Cortex-M3기반 System 보안 플랫폼 구현에 대한 연구

박정길* · 김영길*

*아주대학교

Implementation of system security platform based on Cortex-M3

Jung-kil Park* · Young-kil Kim**

*Ajou University

E-mail : piiuiiq@ajou.ac.kr

요 약

임베디드 시스템에서 타 업체의 요청으로 코드를 공개해야 되면 하드웨어 복사 방지 대책을 세워야 힘들게 개발한 제품을 지킬 수 있다. 보안 IC칩을 사용하지 않고, 핵심코드와 하드웨어 복사를 방지하면서 펌웨어 코드를 공개할 수 있는 보안 플랫폼을 제안하고자 한다.

Cortex-M3기반 MCU의 내부 Flash의 영역을 IAP(In-application programming)와 APP(Application)으로 나누어, IAP영역에는 핵심코드와 보안인증 코드를 배치하고, APP에서는 타 업체의 개발자가 공개된 핵심코드의 Prototype을 사용하여 구현한 Application이 배치되는 플랫폼을 제안한다.

ABSTRACT

In embedded system, if firmware code is opened by other company, must devise hardware copy prevention. That guard valuable product. Not used security IC, Suggested platform is source code open method that prevent core code and hardware copy. And that open firmware code for other company programmer.

Suggest system security platform based on Corex-M3. that consist of IAP(In-application programing) and APP(Applicataion). IAP contain core code and security confirm code. APP is implement by other company developer using core function prototype.

키워드

Cortex-M3, Security, firmware, source code open

Ⅰ. 서 론

산업 환경에서 타 업체가 직접 기능을 수정하기 위하여, 펌웨어 코드 공개를 요구하기도 한다. 이때, 제품 카피의 가능성과 핵심 기술의 노출이 우려되어 코드 공개를 거절하게 된다. 하지만 타 업체가 시장의 요구에 빠르게 대응하기 위해서 펌웨어 코드 공개를 강력하게 요구하고 업체와의 관계로 인하여 공개하는 경우가 발생한다.

이러한 상황에서 힘들게 개발된 시스템을 보호하기 위한 방안이 필요하다. 핵심코드와 하드웨어 복사 방지를 위한 보안 플랫폼을 구축하면 이러한 상황에 유연하게 대처가 가능해 질 것이다.

ARM은 스마트폰 시장에서 95%, 오토모티브 인포메이션 95%, 마이크로컨트롤러 25% 등 임베디드 시스템에서 압도적인 점유율을 기록하고 있다. Cortex-M3는 ARM의 Microcontroller unit을 위한 라인업으로, 32bit급 MCU시장에서 주도적 위치를 차지하고 있다.

본 논문에서는 Cortex-M3기반의 하드웨어 저작 보호 인증과 핵심코드를 보호하면서 타 업체에 펌웨어를 공개하는 시스템 플랫폼을 제안하고 구현에 대해서 연구하였다.

본 논문의 순서는 다음과 같다. 2장에서는 제안되는 보안플랫폼을 소개한다. 3장에서는 제안한 보안플랫폼을 평가한다. 그리고 4장에서 결론을 맺는다.

II. 제안하는 보안 플랫폼

1. 시스템 구성

하드웨어의 복사 방지를 위해서 대부분 Security IC를 써서, Chip 제조사로부터 제공받은 보안인증 라이브러리를 통하여 하드웨어 보안인증을 한다. 하지만 인터페이스를 위한 pin이 낭비되며, 별도의 비용을 지불하게 되며, 보안을 외부에 맡겨야 하는 점이 단점이다. Secutiy IC 사용 없이 자체 보안인증을 하도록 펌웨어가 보안인증에 성공해야 정상 동작하는 시스템을 연구하였다.

이를 위해서는 펌웨어가 보안인증에 성공해야 정상 동작해야하며, 보안인증 코드를 외부에 노출되지 않아야 한다. 따라서 공개해도 되는 코드와 펌웨어의 핵심코드 및 보안인증 코드를 두벌의 프로그램으로 나누고, 두 프로그램이 맞물려 호출되는 구조를 고안하였다. 이는 MCU의 내부 Flash의 영역을 IAP(In-application programming) 프로그램과 APP(Application)프로그램으로 나누어, IAP 영역에는 핵심코드와 보안인증 코드를 배치하고, APP에서는 타 업체에 공개할 Application을 배치하는 구조이다.

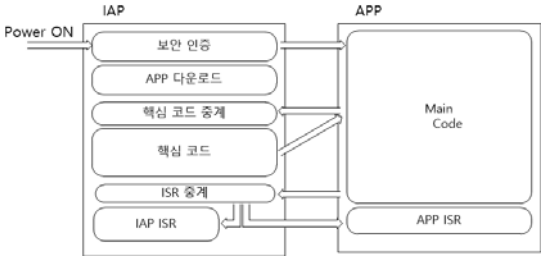

그림1. 제안하는 보안 플랫폼 시스템 구성

전원이 켜지면, IAP프로그램이 먼저 실행되어 보안인증을 수행하고, 성공하면 APP프로그램으로 분기를 해주고, 실패하면 무한루프에 빠지도록 하였다. APP프로그램은 핵심코드의 Prototype만 정의된 가짜 라이브러리를 이용하여 컴파일 된다. 가짜 라이브러리를 호출하게 되면 IAP의 핵심코드 중계가 실행되어 실제 핵심코드를 호출한 것처럼 연결시켜 준다. 또한 인터럽트 발생시, IAP에서 사용되는 인터럽트와 APP에 사용을 허용한 인터럽트 모두 호출되도록 ISR(Interrupt service routine)을 중계한다.

2. 핵심코드 중계

APP와 IAP는 분리된 프로그램으로써 APP이 IAP에 속한 핵심코드 주소를 모른 상태에서 두 프로그램을 연결시켜주기 위해서 SVC(System service call)을 사용하였다. APP은 SVC명령으로 이루어진 가짜 라이브러리를 호출하여 IAP의 SVCall ISR으로 분기하고, 핵심코드 중계가 핵심코드가 실행되도록 연결해준다.

APP과 IAP사이의 핵심코드 중계가 정상적으로 파라미터를 전달하여 동작하기위해서 AAPCS (Procedure Call Standard for ARM Architecture)를 준수하여야 한다. AAPCS에 의하면 함수 호출 시 파라미터 전달은 r0-r3 레지스터를 이용하되 부족 시 스택을 이용한다. return값은 r0, r1 레지스터를 이용한다. 또한 함수 호출 시에 Callee는 Caller의 Context를 복원의 책임이 있다. 따라서 I 핵심코드 중계와 APP의 가짜 라이브러리는 스택과 레지스터를 손상하지 않고 제어하기위해 어셈블리어로 구현 하였다.

가짜 라이브러리는 핵심코드 중계번호로 SVC 명령을 사용한다. SVC명령에 의해 다음 명령이 스택에 RA(Return Address)로 저장된다.

핵심코드 중계 코드는 IAP의 SVC ISR로써 스택의 RA값을 참고하여 핵심코드 중계번호를 파악하고 RA값을 파악한 핵심코드 시작 주소로 변경을 한다. SVC ISR이 수행되고 복귀할 때, 핵심코드 시작주소로 분기하게 되어 APP이 핵심코드를 호출한 것처럼 자연스럽게 핵심코드가 실행되게 된다.

표1. 가짜 라이브러리 코드

```
        PUBLIC   Library_Call_Function
Library_Call_Function
        svc   (핵심코드 중계번호)
```

표2. 핵심코드 중계 코드

```
        PUBLIC   _SVC_Handler
_SVC_Handler
  * r?~r?, lr을 스택에 Push
  * 스텍에서 RA(Return Address)를 r?에 읽음.
  * RA - 2를 하여 r?에 저장.
  * r?를 주소로 SVC명령을 읽어옴.
  * r?를 마스킹하여 SVC 번호를 얻음.
  * SVC번호로 핵심코드 함수주소를 RA에 저장.
  * 스택에 Push한 값 Pop하며 분기.
```

3. 인터럽트 중계

APP의 라이브러리 호출로 인한 SVC Interrupt 발생시 IAP의 SVC ISR이 실행되어야 하며, 핵심코드에서 사용되는 ISR도 IAP에 위치하여 코드를 감출 수 있어야 한다. 또한 APP에서도 ISR 실행을 통하여 application을 구현할 수 있어야 프로그램 작성이 용이해 진다. 따라서 인터럽트가 발생하면 우선 IAP의 인터럽트가 실행되고, APP에서 사용을 허용한 인터럽트이면, 분기를 해주는 인터럽트 중계를 고안했다.

Cortex-M3는 NVIC(Nested Vectored Interrupt Controller)를 통하여 인터럽트를 처리한다. NVIC는 Vector Table을 참고하여 ISR을 호출하며,

Vector Table Offset 레지스터에 의해 Table의 위치를 변경 할 수 있다. 자연스럽게 중계가 되기 위해 APP의 Start-up 코드에서 Vector Table Offset을 APP Vector가 아닌, IAP Vector 위치로 설정했다. IAP Vector에서 적용할 제품의 정책에 따라 APP에게 사용 허가된 인터럽트의 ISR에 인터럽트 중계 코드를 넣어 인터럽트 중계가 APP ISR이 수행되고, 불허된 ISR은 IAP ISR이 수행된다. 인터럽트 중계는 발생한 인터럽트 번호를 참고하여 APP Vector에 등록된 ISR로 분기하도록 수행된다.

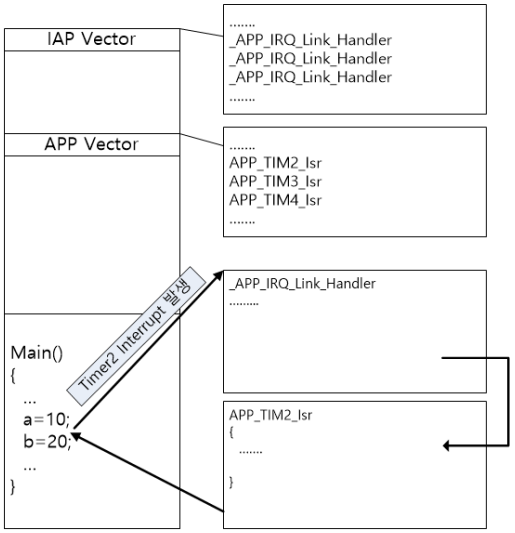

그림2. 인터럽트 중계 설명

표3. 인터럽트 중계 코드

```
PUBLIC  _APP_IRQ_Handler
_APP_IRQ_Handler
 * IPSR를 마스킹하여 인터럽트 번호를 r?에 저장.
 * APP Vector 시작주소+ 4*r?하여 r?에 저장.
 * r?값으로 분기.
```

4. 보안 인증

하드웨어 복사방지를 위해서는 제품 개별로 고유한 인증키를 갖고 있어야 한다. 하드웨어 인증키 생성을 위해서 별도의 인증키 생성프로그램을 사용하여 하드웨어의 비휘발성 메모리에 인증키를 생성하도록 하였다.

하드웨어 복사를 위해 Flash Data를 그대로 카피를 한 경우에는 보안 인증키가 해당 하드웨어에 맞지 않으므로 보안인증이 실패하도록 하였다.

III. 실 험

1. 핵심코드 및 인터럽트 중계 확인

IAP에 Add(), Sub(), Uart1_printf()를 위치시키고, APP의 Main의 본문과 Timer ISR에서 호출하여 핵심코드 중계와 인터럽트 중계를 확인 하였다.

표4. 핵심코드 및 ISR 중계 확인 코드

```
void Uart1_Printf(char *fmt,...);
int  Add(int a, int b);
int  Sub(int a, int b);
 ...
void main(void) {
 ...
 Uart1_Printf("\tCall printf!! %d, %.4f\n",
              123456789, 1000.0001);
 Uart1_Printf("\tAdd : %d + %d = %d\n",
              10, 20, Add(10,20));
 Uart1_Printf("\tSub : %d - %d = %d\n",
              10, 20, Sub(10,20));
 ...
}
 ...
void TIM2_IRQHandler(void){
 ...
 Uart1_Printf("TIM2_IRQ_Start\n");
 Uart1_Printf("\tCall printf!! %d, %.4f\n",
              123456789, 1000.0001);
 Uart1_Printf("\tAdd : %d + %d = %d\n",
              10, 20, Add(10,20));
 Uart1_Printf("\tSub : %d - %d = %d\n",
              10, 20, Sub(10,20));
 Uart1_Printf("TIM2_IRQ_End\n");
}
```

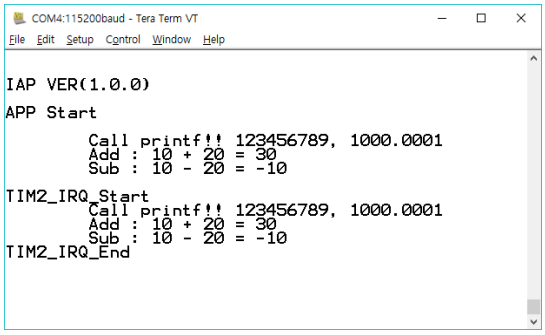

그림3. 핵심코드 및 인터럽트 중계 실행

2. 핵심코드 및 인터럽트 중계 지연시간

핵심코드 중계에 따른 지연시간을 평가하기위하여 Add() 함수를 1000번 호출하여 핵심코드 중계 미사용과 사용 시 수행시간을 측정해하였다.

인터럽트 중계에 따른 지연시간을 평가하기 위하여 Software Interrupt trigger로 Timer2 인터럽트를 1000번 발생하여, 인터럽트 중계 미사용과 사용 시 수행시간을 측정하였다.

측정 결과를 통하여 핵심코드 중계로 인한 지연시간은 1002ns, 인터럽트 중계로 인한 지연시간은 678 ns 임을 확인 하였다. 72MHz의 System Clock에서 실험하였으며, System Clock의 속도가 증가하면 지연시간은 감소한다.

표5. 핵심코드 중계 수행시간

라이브러 중계 사용여부	수행시간(μs)
미사용	1380
사용	378

표6. 인터럽트 중계 수행시간

ISR 중계 사용여부	수행시간(μs)
미사용	1350
사용	672

Ⅳ. 결 론

본 논문에서는 하드웨어 복사방지 보호 인증과 핵심코드를 보호하면서 타 업체에 펌웨어를 공개하는 Cortex-M3 기반 보안 플랫폼을 제안하였다.

제품생산 시, 하드웨어의 비휘발성 메모리에 인증코드를 Write하고, 펌웨어에서 하드웨어 인증하며, 하드웨어의 무단 복제를 방지하였다.

또한, 펌웨어를 보안코드 및 보안 중요도에 따라 분류하여, IAP와 APP의 2벌의 프로그램으로 나누어 핵심코드를 보호하면서, 외부업체가 코드를 일부 공개 및 수정이 가능하도록 하였다.

2벌로 나누어진 프로그램을 핵심코드 중계와 인터럽트 중계를 통하여 1벌의 프로그램처럼 동작하며, 핵심코드 중계는 1002ns, 인터럽트 중계는 678ns의 지연시간이 발생하는 단점을 확인하였다.

이번 연구에서는 코드를 공개 가능한 플랫폼의 구조 구현에 초점을 맞추어 연구를 하였다. 제품생산에서 사용되는 인증코드 Write Tool이 외부에 누출 되었을 때, 보안에 치명적이므로 인증코드 Write Tool의 이원화 방법에 대해서 추가 연구가 필요하다.

참고문헌

[1] 가상하드디스크를 이용한 펌웨어 보호 기법에 관한 연구, 심재범, 나중찬, 과학기술연학대학원대학교 2015

[2] 내부정보유출방지의 혁신적 도약을 위한 S/W 방식 DLP구축, 김승용, 선업혁신연구 (제26권 4호), 2010. 12. pp. 59~89