차세대 저궤도 위성의 비행소프트웨어 리프로그래밍

유범수, 정재엽, 최종욱 정회원

Flight Software Reprogramming for Next Generation LEO Satellites

Bum-Soo Yoo, Jae-Yeop Jeong, and Jong-Wook Choi *Regular Members*

요 약

위성의 임무 수행 도중에 발생하는 소프트웨어 버그는 치명적인 문제점을 야기하여 위성 전체의 임무 실패를 초래할 수도 있다. 이를 막기 위해 개발단계부터 수많은 테스트와 검증을 수행하여 비행소프트웨어가 높은 신뢰도를 지니도록 만들어 준다 [1]. 하지만 위성이 발사 후 궤도에 올라갔을 때 하드웨어 측면에서의 문제, 혹은 미 발견 버그 등의 예상치 못한 문제들이 발견될 수도 있다. 이 경우 비행소프트웨어를 궤도 상에서 수정해야만 위성이 지속적으로 임무를 수행할 수 있다. 본 논문에서는 저궤도 위성의 reprogramming capability를 확인하고 reprogramming 절차에 대해 알아보고 검증한다.

Key Words : Satellite Flight Software, LEO Satellite, Reprogramming

ABSTRACT

In satellites, even a small error in flight software could cause a failure of missions. Therefore, there are strict development and verification processes for a high reliability of flight software. However, satellites on orbits could meet unexpected situations including hardware malfunction. In this case, it is necessary for flight software to be updated to cope with the unexpected situations and to continue their missions. This paper reviews reprogramming capability of next generation LEO satellites.

Ⅰ. 서 론

위성의 임무 수행 도중에 발생하는 소프트웨어 버그는 위성 시스템에 치명적인 문제점을 야기하여 위성 전체의 임무 실패를 초래할 수 있다. 이를 방지하기 위해 개발, 시험, 발사 단계에서 까지 수많은 테스트와 검증을 수행하여 위성비행소프트웨어가 높은 신뢰도를 지니도록 한다[1]. 하지만 위성 발사 후 궤도에 올라갔을 때에도 소프트웨어 버그 및 하드웨어 측면에서의 문제들이 발견될 수도 있다. 이 경우 위성비행소프트웨어를 궤도상에서 수정해야만 위성이 지속적으로 임무수행을 할 수 있는 경우가 발생한다. 본 논문에서는 차세대 저궤도 위성의 위성탑재컴퓨터를 patch하기 위한 reprogramming capability를 확인하고, reprogramming 절차에 대해 알아보고 검증한다.

Ⅱ. 저궤도 위성비행소프트웨어 리프로그래밍

1. 위성비행소프트웨어 리프로그래밍 시 고려사항

저궤도 위성에서 reprogramming을 설계하기 위해서는 resource margin, parameter, linking, compiler & linker 4가지를 고려해야 한다.

– Resource Margin

일반적으로 reprogramming을 수행하면 기존의 code보다 patch 후의 code가 커지게 된다. 따라서 메모리 상에서 여유공간을 reserved로 잡아놓아야 추후 비행소프트웨어에서 문제가 생겼을 때 수정이 가능하다. 하지만 단순히 빈 곳을 reserved로 잡는 것이 아니라 .text, .data, .bss/COMMON과 같이 코드의 각 영역에 대해 margin을 할당해 주어야만 reprogramming이 가능하다. 추가로 code patch가 수행이 되면 새로운 telemetry를 추가해야 할 수도 있다. 이를 위해 telemetry를 추가할 수 있는 메모리 영역 또한 확보해둘 필요가 있다.

한국항공우주연구원, 위성기술연구단, 위성비행소프트웨어팀 ({bsyoo, jyjeong, jwchoi}@kari.re.kr
접수일자 : 2017년 9월 20일, 최종게재확정일자 : 2017년 9월 26일

– Parameter

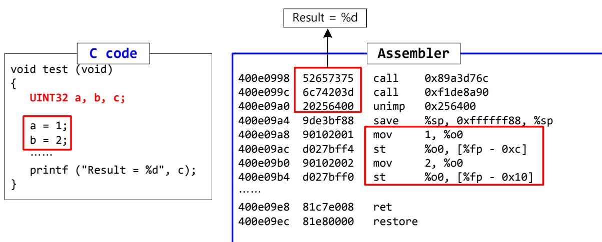
비행소프트웨어는 많은 table을 참조하고, table에 따라 제어가 된다. 이는 상황에 따라 table만 제어하여 비행소프트웨어의 동작을 조절하게 만들어 주어, 매번 제어를 위해 reprogramming하는 것을 막아준다. Table기반으로 제어 또한 memory 로 mapping 시켜놓을 경우, memory upload를 이용하여 쉽게 위성을 제어할 수 있다.

– Linking

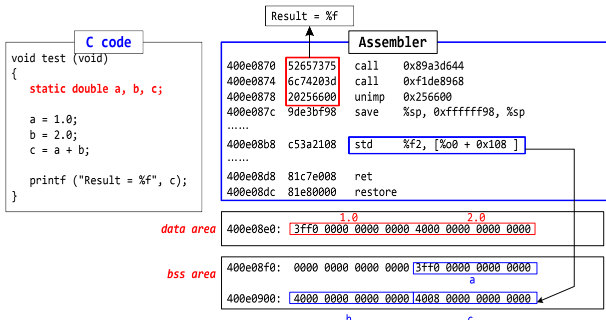
Linking에는 크게 static linking 및 dynamic linking이 있다. Static linking은 소프트웨어의 함수 및 변수들이 절대 주소를 가지게 생성이 되고, dynamic linking은 operation system (OS)에서 실시간으로 주소를 할당한다. 차세대 위성에서 사용하는 VxWorks의 경우 windows/linux와 달리 static linking만을 지원한다. 따라서 절대 주소를 가지게 되고 symbol table를 필요로 하지 않게 된다. 이 때 각각의 코드 영역마다 margin을 적절히 확보해 두어야만 추후 reprogramming이 용이해진다.

– Compiler & Linker

Code가 assembler로 변할 때 compiler 및 linker 에 따라 data들이 다른 곳으로 할당이 된다. 그림 1은 compiler에서 메모리를 할당한 그림이다. 보면 그림 1(a)에서는 string이 local data로 가고 integer 값은 assembler안으로 들어갔다. 반면 그림 1(b)에서는 string은 local 값으로 갔고 static double 은 ..bss 영역으로 갔다. 추가로 double에 대한 초기값들은 .data 로 할당이 되었다. 이처럼 메모리 할당이 다양하고 경우에 따라 다른 영역에 할당이 되기 때문에 reprogramming을 위해서는compiler 및 linker에서 어느 메모리 영역에 할당하는지 고려하여 설계한다.



(a) Local integer and string in assembler



(b)

그림 1.

2. 위성탑재컴퓨터의 메모리 구조

차세대 저궤도 위성에서는 SPARC v8 기반의 AT697F 프로세서를 이용하여 On Board Computer (OBC)를 구성한다 [2]. OBC는 크게 연산을 처리하는 PM697과 통신을 처리하는 TCTM으로 나눠지며 각각의 메모리 구조는 그림 2와 같다. PM697의 메모리는 bootROM이 올라가는 PROM, 위성 비행소프트웨어가 올라가는 non-Volatile Memory (NVMEM), 마지막으로 instruction이 수행이 되는 SRAM이 있다. NVMEM의 경우 A, B 2개가 있는데 2곳에 동일한 소프트웨어를 올려서 신뢰도를 높인다. PM697에 전원이 인가되면 우선 PROM을 읽어서booting sequence를 수행한다. Booting sequence의 마지막에는 NVMEM에 올라가 있는 비행 소프트웨어를 SRAM에 upload하고, SRAM에서 비행소프트웨어를 실행시킨다. TCTM의 메모리는 Safe Guard Memory (SGM)의 메모리는 목적에 맞게 나눠서 사용한다. 그 중에는 PM697 reprogramming을 위해 할당된 code patch area도 있다.

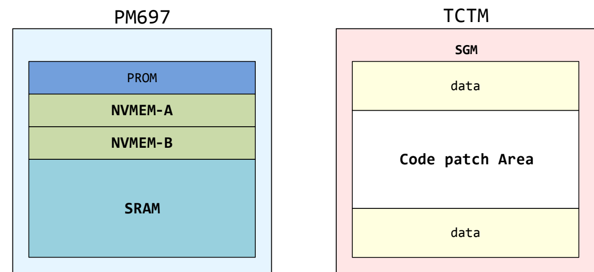


그림 2. 위성탑재컴퓨터의 메모리 구조

3. Reprogramming Scheme

저궤도 위성은 SRAM, SGM, NVMEM, 총 3가지의 reprogramming capability를 가지고 있다.

– SRAM Code Patch

SRAM은 비행소프트웨어가 돌아가는 메모리로, SRAM code patch는 해당 메모리에서 직접 소프트웨어를 수정하는 방법이다. SRAM code patch에는 2가지 방법이 있는데, 문제를 발생시키는 소프트웨어의 작은 부분을 수정하는 inline code patch method와 문제가 되는 부분 전체를 수정하는 jump logic return method가 있다 [3].

Inline code patch method는 patch 하고자 하는 code에다가 필요한 instruction값을 over writing하는 것이다. 그림 3은 inline code patch method의 대표적인 동작인 상수 값 patch를 보여준다. 상수 값을 10에서 20으로 바꾸고자 해당 code에 대한 instruction을 확인하니 "add %o0, 0xa, %o1"을 확인할 수 있었다. 그 후 "add %o0, 0x14, %o1"로 바꾸는 instruction인 0x92022014를 적어서 해당 메모리 주소인 0x400E09D0에 넣는다 [4]. 이 방법은 간편하지만 over

writing이기 때문에 patch 하는 instruction이 더 클 경우 사용하기 어렵다. 뿐만 아니라, compiler를 통하지 않고 assembler를 code hand-writing method로 작업해야 하기 때문에 작업 분량이 많을 경우 사용하기 어렵다.

CMD Type	ID	Length	Week	Address	SRAM data	CRC
04	0000	10	0000	400E09D0	92022014	0204

```
0x400e09d0 : 92022014 add %o0, 0x14, %o1
```

```
int func_a (int a, int b)
{
   int result;
   ......
   result = a + b + 10;
  0x400e09c4        +0x00c:    ld       [%fp + 0x44], %o0
  0x400e09c8        +0x010:    ld       [%fp + 0x48], %o1
  0x400e09cc        +0x014:    add      %o0, %o1, %o0
  0x400e09d0        +0x018:    add      %o0, 0xa, %o1
  0x400e09d4        +0x01c:    st       %o1, [%fp - 0xc]

   return (result);
}
```

그림 3. SRAM patch with inlink patch method

Jump logic return method는 patch할 instruction많아서 over writing으로는 처리가 불가능할 때, 해당 모듈 전체를 바꾸는 방법이다. 그림 4는 jump logic return method의 동작을 보여준다. function_a 전체를 수정하기 위해 function_b를 빈 메모리 영역에 upload 하는 code patch#1을 수행한다. 그 후 function_a에 inline code patch method를 통해 function_b로 jump 하도록 함수의 시작 부분의instruction을 수정하는 code patch#2를 수행한다. 이 때 function_b에서는 수정한 operation을 수행한 뒤, 다시 원래 function_a의 끝부분으로 return하여 임무를 계속 수행할 수 있도록 만든다. 이 방법은 함수 전체를 바꿀 수 있지만 기존의 code들의 메모리 주소가 변경되면 안 되기 때문에 code patch에 주의할 필요가 있다. 뿐만 아니라 VxWorks에서는 메모리 영역이 구분되기 때문에, 각각의 .text, .data, .bss&COMMON 영역마다 reserved 영역을 확보해야 한다. 마지막으로 위 2가지 SRAM patch를 하는데 있어서 patch 후에는 반드시 instruction cache를 flush 해주어야만 code patch의 동작을 보장할 수 있다.

SRAM patch는 비행소프트웨어가 동작하는 도중에 패치를 수행한다. 소프트웨어가 해당 instruction을 수행하는 도중에 code patch가 적용이 될 경우, 동작을 보장하기 힘들 뿐더러, 치명적인 오류를 발생시킬 수도 있다. 따라서 위성의 임무에 지장을 주지 않기 위해, SRAM patch command가 들어오면 SRAM patch buffer에 따로 저장을 한다. 그 후, patch하고자 하는 코드가 수행되지 않는 것이 보장되는 시점인 background task에서 patch를 수행한다 [5].

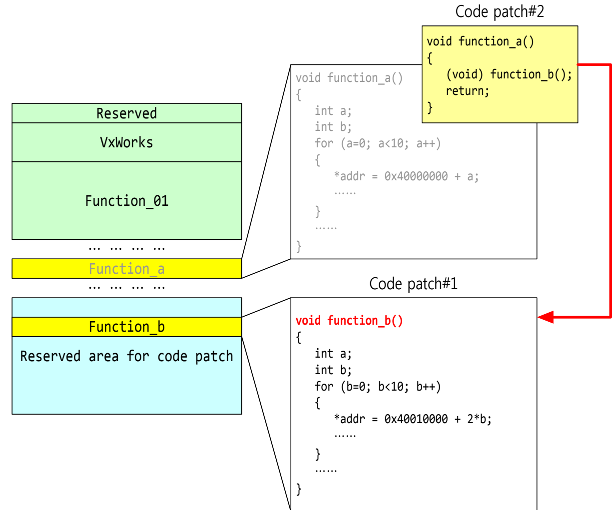


그림 4. SRAM patch with jump logic return method

– SRAM Code Patch

SGM은 PM697이 아닌 TCTM의 메모리에다가 패치 데이터를 넣어 놓고, 나중에 지상에서의 개입에 따라 SRAM code patch를 수행하는 것이다. SGM 패치를 SRAM과 비교하면, 많은 데이터를 한번에 올려놓고 패치를 수행할 수 있다. 그리고 특정 패치를 미리 올려놓았다가 필요하다고 판단이 될 때 지상의 개입에 따라 적용할 수 있기 때문에 좀더 유연하게, 반복적으로 쓸 수 있다. 위성의 reconfiguration 후 booting sequence를 보면, NVMEM에 있는 비행소프트웨어를 SRAM으로 매번 복사한다. 따라서 SRAM code patch는 reconfiguration후에 사라진다. 하지만 SGM의 data는 reconfiguration 이후에도 남아있기 때문에 지상의 개입을 통해 upload 없이 code patch 적용이 가능하다.

SGM code patch는 SGM patch table에 의해 관리가 된다. 그림 5는 SGM patch table및 data의 구조로, data에는 code patch data가 저장이 되고 해당 code patch를 patch table을 통해 관리한다. Patch table은 각각의 patch에 8byte씩 할당되어 number (2 bytes), address (2 bytes), size (2 bytes), CRC (2 bytes) 값이 저장이 된다. Address는 code patch 내용이 어디에 있는지 code patch data area의 특정영역을 알려주고, size는 해당 영역에서의 data 크기를, CRC는 오류 검사 값을 나타낸다. Patch table 앞에는 header와 몇 개의 patch 가 들어있는지 count 값이 있다. SGM memory upload command가 들어오면 해당 data를 SGM code patch data area에 저장하고 SRAM과 SGM의 patch table을 업데이트 한다. 위 과정을 반복하며 최대 447개까지의 SGM code patch를 저장한다. 추후 특정 patch number를 적용하라는 지상 command가 들어올 경우, SGM patch data에서 값을 읽어와서 patch table에 있는 CRC를 이용하여 오류 검사를 수행한다. 오류가 없을 경우 patch data를 SRAM cod patch 방식을 이용하여 적용한다. Reconfiguration시 SRAM의 patch table이 없어지지만, booting 후 초기화 과정에서 다시

SGM의 patch table 값을 복사해오기 때문에 SRAM 의 patch table이 없어져도 문제가 되지 않는다.

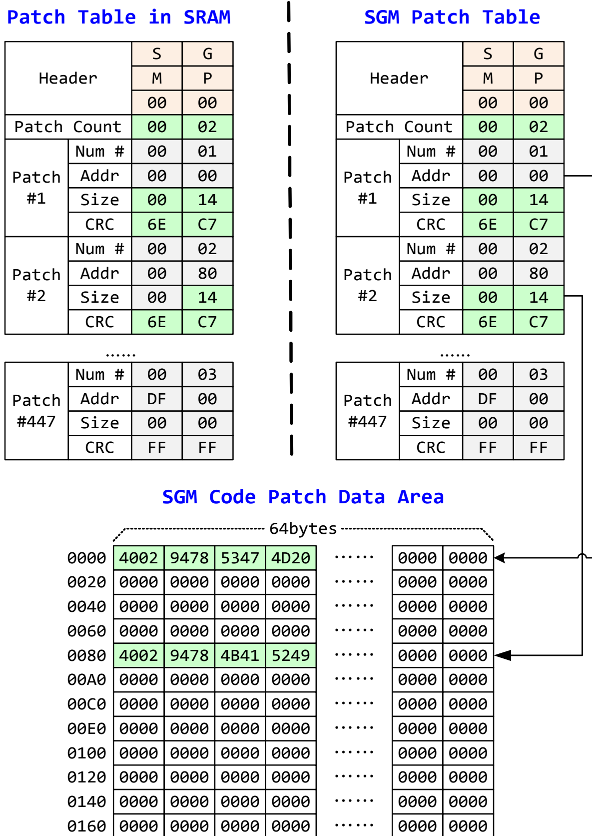


그림 5. SGM patch table의 구조

– NVMEM Code Patch

위성이 reconfiguration을 수행하면 기 수행했던 SRAM code patch는 전부 사라진다. 따라서 필요할 경우 다시 data upload부터 전부 다시 수행해야 한다. SGM code patch의 경우 SRAM과 달리 data upload를 할 필요는 없지만, 지상의 개입을 통해 전부 다시 다 적용을 해야 한다. 이는 위성에 문제가 생겼을 때 수행하는 몇몇 critical path 에서는 사용이 불가능하다. 이를 해결하기 위해 NVMEM에서 탑재되어 있는 비행소프트웨어를 수정하는 NVMEM reprogramming 기능이 필요하다. Code patch한 NVMEM은 booting sequence에서 SRAM으로 복사가 되어 reconfiguration 이후에도 계속 적용이 된다.

그림 6는 NVMEM reprogramming을 보여준다. 지상에서 미리 지정해 놓은 SRAM 영역에 data를 upload 한다. 그 후 NVMEM reprogramming command를 보낼 경우, 256 byte씩 over writing을 수행하여 NVMEM을 code patch한다. NVMEM의 끝에는 NVMEM 에 대한 length와 CRC 정보가 들어가 있다. 따라서 code patch 후에는 맞는 CRC와 length 값으로 수정해야 추후 오류 검사를 통과할 수 있다. PM697에는 NVMEM A, B, 2개가 존재한다. 2개를 동시에 patch할 경우, code patch를 잘못 수행하면 완전히 망가져 버린다. 따라서 NVMEM 1개만 code patch하여 동작 여부를 확인 한 뒤, 나머지 NVMEM 1개에도 code patch를 수행한다.

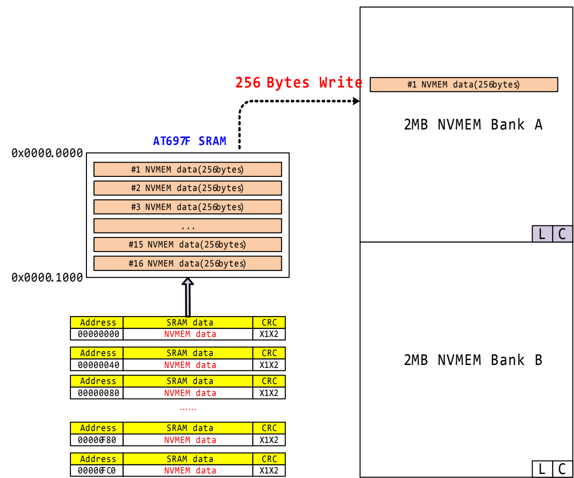


그림 6. NVMEM patch diagram

Ⅲ. Experiment

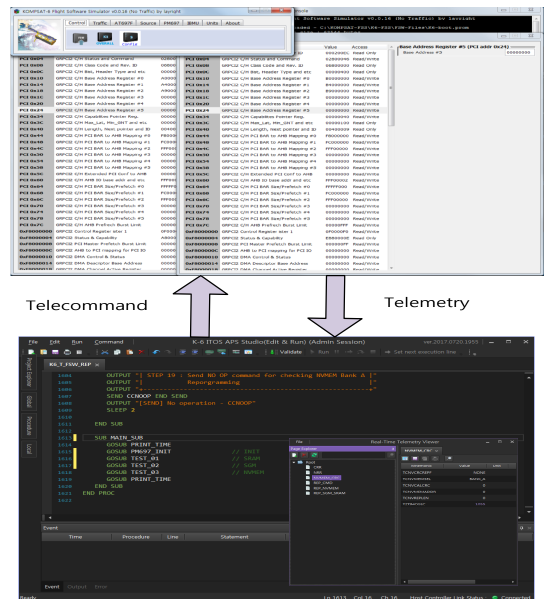


그림 7. 위성 시뮬레이터와 ITOS를 이용한 Reprogramming Experiment

위의 3가지 code patch 방법은 위성의 동작을 모사해주는 프로세서simulator와 지상지원 장비를 모사하여 telemetry를 받고 telecommand를 보내는 지상지원장비 simulator를 이용하여 test 하여 검증하였다[6].

Ⅳ. 결론

본 연구에서는 저궤도 위성의 임무 수행에 있어서

reprogramming 기능에 대해 확인 및 검증을 하였다. 위성의 비행소프트웨어는 다른 embedded시스템과 달리 시스템을 정지시킨 뒤 수정을 할 수가 없고 개발자가 직접 시스템에 가서 수정을 할 수도 없다. 대신 본 논문에서 다루고 있는 reprogramming 기능을 이용하여 SRAM, SGM, NVMEM code patch를 수행하여, 우주에서의 예상치 못한 상황에 대응할 수 있도록 만든다. 이를 통해 소프트웨어의 신뢰도를 높여서 위성이 성공적으로 임무를 수행할 수 있도록 만들어 줄 것이다.

참 고 문 헌

[1] B.-S. Yoo, “Verification of flight software with unit test,” in Proc. Avionics Systems Symp. Korea, Jul. 2017.

[2] J.-W. Choi and Y.-J. Cheon, “Study of Next Space Processor for Development of Flight Software,” in Proc. Conf. Korean Society of Aeronautical & Space Science, Nov, 2012.

[3] J.-W. Choi et al., “On-Orbit Flight Software Reprogramming Scheme for Next Generation LEO Satellites,” in Proc. Conf. Korean Society of Aeronautical & Space Science, Dec. 2009.

[4] SPARC V8 Architecture manual, Gaisler, 1991.

[5] J.-W. Choi, et al., “Design and Development of PCI-based 1553B communication Software for Next Generation LEO On-Board Computer,” J. Satellite, Info, and Comm., vol. 11, no. 3, Sep. 2016, pp. 65-71.

[6] J.-W. Choi, “Development of Space Processor Emulator/Simulator for Flight software Development,” in Proc. Avionics Systems Symp. Korea, Jul. 2017.

저자

유 범 수(Bum-Su Yoo)

・2009년 2월 : 한국과학기술원 전기 및 전자공학 학사졸업
・2011년 2월 : 한국과학기술원 전기 및 전자공학 석사졸업
・2016년 2월 : 한국과학기술원 전기 및 전자공학 박사졸업
・2015년 12월 ~ 현재 : 한국항공우주연구원 위성탑재소프트웨어팀 선임연구원

<관심분야> : 임베디드시스템, 로보틱스

정 재 엽(Jae-Yeop Jeong)

・2007년 2월 : 충남대학교 컴퓨터공학 학사졸업
・2009년 2월 : 충남대학교 컴퓨터공학 석사졸업
・2009년 1월 ~ 2013년 12월 : LIG넥스원 항공연구센터 선임연구원
・2014년 1월 ~ 현재 : 한국항공우주연구원 위성탑재소프트웨어팀 선임연구원

<관심분야> : 임베디드시스템, 실시간운영체제

최 종 욱(Jong-Wook Choi)

・1999년 2월 : 경북대학교 전자공학 학사졸업
・2001년 2월 : 경북대학교 전자공학 석사졸업
・2016년 2월 : 충남대학교 컴퓨터공학과 박사졸업
・2000년 12월 ~ 현재 : 한국항공우주연구원 위성탑재소프트웨어팀 선임연구원

<관심분야> : 시뮬레이터, 실시간운영체제