
TMS320C54X DSP보드를 이용한 IDEA의 구현

송 종 관*, 윤 병 우*

IDEA Implementation On TMS320C54X DSP Board

Jong-kwan Song*, Byung-Woo Yun*

요 약

본 논문에서는 데이터 암호화 알고리즘으로 성공적으로 사용되고있는 IDEA에 대한 분석과 이를 TMS320C54X DSP 보드를 사용하여 구현한 결과를 기술한다. 또한 모듈러 ($2^{16}+1$) 곱셈 연산을 사용함으로써 처리시간을 상당히 줄일 수 있음을 보였다. 구현된 결과는 초당 약 250 ~ 300 Mbyte의 처리 속도를 보여준다.

Abstract

This paper describes the principles of IDEA(international data encryption algorithm) which has been widely accepted as a data encryption algorithm and the implementation of the algorithm on the TMS320C54X DSP board is addressed. It is also shown that the processing time is significantly reduced by adapting high speed multiplication modulo ($2^{16}+1$) algorithm. The result shows data rates of about 250~300Mbyte/sec.

1. 서 론

오늘날 사회가 점차 정보화사회로 변해감에 따라 정보에 대한 안전성 문제가 매우 심각하게 대두되고 있다. 정보의 안전한 보호는 군사 외교 분야는 물론이고 상업 및 여러 산업 분야에서도 매우 중요하게 취급되고 있다. 따라서 인터넷이나 전자 매체를 통하여 정보를 주고받을 때 문제가

되는 제3자로부터의 공격을 피할 수 있는 방법이 매우 활발하게 연구되고 있다. 제3자로부터 정보의 공격을 피하기 위해서는 정보를 암호화하여 전송하는 것이 필수적이다. 정보의 보호 방법은 물리적인 접근을 통제하는 것으로부터 비밀 번호의 다단계 이용, 컴퓨터 운영체제의 강화 등 많은 수단이 있다. 그러나 컴퓨터 시스템이나 통신에서 저장과 교류의 대상이 되는 정보 파일의 직접적인 보호가

* 경성대학교 공과대학 전기전자 컴퓨터공학부

접수일자 : 1998년 10월 8일

가장 기본적이고 안전한 수단이다. 이에 따라 많은 정보 암호화 알고리즘이 발표되었다.

인터넷 망이나 초고속 정보 통신망에서는 고속으로 데이터를 송수신하므로 정보의 암호화를 위해서는 알고리즘이 고속으로 수행되어야 한다. 또한 디스크에 데이터를 저장할 때 사용자는 데이터의 저장 속도에 매우 민감하다. 따라서 여러 가지 용도에서 데이터의 암호화를 빠른 시간에 수행하기 위해서는 DSP(Digital Signal Processor) 전용칩을 이용하여 알고리즘을 고속으로 수행하거나 Crypto-API 칩을 설계하여 사용하는 것이 바람직 할 것이다. 본 논문에서는 TMS320C542 EVM보드를 이용하여 암호화 알고리즘 중 성공적으로 보급된 IDEA를 고속으로 구현하였다.

본 논문의 구성은 다음과 같다. II장에서는 IDEA 알고리즘의 분석을 통하여 그 원리를 기술하였다. III장에서는 이를 토대로 DSP 보드에서 어셈블리어언어를 통하여 알고리즘을 구현하고 핵심 연산인 모듈러 곱셈 연산을 고속으로 수행하는 방안을 적용하여 알고리즘의 전체 수행 속도를 개선하였다.

II. IDEA 알고리즘의 분석

IDEA(International Data Encryption Algorithm)는 Lai와 Massey에 의해 개발된 블록 암호화 방식으로 이전까지 자주 사용하고 있던 DES(Data Encryption Standard)의 암호학적 강도를 증가시켜 DES를 대체하기 위하여 제안되었다[1-3]. 이 알고리즘은 PGP(Petty Good Privacy)의 기밀성 서비스 제공용으로 사용되면서 성공적으로 보급되었다.

IDEA에서 평문 및 암호문은 64비트의 블록들로 구성되며 사용자에 의해 결정되는 암호키는 키에 대한 전사공격을 고려하여 128비트로 결정되었다. DES의 암호학적 안정성은 주로 XOR연산과 비선형 S-box에 의존하는데 반해 IDEA에서는 평문의 통계적 특성과 암호문의 통계적 특성을 복잡하게 하기 위해 세가지 서로 다른 연산을 이용한다. 확산(Diffusion)은 평문의 각 비트와 암호 키의 각 비트가 암호문의 모든 비트에 영향을 주어야 하므로 IDEA에서는 확산 특성이 우수하게 설계되었다. 혼돈(confusion) 특성을 위해서는 서로 다른 세가지

연산을 사용하며 세 함수는 모두 두개의 16비트 시퀀스를 입력 받아 하나의 16비트 시퀀스를 출력한다. 이 중 첫번째 연산은 비트 단위의 XOR() 연산이며, 두 번째는 입출력이 모두 비부호(unsigned) 정수인 모듈러 2^{16} 덧셈 연산이고, 세 번째는 비부호 정수에 대한 모듈러 $2^{16}+1$ 곱셈 연산이다. 여기서 모듈러 $2^{16}+1$ 곱셈 연산에서 입력이 0인 경우 이 수는 2^{16} 을 의미한다. 모듈러 $2^{16}+1$ 곱셈에 대한 의사 코드는 그림 1과 같다.

A, B : 16비트 입력

multi16(A,B)

```
{
    if(A==0) A = 2^16;
    if(B==0) B = 2^16;
    output = (AB)%(2^16+1); if(output==2^16) output = 0;
}
```

그림 1. 모듈러 $2^{16}+1$ 곱셈에 대한 의사 코드

Fig. 1. Pseudo code of modular $2^{16}+1$ multiplication

1. IDEA 암호화 과정

그림2는 IDEA의 전체 블록도를 나타낸다. 그림에서 보듯이 IDEA는 전체 8라운드로 구성되며 최종 출력단에서 출력을 변환하는 하나의 출력 변환단을 갖고 있다. 이 출력 변환단은 부호화와 복호화에 동일한 구조를 이용하기 위한 것이다[4]. 각 라운드는 평문으로부터 주어지는 4개의 16비트 부분 블록을 출력으로 낸다. 이 때 각 라운드에서의 암호키는 6개의 16비트 서브 키를 이용하게 되며 출력 변환단에서 4개의 16비트 서브 키를 이용하게 된다. 따라서 전체 변환에서는 52개의 서브키가 사용된다.

IDEA의 복호화 과정은 암호화 과정과 근본적으로 동일하다. 단, 복호화 때 사용되는 서브키는 암호화에서의 서브키로부터 구해진다.

2. IDEA의 키 스케줄

IDEA에서 사용되는 52개의 암호화 키는 외부에서 입력되는 128비트의 키로부터 생성하여 사용하며 생성 과정은 다음과 같다[5].

1. 128 비트 키를 8개의 16비트 블록으로 나누어 그 중 앞의 6개의 블록을 서브키로 택한다.

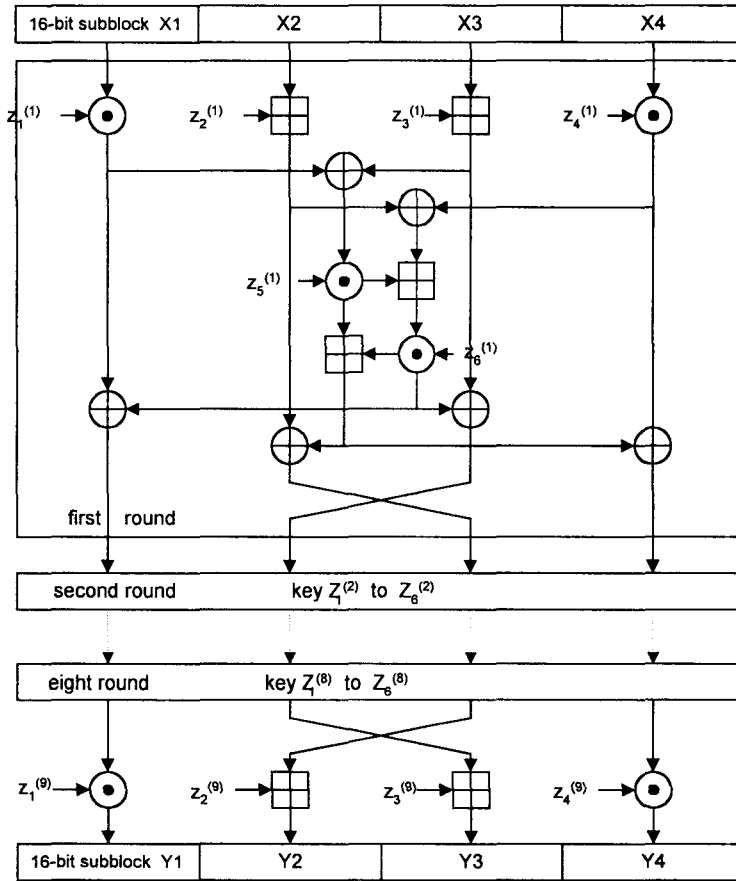


그림 2. IDEA의 블럭도
Fig. 2. Block diagram of IDEA algorithm

2. 128비트 키를 25비트씩 좌측으로 순환(rotate) 시킨다.
3. 1, 2의 과정을 반복하여 8라운드까지의 서브 키를 선택한다.
4. 최종 출력단에서 사용할 4개의 서브 키를 같은 방식으로 택한다.

IDEA의 복호화에 사용되는 52개의 16비트 서브 키는 암호화에 사용되는 서브 키로부터 표 2-1과 같이 구해진다. 여기서 \oplus 의 모듈러 $2^{16}+1$ 곱셈 연산에 대한 역원이고 \ominus 의 모듈러 2^{16} 덧셈에 대한 역원을 나타낸다.

복호화 과정은 암호화 과정과 동일하며 단지 사용되는 서브키가 다를 뿐이다. 이러한 과정에 의해

복호화가 된다는 사실에 대해서는 [2]에 자세한 내용이 주어져 있다.

III. IDEA 알고리즘의 구현

본 연구에서는 IDEA을 TI사의 고속 DSP 칩인 TMS320C54X를 이용하여 구현하였다. IDEA에서 사용되는 핵심 연산은 다음과 같다.

- \oplus : 16비트 시퀀스에 대한 비트별 논리 XOR 연산
- [+] : " " 모듈러 2^{16} 덧셈
- (.) : " " 모듈러 $2^{16}+1$ 곱셈

이들은 모두 16비트 시퀀스를 입력 및 출력 데이터로 사용한다. 구현에 사용된 DSP 칩, TMS320C54X

표 1. IDEA의 복호화에 사용되는 서브키

Table 1. Subkeys in the IDEA decoding

라운드	암호화 키 블록						복호화 키 블록					
1	$Z_1^{(1)}$	$Z_2^{(1)}$	$Z_3^{(1)}$	$Z_4^{(1)}$	$Z_5^{(1)}$	$Z_6^{(1)}$	$Z_1^{(9)-1}$	$-Z_2^{(9)}$	$-Z_3^{(9)}$	$Z_4^{(9)-1}$	$Z_5^{(8)}$	$Z_6^{(8)}$
2	$Z_1^{(2)}$	$Z_2^{(2)}$	$Z_3^{(2)}$	$Z_4^{(2)}$	$Z_5^{(2)}$	$Z_6^{(2)}$	$Z_1^{(8)-1}$	$-Z_2^{(8)}$	$-Z_3^{(8)}$	$Z_4^{(8)-1}$	$Z_5^{(7)}$	$Z_6^{(7)}$
3	$Z_1^{(3)}$	$Z_2^{(3)}$	$Z_3^{(3)}$	$Z_4^{(3)}$	$Z_5^{(3)}$	$Z_6^{(3)}$	$Z_1^{(7)-1}$	$-Z_2^{(7)}$	$-Z_3^{(7)}$	$Z_4^{(7)-1}$	$Z_5^{(6)}$	$Z_6^{(6)}$
4	$Z_3^{(4)}$	$Z_2^{(4)}$	$Z_3^{(4)}$	$Z_4^{(4)}$	$Z_5^{(4)}$	$Z_6^{(4)}$	$Z_1^{(6)-1}$	$-Z_2^{(6)}$	$-Z_3^{(6)}$	$Z_4^{(6)-1}$	$Z_5^{(5)}$	$Z_6^{(5)}$
5	$Z_1^{(5)}$	$Z_2^{(5)}$	$Z_3^{(5)}$	$Z_4^{(5)}$	$Z_5^{(5)}$	$Z_6^{(5)}$	$Z_1^{(5)-1}$	$-Z_2^{(5)}$	$-Z_3^{(5)}$	$Z_4^{(5)-1}$	$Z_5^{(4)}$	$Z_6^{(4)}$
6	$Z_1^{(6)}$	$Z_2^{(6)}$	$Z_3^{(6)}$	$Z_4^{(6)}$	$Z_5^{(6)}$	$Z_6^{(6)}$	$Z_1^{(4)-1}$	$-Z_2^{(4)}$	$-Z_3^{(4)}$	$Z_4^{(4)-1}$	$Z_5^{(3)}$	$Z_6^{(3)}$
7	$Z_1^{(7)}$	$Z_2^{(7)}$	$Z_3^{(7)}$	$Z_4^{(7)}$	$Z_5^{(7)}$	$Z_6^{(7)}$	$Z_1^{(3)-1}$	$-Z_2^{(3)}$	$-Z_3^{(3)}$	$Z_4^{(3)-1}$	$Z_5^{(2)}$	$Z_6^{(2)}$
8	$Z_1^{(8)}$	$Z_2^{(8)}$	$Z_3^{(8)}$	$Z_4^{(8)}$	$Z_5^{(8)}$	$Z_6^{(8)}$	$Z_1^{(2)-1}$	$-Z_2^{(2)}$	$-Z_3^{(2)}$	$Z_4^{(2)-1}$	$Z_5^{(1)}$	$Z_6^{(8)}$
9	$Z_1^{(9)}$	$Z_2^{(9)}$	$Z_3^{(9)}$	$Z_4^{(9)}$			$Z_1^{(1)-1}$	$-Z_2^{(1)}$	$-Z_3^{(1)}$	$Z_4^{(1)-1}$		

는 16 비트 프로세서이므로 IDEA 구현에 적합하다. 세 가지 핵심 연산 중 처리 속도에 가장 많은 영향을 주는 연산은 모듈러 $2^{16}+1$ 곱셈 연산으로 실제 가장 많은 처리 시간을 요하는 부분이다. 따라서 이 연산을 효율적으로 구현하는 것이 IDEA의 고속 구현에 필수적이다. 모듈러 $2^{16}+1$ 곱셈 구현에 있어서 입력이 0인 경우 2^{16} 으로 대체하게 된다. 2^{16} 을 표현하기 위해서는 17비트가 소요되므로 double precision 연산을 요구하게 된다. 또한 매번 곱셈 수행 전에 입력 데이터가 0인지를 검사하는 루틴이 필요하다. 그림 3은 이에 대한 의사코드를 보여주며, 그림 4는 이러한 방식으로 구현된 TMS320C54X의 코드를 보여 준다.

모듈러 $2^{16}+1$ 곱셈 연산의 구현은 다음과 같은 방법으로 수행 속도를 개선할 수 있다. 처리 속도 개선을 위하여 첫 번째 입력 a가 영인 경우나 두 번째 입력 b가 영인 경우는 매우 드물게 발생한다는 점을 이용한다. 즉 a=0 또는 b=0인 경우는 다른

```

If a=0    p = 0x00010001 b
Else if b=0 p = 0x00010001 a
Else
    |
    q = a*b
    p = (q&0xffff)-(q>>16)
    If p<0 p = p + 0x00010001
    }
Return(p&0xffff)
    
```

그림 3. 모듈러 $2^{16}+1$ 곱셈 연산의 의사코드
Fig. 3. Pseudo-code of modular $2^{16}+1$ multiplication

```

Multi16:
LD      *DATA1, A
LD      *DATA2, B
BC      a_zero, AEQ
BC      b_zero, BEQ
; A != 0, B != 0
LD      *DATA1, T
MPYU   *DATA2, A
B       modulo

B_zero:
; A != 0, B = 0
LD      *DATA1, 16, A
B       modulo

A_zero:
BC      ab_zero, BEQ
; A = 0, B != 0
LD      *DATA2, 16, A
B       modulo

Ab_zero:
ST      #1, *OUTPUT
RET

Modulo:
LD      #1h, 16, B ; B = 2^16+1
ADD     #1, B

Next_sub:
STM     #15, BRC
RPTB   end_multi-1
SUB     B, 15, A
BC     next01, AGEQ
ADD     B, 15, A

Next01:
LD      A, 1, A

End_multi:
LD      A, -16, A
LD      #1h, 16, B ; B = 10000h
SUB     A, B ; if result is 2^16
BC     next02, BNEQ; result = 0
LD      #0, A

Next02:
STL     A, *OUTPUT
RET
    
```

그림 4. 모듈러 $2^{16}+1$ 곱셈 연산의 구현 코드
Fig. 4. Implemented code of modular $2^{16}+1$ multiplication

65535 가지의 입력보다 매우 드물게 발생한다. 여기서 a, b 중 적어도 하나가 영인 경우에는 그 곱셈의 결과 또한 영임을 볼 수 있다. 따라서 a, b 중 적어도 하나가 영인 경우를 DSP에서 단 1 cycle에 검사 할 수 있다. 이 검사를 통하여 곱셈 결과가 영인 경우 즉, a, b 둘 중 하나가 영인 경우에만 수정을 하면 된다. 따라서 대부분의 경우에는 수정이 필요치 않고 아주 드물게 수정이 필요하므로 전체적으로 보아 상당한 속도 개선이 이루어진다. 그림 5는 고속 곱셈 연산의 의사코드를 보여주며, 그림 6은 실제 구현된 어셈블리 코드를 보여준다.

```

Q = a*b
p = (q&0xffff)-(q>>16)
If p=0
    p = 0x00010001-a
    p = 0x00010001-a
Return(p&0xffff)
    
```

그림 5. 고속 모듈러 $2^{16}+1$ 곱셈 연산의 의사 코드
Fig. 5. Pseudo-code of high speed modular $2^{16}+1$ multiplication

```

Multi16:
LD      *DATA1, T
MPYU   *DATA2, A
AND     #0ffff, A, B
SUB     A, -16, B
BC      iszero, BEQ
; if A is 0, either DATA1 or DATA2 is 0
XC      1, NC      ; if AL-AH<0, add 1
ADD     #1h, B
Mul_out:
STL     B, *OUTPUT
RET
Iszero:
LD      #1, B
; if(DATA1==0) OUTPUT = 2^16+1 - DATA2
SUB     *DATA1, B
; if(DATA2==0) OUTPUT = 2^16+1 - DATA1
SUB     *DATA2, B
B       mul_out
    
```

그림 6. 고속 모듈러 $2^{16}+1$ 곱셈 연산의 구현 코드
Fig. 6. Implemented code of high speed modular $2^{16}+1$ multiplication

고속 곱셈 연산을 사용하지 않는 경우 한 블록의 처리에는 6378 DSP cycle이 소요된다. 이는 25nsec clock 사용 시 약 50Kbyte/sec의 처리속도를 가짐을 의미한다. 이러한 처리속도는 고속 곱셈 연산을 사용함으로써 상당히 향상되었다. 고속 곱셈 연산은 데이터가 영인 경우와 영이 아닌 경우 소요 cycle이 다르지만 일반적인 데이터를 사용한 실험 결과 한 블록의 처리에 약 1063~1276cycle이 소요되었다. 이는 역시 25nsec clock 사용 시 약 250~300Kbyte/sec의 처리 능력을 가지므로 약 5~6배의 처리 속도 개선이 이루어짐을 알 수 있다.

IDEA 알고리즘의 키 운용 체계는 그림 7과 같다. 프로그램 초기화에서 파일 키는 랜덤 값을 가지는 초기 파일 키로부터 주어진다. 이 초기 파일 키는 내장 키와 결합(구현에서는 XOR 연산을 사용)하여 적용 키를 구하게 되고 이를 이용하여 파일을 암호화 한다. 암호화가 완료되면 암호 파일 중 128비트를 다음 번 암호화를 위한 파일 키로 저장하고 이를 또한 암호화된 파일의 헤더에 포함시켜 복호화 가능토록 한다. 이러한 방식에 따라 암호화 시 마다 매번 파일 키를 바꾸어 주게 된다.

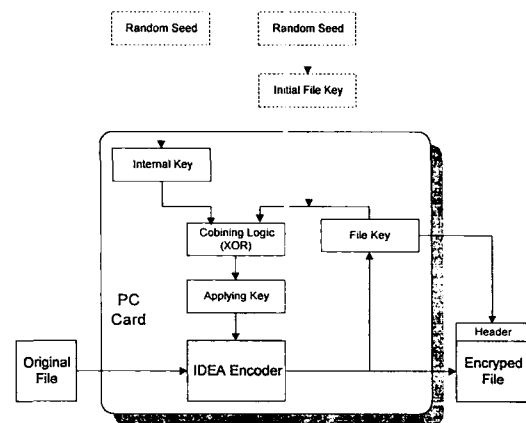


그림 7. 키 운용 체계
Fig. 7. Key scheduling diagram

IV. 결 론

본 논문에서는 암호화 알고리즘인 IDEA를 분석하고 TMS320C54X DSP 보드에서 어셈블리 언어

로 구현하였다. 또한 수행 속도에 매우 큰 영향을 미치는 핵심 연산인 모듈러 곱셈 연산에 대한 고속 알고리즘을 채택하여 속도 개선을 이루었다. 이 연구 결과는 노트북 등에서와 같이 이동성을 중시하는 소형 경량의 보안 시스템 개발 및 클라이언트/서버에 바탕을 둔 분산 망에 사용자의 인증, 전자 결제 시스템에서의 디지털 서명, 정보를 디스크에 저장할 때 정보의 보안 등을 위해 적용될 수 있다.

참고문헌

- [1] X. Lai and J. L. Massey, "A Proposal for a New Block Encryption Standard," EUROCRYPT 90, Proceedings, pp. 389-404, Springer, 1990
- [2] M. J. Wiener, "Efficient DES Key Search", Preprint
- [3] E. Biham and A. Shamir, "Differential Cryptanalysis of the full 16-round DES", CRYPTO'92, Proceedings, Springer, 1992
- [4] 한국전자통신연구소, PC 카드 보안을 이용한 클라이언트와 서버 인증 기법 연구, 1996년
- [5] X. Lai, On the Design and Security of Block Ciphers, Dissertation, ETH Zurich, No. 9752, 1992



송 종 관(宋鍾官)
 1989년 2월 부산대학교 전자공학과 졸업(공학사)
 1991년 2월 한국과학기술원 전기 및 전자공학과 졸업(공학 석사)
 1995년 8월 한국과학기술원 전기 및 전자공학과 졸업(공학 박사)

1995년~1991년 한국이동통신 중앙연구소 선임연구원으로 근무

1997년 이후 경성대학교 전기전자 컴퓨터공학부 조교수로 재직

* 주관심분야 : 영상처리 및 통신 등



윤 병 우(尹炳愚)
 1987년 부산대학교 전자공학과 졸업
 1989년 부산대학교 대학원 전자공학과 석사학위 취득
 1992년 부산대학교 대학원 전자공학과 박사학위 취득

1993년~1995년 전자통신연구원 이동통신연구단 선임연구원

1995년~현재 경성대학교 전기전자·컴퓨터공학부 조교수

* 관심분야 : 적응신호처리, 이동통신, ASIC 등