 수 있다.

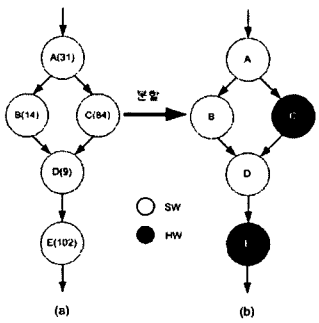

그림 2. 부하 노드를 고려한 분할

[그림 2]의 (a)는 상위 레벨 알고리즘 모델링 과정을 통해서 대상 시스템에서 실행될 노드와 각 노드의 실행 시간을 나타낸다. 각 노드의 실행 시간을 비교하여 실행 시간이 많이 걸리는 부하 노드를 설계자의 경험에 의존하여 C와 E 노드를 선택한다. 이를 바탕으로 하드웨어와 소프트웨어로 분할한 결과가 (b)이다.

본 논문에서는 분할 노드의 성능을 분석하기 위하여 (식 1)에서와 같이 실행 시간과 필요한 메모리 크기, 그리고 게이트 크기를 가진 식을 제시하였다.

$$P=\sqrt{T^2+MR^2+GR^2}$$
$$\begin{pmatrix} P: \text{수행성능} \\ T: \text{실행시간[S]} \\ MR: \text{메모리크기[KB]} \\ GR: \text{게이트크기[Logic cell]} \end{pmatrix} \quad (1)$$

(식 1)을 3차원 상의 좌표로 나타내면 [그림 3]과 같다.

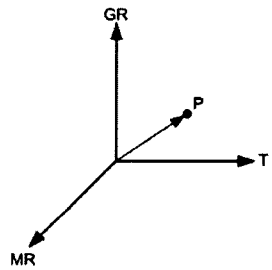

그림 4. 3차원 좌표상의 P

원점에서 P점과의 거리가 가까울수록 높은 성능을 나타낸다.

III. 구현

적용한 암호화 알고리즘은 전자 상거래, 전자 주민카드, 인터넷 보안과 인증에 널리 이용하는 DES와 SHA-1으로 이를 통합 설계 하였다.

DES는 feistel 구조를 갖는 블록 암호화 알고리즘으로 [그림 4]와 같이 64 비트로 이루어진 평문을 받아들여 16 라운드의 처리과정을 거치며, 암호화 과정은 세 단계로 이루어진다.

첫 번째 단계는 64 비트의 입력 비트열의 순서를 바꾸는 IP(Initial Permutation)단계이다. 두 번째 단계에서는 순열과 치환이 포함된 암호 함수를 16 회 반복 수행한다. 마지막으로 64 비트 암호문을 생성

하기 위해 IP-1 블록을 수행한다.

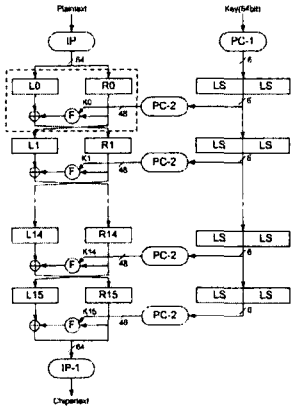

그림 5. DES 블록 다이어그램

서브키의 생성도 세 단계로 나누어 진행된다. 첫 단계로 순열 함수를 통과한 후 16 회에 걸쳐 좌측 순환 이동을 수행한다. 이때 순환 이동 후 생성된 값들이 순열 함수를 통과하여 각 단계의 키를 생성한다.

순열 함수는 각 반복 과정에서 동일하지만 키 비트의 반복적인 이동으로 각각 다른 서브키를 생성한다. 복호화는 암호화와 동일한 구조를 가지고 있다[8].

해쉬 알고리즘인 SHA-1은 다른 암호화 알고리즘과 달리 복호화 과정이 없으며 출력 값인 해쉬 값을 직접 이용하기 때문에 강한 충돌 저항성을 요구하며 복호화가 불가능해야 한다. 본 논문에서 적용한 SHA-1은 미국 NIST(National Institute of Standards and Technology)에 의해 개발된 알고리즘으로 512 비트 단위의 메시지 블록을 처리한다[9]. 512 비트 단위의 메시지 블록은 32 비트의 소 블록으로 나누어 각 단계마다 처리된다. [그림 5]는 SHA-1의 단계 연산의 구조를 보여준다.

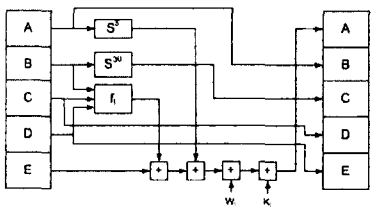

그림 6. SHA-1의 단계 연산

단계 연산은 총 80 단계의 연산을 거치며 이 과정에서 메시지 변수 M_t를 받아서 처리한다. W_t는 다음의 (식 2)와 같이 구한다.

$$W_t = M_t (0 \le t \le 15)$$
$$W_t = S^1[W_{t-3} \otimes W_{t-8} \otimes W_{t-14} \otimes W_{t-16}] \quad (2)$$
$$(16 \le t \le 79)$$

t는 512 비트의 입력 메시지 변수가 거치는 단계를 말하며, 범위는 0≤t≤79 이다.

[그림 6]은 통합 설계한 전체 블록도를 나타낸다.

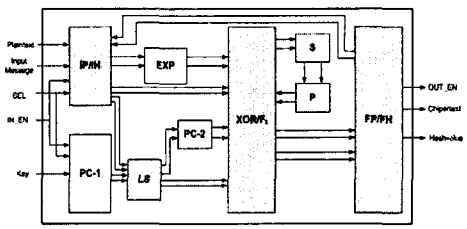

그림 8. 통합 블록도

IP/IH, LS, XOR/Ft 그리고 FP/FH 블록은 DES와 SHA-1의 유사 기능을 통합하여 수행하며, 나머지 블록은 DES 전용으로 사용된다. 통합한 블록을 하드웨어와 소프트웨어로 분할 설계하기 위해 altera의 excalibur를 사용 하였다.

Ⅳ. 분석

[그림 6]의 통합 암호화 알고리즘을 실행 시간 및 공유된 블럭을 기준으로 6가지 경우로 나누었다[표 1참조]. 하드웨어와 소프트웨어로 구현한 후 측정값을 나타낸 것은 [표2,3]과 같다.

표 1. 각 경우를 하드웨어와 소프트웨어로 분할

분할 / 경우	하드웨어	소프트웨어
0	-	All
1	LS	IP/IH, EXP, XOR/Ft, S, P, PC-1, PC-2, FP/FH
2	XOR/Ft	IP/IH, EXP, S, P, PC-1, LS, PC-2, FP/FH
3	LS, XOR/Ft	IP/IH, EXP, S, P, PC-1, PC-2, FP/FH
4	IP/IH, LS, XOR/Ft	EXP, S, P, PC-1, PC-2, FP/FH
5	IP/IH, LS, XOR/Ft, FP/FH	EXP, S, P, PC-1, PC-2

표 2. 각 경우별 소프트웨어 처리 결과

성능 / 경우	소프트웨어		
	클럭수	실행 시간	메모리 크기
0	1,030,969,600	5.154848	162.64
1	831,024,000	4.155120	145.01
2	831,054,400	4.155272	145.00
3	631,108,800	3.155544	127.37
4	526,243,200	2,631216	108.71
5	419,784,000	2.098920	91.15

각 경우를 소프트웨어와 하드웨어로 처리한 부분의 실행 시간, 소프트웨어 처리에 필요한 메모리 크기 및 하드웨어 설계 시 필요한 게이트 크기를 나타낸다. [표 2]에서 경우 1과 경우 2의 실행 시간을 비교하면 거의 유사함을 알 수 있다. 이것은 LS 블록과 XOR/Ft 블록 특성이 치환, XOR 그리고 순환연산으로 구성되어 실행되기 때문이다.

표 3. 각 경우별 하드웨어로 처리 결과

성능 / 경우	하드웨어		
	클럭수	실행 시간[us]	게이트 크기
0	-	-	-
1	41	0.40898	347
2	30	0.29140	79
3	43	0.42956	430
4	47	0.46350	593
5	49	0.48250	830

[그림 6]의 공유된 블록을 하드웨어로 설계하는 경우 [표 2]의 소프트웨어 실행 시간은 감소하나 하드웨어 게이트 크기는 증가한다. 하드웨어로 처리된 부분에서 P에 큰 영향을 주는 것은 실행 시간보다는 게이트 크기인 것을 알 수 있다.

[그림 7]은 각 경우별 P값을 비교한 그래프이다.

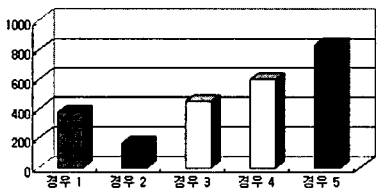

그림 9. 각 경우별 P값 비교

소프트웨어적으로 적합한 명령어가 없는 xor 연산을 주로 수행하는 XOR/Ft 블록을 하드웨어로 구현한, 경우 2의 P값이 가장 적음을 알 수 있다. 즉 경우 2의 성능의 제일 높다.

V. 결론

본 논문은 SoC 환경에서 하드웨어와 소프트웨어를 효율적으로 codesign하기 위하여 각 노드의 실행 시간을 비교한 후, 분할하는 것을 고려하였다. 이를 적용하여 통합 암호화 알고리즘의 성능 향상을 확인하였다. 비교한 성능은 실행 시간 및 메모리 크기 그리고 회로의 게이트 크기이다. 그 결과 세 가지 비교 대상에 최적화하여 codesign 성능을 찾을 수 있었다. 그러나 설계자의 설계 사양에 부합하는 요건을 모두 만족하기 위해서는 각각의 경우를 고려하여 설계해야 한다. 모바일 환경과 같은 임베디드 시스템에서 위와 같은 codesign 방법을 적용하면 성능 향상을 기대할 수 있다.

참고문헌

[1] http://www.altera.com/

[2] Karam S. Chatha and Ranga Vemuri, "Hardware -Software Partitioning and Pipelined Scheduling of Transformative Applications," *IEEE Transactions on VLSI*, Vol. 10, No. 3, pp. 193-208, June, 2002.

[3] P. Waldeck, N. Bergmann, "Dynamic hardware - software partitioning on reconfigurable system - on-chip," *IEEE International Workshop on System-on-Chip*, pp. 102-105, 30 Jun 2 Jul, 2003.

[4] P. Brunet, C. Tanougast, Y. Berviller, S. Weber, "Hardware partitioning software for dynamically reconfigurable SoC design," *IEEE International Workshop on System-on-Chip*, pp. 106-111, 30 Jun-2 Jul, 2003.

[5] Giovanni De Micheli, Rajesh K. Gupta, "Hardware/Software Co-Design," *Proceedings of the IEEE*, Vol. 85, No. 3, pp. 349-365, Mar 1997.

[6] Juanjo Noguera, Rosa M. Badia "HW/SW Codesign Techniques for Dynamically Reconfigurable Architecture," *IEEE Transactions on VLSI*, Vol. 10, No. 4, pp. 339-415, Aug 2002.

[7] Rochit Rajsuma, *System-on-a-Chip : Design and Test*, Artech House, pp. 14-18, 2000.

[8] Man Young Rhee, *Internet Security*, Wiley, pp. 57-160, 2002.

[9] U.S Department of Commerce, *Secure Hash Standard*, FIPS PUB 180-1, NIST, 1995

[10] Steve Fuber, *ARM system-on-chip architecture*, Addison Wesley, pp. 216-220, 2000.