차세대 위성탑재컴퓨터를 위한 프로세서 모니터 및 고장주입 시스템의 설계 및 구현

정재엽* 정회원, 최종욱*, 천이진**

Design and Implementation of a Processor Monitor and Fault Injection System for Next Generation Spacecraft Computer Board

Jae-Yeop Jeong*, Jong-Wook Choi*, Yee-Jin Cheon**

요 약

위성탑재컴퓨터의 정상동작을 검증하기 위해 프로세서의 모니터링 및 디버깅은 필수적이며, 현재 Aeroflex Gaisler의 GRMON을 사용하고 있다. GRMON은 LEON 프로세서를 모니터링 및 디버깅하기 위한 다양한 기능을 제공하지만, 국내에서 제작한 위성탑재컴퓨터에 사용할 수 없는 기능이 많기 때문에 가격 대비 성능이 낮다. 또한 LEON 프로세서의 DSU를 이용하면 모든 메모리맵에 접근이 가능하여 프로그램 실행 중 고장을 주입할 수 있음에도 불구하고, GRMON을 수정할 수 없기 때문에 그동안 위성탑재소프트웨어를 수정하여 하드웨어에 고장을 주입하는 방식을 사용하였다. 이런 고장주입 방법은 위성탑재소프트웨어의 형상을 변경하는 것이므로 고장에 따른 소프트웨어의 영향성을 정확히 판단할 수 없다. 이에 향후 저궤도 관측위성에 적용될 LEON2FT AT697F프로세서를 탑재한 차세대 위성탑재컴퓨터(NGSCB, Next Generation Spacecraft Control Board)에서 프로세서 디버깅을 위한 기본 기능을 제공하고, 실제 위성에 탑재되는 위성탑재소프트웨어와 동일한 환경에서 하드웨어 고장을 주입할 수 있는 시스템을 설계 및 구현하였다.

Key Words : LEON2FT, AT697F, NGSCB, Processor Monitor, Fault Injection

ABSTRACT

In order to verify normal operation of satellite OBC(On Board Computer), it is essential that processor monitoring and debugging. So we are using the GRMON of Aeroflex Gaisler. It provides a lot of features for debugging of LEON processor but we can't use that features on the NGSCB(Next Generation Spacecraft Computer Board) except a few things. So the cost-effectiveness is very low. And for hardware fault injection, we are using a method of modify satellite flight software, because we can't modify GRMON. This method can not guarantee normal operation of the satellite flight software. So in this paper we were developed the processor monitoring and fault injection tool for NGSCB.

I. 서 론

현재까지 국내에서 개발된 저궤도 관측위성의 탑재컴퓨터는 Intel과 ESA(European Space Agency)에서 개발 된 프로세서를 사용하였다. Intel사의 80186/80386을 시작으로 최근까지는 ESA의 SPARC v7 기반의 ERC32 프로세서를 확장한 MCMERC32를 사용하였으며, 향후에는 SPARC v8기반 LEON2FT 프로세서인 AT697F가 탑재될 예정이다[1][2].

위성탑재소프트웨어의 개발 및 검증을 위해서 실시간으로 프로세서를 모니터링하고 디버깅 할 수 있는 기능이 필수적이며, 탑재소프트웨어를 수정하지 않고 외부에서 고장주입(Fault Injection)이 가능하여야 한다. 이를 위해 기존에는 Aeroflex Gaisler에서 개발 된 GRMON[5][6]을 사용하였으나, GRMON에서 제공되는 기능 중 국내에서 제작한 위성탑재컴퓨터에 적용할 수 있는 기능은 매우 제한적이었으며, LEON2FT 프로세서를 지원하는 GRMON을 더 이상 개발하지 않는 문제점이 있다.

이러한 문제점을 해결하기 위해 LEON2FT AT697F 프로

*한국항공우주연구원 위성탑재소프트웨어팀 jyjeong@kari.re.kr *한국항공우주연구원 위성탑재소프트웨어팀 jwchoi@kari.re.kr
**한국항공우주연구원 위성탑재소프트웨어팀 yjcheon@kari.re.kr
접수일자 : 2014년 11월 17일, 수정완료일자 : 2014년 12월 19일, 최종게재확정일자 :2014년 12월 19일

세서를 탑재한 차세대 위성탑재컴퓨터(이하 NGSCB, Next Generation Spacecraft Control Board)에서 프로세서 디버깅을 위해 필요한 기본 기능을 제공하고, 실제 위성에 탑재되는 위성탑재소프트웨어와 동일한 환경에서 하드웨어 고장을 주입할 수 있는 시스템을 설계 및 구현하였다. NGSCB 모니터 및 고장주입 기능의 확장성을 위해 LUA[9][10] 스크립트를 적용하여 프로세서 모니터 프로그램의 수정 없이 다양한 기능을 스크립트로 실행할 수 있게 하였다. 이를 통해 NGSCB를 실시간으로 모니터링 및 디버깅 할 수 있으며, 특히 위성탑재소프트웨어를 변경하지 않고 다양한 고장주입을 통해 위성탑재소프트웨어의 FDIR(Fault Detection, Isolation and Recovery) 매커니즘을 검증할 수 있다.

Ⅱ. 관련 기술 분석

1. 차세대 위성탑재컴퓨터 (NGSCB)

NGSCB는 향후 개발 예정인 저궤도 관측위성의 위성탑재컴퓨터에 적용하기 위해 시험용으로 개발되었으며, ESA LEON2FT 기반의 AT697F가 탑재된다. AT697F를 기반으로 memory bus에는 EEPROM, SRAM, Aux FPGA가 연결되고, 외부 통신을 담당하는 2개의 FPGA는 PCI Bus로 연결되어 있다. 단일 칩의 단종에 따른 위험성을 최소화하기 위해 다양한 디바이스를 IP(Intellectual Property)로 개발하여 FPGA에 구현하였으며, 2개의 FPGA에는 다양한 미션을 수행할 수 있도록 1553B, SpaceWire, GRCAN, DMAUART, AHBUART, APBUART, GPIO등이 FPGA IP Core로 구현된다. 또한 FPGA에서 발생하는 인터럽트를 처리하는 IRQMP와 AHB Bus의 상태를 모니터링 하는 AHBSTAT, AHBTRACE가 구현되어 있다. 각 FPGA내의 IP Core는 AMBA AHB/APB bus로 연결되며, AT697F와 PCI 통신을 할 수 있도록 GRPCI2가 탑재된다[2][4].

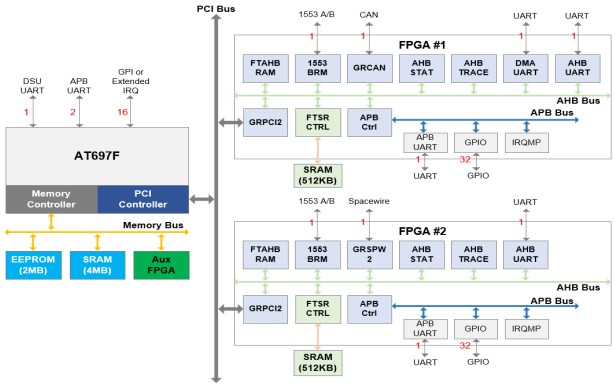

그림 1. NGSCB Block Diagram

2. AT697F DSU (Debug Support Unit)

AT697F는[7] 하드웨어 디버깅을 위해 DSU(Debug Support Unit)과 DCL(Debug Communication Link)을 지원한다. DSU는 프로세서를 error/normal/debug mode로 설정할 수 있으며, 모든 프로세서 register 및 cache memory에 read/write access를 가능하게 한다. 또한 DSU는 8KB(512 line, 1line = 16byte) 크기의 trace buffer를 지원하여 실행된 instruction이나 internal bus에 전송되는 data를 저장하여 문제가 발생했을 경우 확인할 수 있는 기능을 제공한다.

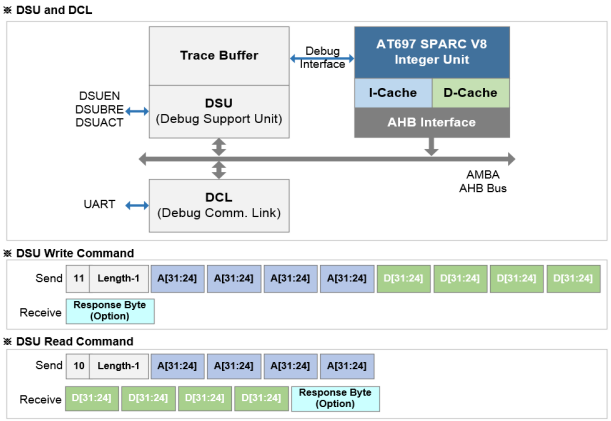

그림 2. DSU Block Diagram and Command Protocol

AT697F DSU는 DCL을 통해 외부 AHBUART와 연결되며, DCL이 지원하는 read/write command protocol을 이용하여 외부에서 하드웨어에 접근이 가능하다. write command는 MSB 2bit를 11b로 설정해야 하며, 한 번에 최대 2^6개의 word를 전송할 수 있다. read command는 MSB 2bit를 10b로 설정해야 하며, 한 번에 최대 2^6개의 word를 수신할 수 있다. 또한 DSU control register를 설정하여 response byte를 선택적으로 수신할 수 있다.

표 1. DSU Memory Map

Address	Register
0x90000000	DSU control register
0x90000004	Trace buffer control register
0x90000008	Time tag counter
0x90000010-0x9000001C	AHB registers
0x90010000-0x9001FFFC	Trace buffer
0x90020000-0x9003FFFC	IU/FPU register file
0x90080000-0x900FFFFC	IU special purpose register
0x90080000	Y register
0x90080004	PSR register
0x90080008	WIN register
0x9008000C	TBR register
0x90080010, 0x90080014	PC, nPC register
0x90080018	FSR register
0x9008001C	DSU trap register
0x90080040	ASR16
0x90080060-0x9008007C	ASR24-ASR31
0x90100000-0x9017FFFC	Instruction cache tags & data
0x90180000-0x901FFFFC	Data cache tags & data

AT697F의 레지스터 및 cache memory를 접근하기 위해서는 표 1의 memory map에서 해당 영역을 통해 접근할 수 있으며, RAM, FPGA등과 같은 외부 디바이스는 전체 system memory map에서 해당 디바이스의 영역을 바로 접근하여 제어가 가능하다.

3. GRMON

GRMON은[5][6] Aeroflex Gaisler에서 개발한 LEON 프로세서용 Debug Monitor로써, LEON2/3 프로세서를 지원하는 GRMON과 LEON3 이후 프로세서를 지원하는 GRMON2가 있다. GRMON은 Target에 debug protocol이 구현되어 있는 하드웨어가 AHB Master로 동작할 경우 연결이 가능하며, Serial, Ethernet, JTAG, USB, PCI, SpW와 같은 인터페이스를 지원한다. GRMON은 Target의 모든 레지스터 및 메모리를 접근하여 모니터링 및 디버깅을 위한 다양한 기능을 제공한다. disassembler와 trace buffer 관리, 응용프로그램 다운로드 및 실행, break point, watch point, GDB(GNU debugger) 등의 기능을 제공한다.

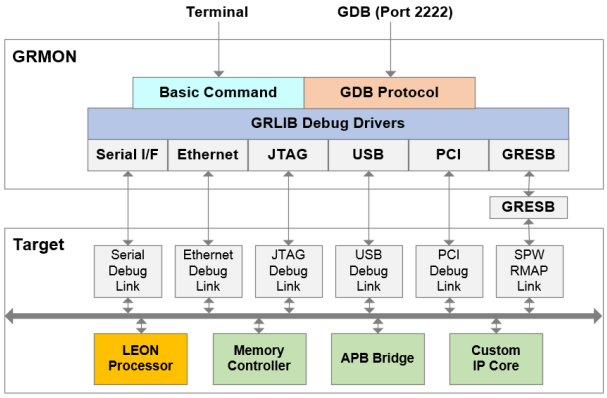

그림 3. GRMON overview

4. LUA Script Language

고급언어로 작성된 프로그램은 컴파일을 하거나 인터프리터를 이용하여 실행시킬 수 있다. 컴파일러는 고급 명령어들을 직접 기계어로 변환하지만, 인터프리터는 고급 명령어를 중간 형태로 번역한 다음 그것을 바로 실행한다. 컴파일된 프로그램은 일반적으로 인터프리터를 이용하여 실행하는 것보다 속도가 빠르지만, 프로그램의 수정이 발행할 때마다 컴파일을 다시 수행해야하는 단점이 있다. 이에 비해 인터프리터 방식은 컴파일 과정이 필요 없이 바로 실행이 가능하다. 이런 이유로 인터프리터는 종종 프로그램 개발단계에서 테스트를 목적으로 사용된다.

LUA는[9][10] 인터프리터 언어중의 하나로 직접적으로 인터프리트 되지 않고, 바이트 코드로 컴파일 되어 LUA 가상머신에서 실행된다. 다른 대부분의 가상머신이 스택 기반인 것과 달리 LUA 가상머신은 CPU의 구조와 유사한 레지스터 기반이라서 실제 하드웨어 디자인과 유사성이 있다. 레지스터 기반의 가상머신은 값을 과다하게 복사하는 것을 방지할 수 있고, 함수를 구성하는 명령어를 줄일 수 있는 것이 장점이다. LUA는 스크립트 처리 속도가 빠르며, 변환 후 파일 용량의 증가가 크지 않다. 또한 ANSI 표준을 지키므로 주로 C/C++에 기반을 둔 프로그램 안에 LUA 코드를 라이브러리화해서 포함할 수 있다. C/C++과 연동 시 LUA STACK을 이용하여 LUA와 C/C++간 함수 호출 및 인자를 전달할 수 있다.

Ⅲ. 프로세서 모니터 및 고장주입 시스템 설계 및 구현

GRMON을 사용할 경우 일반적인 LEON 프로세서에 대한 디버깅은 가능하나, 특정 프로세서의 추가적인 기능이나 NGSCB와 같이 PCI를 이용하여 Device를 확장할 경우 이에 대한 디버깅 및 모니터링이 어렵다. 또한 DSU의 기능을 이용함에도 불구하고, GRMON을 수정할 수 없으므로 실시간으로 고장을 주입할 수 없다. 그러므로 향후 개발 예정인 저궤도, 정지궤도 위성탑재컴퓨터를 위한 프로세서 모니터 및 고장주입 시스템의 개발은 필수적이다.

본 논문에서 개발한 프로세서 모니터 및 고장주입 시스템(이하 LEONMON)은 AT697F의 AHBUART를 이용하여 DSU에 접근하며, 이를 이용하여 프로세서의 기본적인 디버깅을 위한 기능을 제공한다. 또한 LUA 라이브러리를 LEONMON에 탑재하여 LUA를 이용함으로써 NGSCB의 상태 모니터링 및 고장 주입을 프로그램 수정 없이 가능하게 하였다. LEONMON은 그림 4와 같이 Monitor Manager를 중심으로 LEONMON Command Parser, Instruction Disassembler, Execution File Loader, LUA Script Parser, LUA Script Loader, DSU Communicator, Logger로 구성된다. 사용자는 User Console을 통해 LEONMON을 제어할 수 있으며, LEONMON과 Target은 UART를 이용하여 연결된다. Target에서 실행되는 응용프로그램의 결과는 NGSCB의 APBUART를 통해 확인할 수 있다.

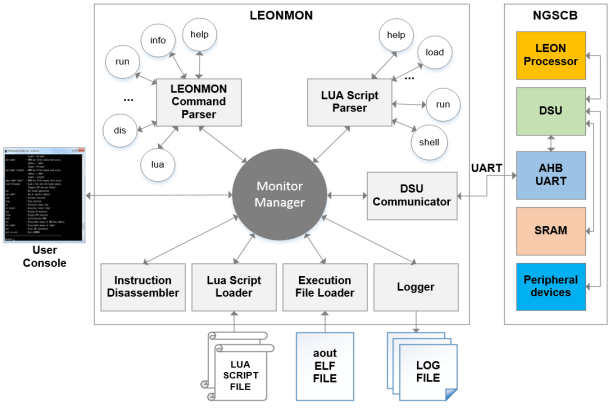

그림 4. LEONMON Architecture

1. DSU Communicator

Host UART를 이용하여 Target의 AHBUART와 연결되며 DSU Communication Protocol을 이용하여 최대 64 word 단위로 데이터 송수신이 가능하다.

2. Logger

LEONMON start option에서 *-log <filename>*을 이용할 경우 file pointer를 이용하여 사용자가 지정한 <filename>에 user console에 출력되는 모든 내용을 저장한다.

3. Monitor Manager

Monitor Manager는 LEONMON의 핵심으로 프로그램 실행 시 start option을 이용하여 시스템의 상태를 설정하고, AHBUART를 통해 DSU와 연결을 시도하여 연결이 완료되면 LEONMON 시스템 초기화를 수행한다. 또한 사용자로부터 명령을 입력받아 LEONMON Command Parser, LUA Script Parser에 전달하며, 각 Parser로부터 명령을 수신하여 Instruction Disassembler, Lua Script Loader, Execution File Loader, Logger, DSU Communicator에 명령을 전달한다.

3.1 LEONMON 초기화

프로그램이 실행되면 먼저 사용자가 입력한 start option을 분석하여 시스템에 적용한다. 적용이 완료되면 Host UART를 설정하고, Target과의 연결을 시도하며, 연결이 되지 않으면 프로그램을 종료하고, 연결이 정상적으로 완료될 경우에는 LEONMON 시스템 초기화를 수행한다.

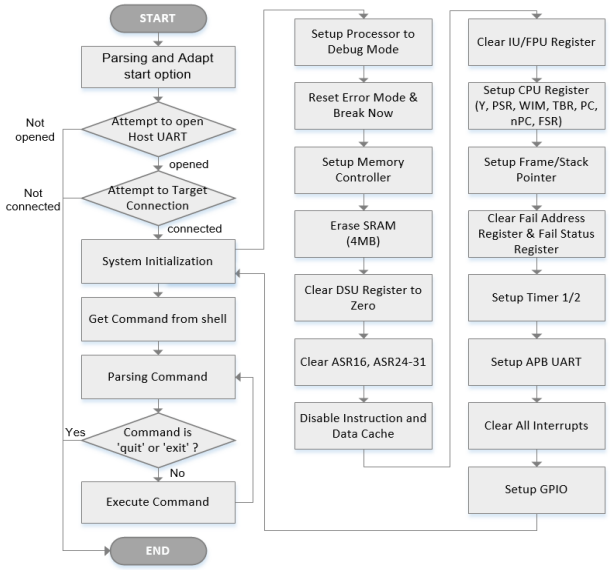

그림 5. LEONMON Initialization

LEONMON 시스템 초기화는 LEON2FT 프로세서를 Debug Mode로 설정하면서 시작한다. 그 후 SRAM의 정상 동작을 위해 memory controller register를 설정하고, EDAC(Error Detection And Correction) 기능을 활성화하기 위해 SRAM 영역을 0으로 초기화 한다. 메모리 초기화가 완료되면 DSU의 register를 0으로 초기화 하고, cache의 오동작을 방지하기 위해 instruction/data cache를 disable한다. 그 후 CPU와 관련된 Y, PSR, WIM, TBR, PC, nPC, FSR, FP, SP, Fail register를 설정하고, 모든 Interrupt를 초기화한다. 마지막으로 Peripheral인 Timer, APBUART, GPIO를 설정함으로써 LEONMON 시스템 초기화가 종료된다. 초기화가 완료된 후에는 사용자로부터 명령을 입력받아 해당 명령을 수행하며, 'quit' 또는 'exit'명령을 입력하면 LEONMON 프로그램이 종료된다.

3.2 Start Options

LEONMON 실행 시 사용자는 표 2의 option을 이용하여 시스템을 설정할 수 있다. Target에 대한 설정으로 NGSCB AT697F의 AHBUART에 연결할 경우 '-ngscb'를 이용하고, FPGA의 AHBUART에 연결할 경우 '-ngscb_fpga'를 이용한다. 향후에는 개발 예정인 Target에 대해 확장할 수 있는 option을 제공할 예정이다. 또한 Host 및 Target의 UART를 설정할 수 있으며, '-log'를 이용하면 LEONMON에서 사용자가 실행한 모든 정보를 파일로 저장한다.

표 2. LEONMON Start Options

Options	Description
-ngscb	Target을 NGSCB로 설정
-ngscb_fpga	Target을 NGSCB FPGA로 설정
-uart <comport>	Host UART comport 설정
-baud <baudrate>	Host UART baudrate 설정 (Default: 115200bps)
-auart <baudrate>	Target APBUART baudrate 설정 (Default: 115200bps)
-log <filename>	User Console에 출력되는 모든 내용을 <filename>에 저장

3.3 DSU Connection

DSU Connection을 위해 start option에서 적용한 값을 이용하여 Host UART를 설정한 후 Target의 AHBUART와 연결을 시도한다. DSU는 데이터 사이의 falling edge 차이를 계산하여 baudrate를 자동으로 설정하는 기능을 제공하며, Host UART를 이용하여 0x55값을 두 번 전송하면 Target의 baudrate가 Host에 맞도록 설정된다. Host와 Target의 연결 여부는 DSU UART Scaler 값의 정상여부를 판단하여 확인할 수 있다.

4. LEONMON Command Processing

4.1 LEONMON Command Parser

LEONMON의 초기화가 정상적으로 완료되면 user

console에 'leonmon>' prompt가 출력되고, 사용자는 표 3의 command를 이용하여 시스템을 운용할 수 있다.

LEONMON Command Parser는 Monitor Manager로부터 전달받은 command를 분석하여 실행하며, 필요한 정보를 user console에 출력한다. LEONMON Command Parser는 command 처리를 위해 Monitor Manager를 통하여 DSU Communicator, Instruction Disassembler, Execution File Loader, Logger를 호출한다.

표 3. LEONMON Commands

Command	Description
help	LEONMON command list 출력
info sys	AMBA Bus에 연결된 Device 정보 출력
info reg	System Register 정보 출력
info drv	ESA, Gaisler Device 정보 출력
wmem	Memory Write
mem	Memory Read
load	프로그램 Load
run	Load된 프로그램 실행
st	Instruction 순차 실행
stop, cont	프로세서 실행 중단, 프로세서 실행 재개
dis	Instruction disassemble
wash	SRAM memory 초기화
reg	IU register 정보 출력
float	FPU register 정보 출력
report, check	프로세서 상태정보 출력
lua	Lua Shell 실행
quit, exit	LEONMON 프로그램 종료

4.2 Instruction Disassembler

Instruction Disassembler는 *st, dis, reg, report, check* command에서 Monitor Manager를 통해 호출된다. 각 command에서 32bit의 instruction을 전달하면 SPARC V8 Instruction Set으로 변환하여 [address, hex value, instruction set] 양식으로 user console에 출력한다.

4.3 Execution File Loader

Execution File Loader는 *load* command를 실행하면 Monitor Manager를 통해 호출된다. 현재 LEONMON에서 지원하는 실행파일은 ELF와 aout 형식이며, 파일 형식이 정확할 경우에만 DSU를 이용하여 64 word 단위로 다운로드를 수행한다. SRAM에 다운로드가 정상적으로 완료되면 PC에 Entry Point address를 설정하여 실행 준비를 완료한다.

5. LUA Command Processing

5.1 LUA Command Parser

LEONMON에 사용한 LUA는 v5.2.3으로 LUA를 빌드하여 라이브러리를 생성한 후 LEONMON과 함께 빌드하여 사용이 가능하다. LUA Script에서 LEONMON 함수를 호출하기 위해 LUA shell 실행 시 lua_register() API를 이용하여 LUA 가상머신에 LEONMON API를 등록해야 한다. LEONMON은 C로 작성되었으며 LUA의 통신을 위해 LUA Stack을 사용한다. LUA Stack은 LUA에서 제공하는 LUA Stack 운용 API를 이용하여 제어가 가능하다. LEONMON shell에서 *lua* command를 실행하면 user console에 'lua>' prompt가 출력되며 표 4의 command를 이용하여 시스템을 운용할 수 있다. LUA Command Parser는 Monitor Manager로부터 전달받은 LUA command를 분석하여 실행하며, script를 로드하기 위해 LUA Script Loader를 호출하고, 사용자에게 필요한 정보를 user console에 출력한다.

표 4. LUA Commands

Command	Description
help	LUA command list 출력
load	LUA Script load
unload	LUA Script unload
run	LUA Script 실행
shell	LEONMON command 호출
quit, exit	LUA shell 종료

5.2 LUA Script Loader

LUA Script Loader는 script file을 LUA 가상머신에 load/unload하는 역할을 수행한다. LUA Script file을 load할 경우 luaL_loadfile() API를 이용하여 파일을 load하며, lua_pcall()을 이용하여 script의 global variable을 생성하여 사용자가 탐색이 가능하도록 설정한다. script를 unload할 경우 lua_close()를 호출하여 Lua state variable을 close한다.

Ⅳ. 시험결과 분석 및 고찰

LEONMON Command 시험 결과는 그림 7과 같으며, GRMON과 비교하였을 때 각 Command가 정상적으로 수행됨을 확인하였다.

고장주입시스템의 시험은 위성탑재소프트웨어 수정 없이 시험이 불가능 했던 Single Bit EDAC Error를 예제로 설명한다. 먼저 위성탑재소프트웨어와 유사한 환경을 만들기 위해 NGSCB에서 VxWorks를 실행한 후 task형태로 모의 위성탑재소프트웨어를 실행시킨다. LEONMON을 이용하여 NGSCB의 SRAM을 초기화 하고, VxWorks를 SRAM에 로드한 뒤 실행한다. Host에서는 Tornado를 VxWorks와 연결하여 모의 위성탑재소프트웨어를 다운로드 하여 실행한다. 모의 위성탑재소프트웨어는 SRAM 영역에 대해 Scrubbing을 수행하여 EDAC Error를 탐지할 수 있으며, EDAC Error

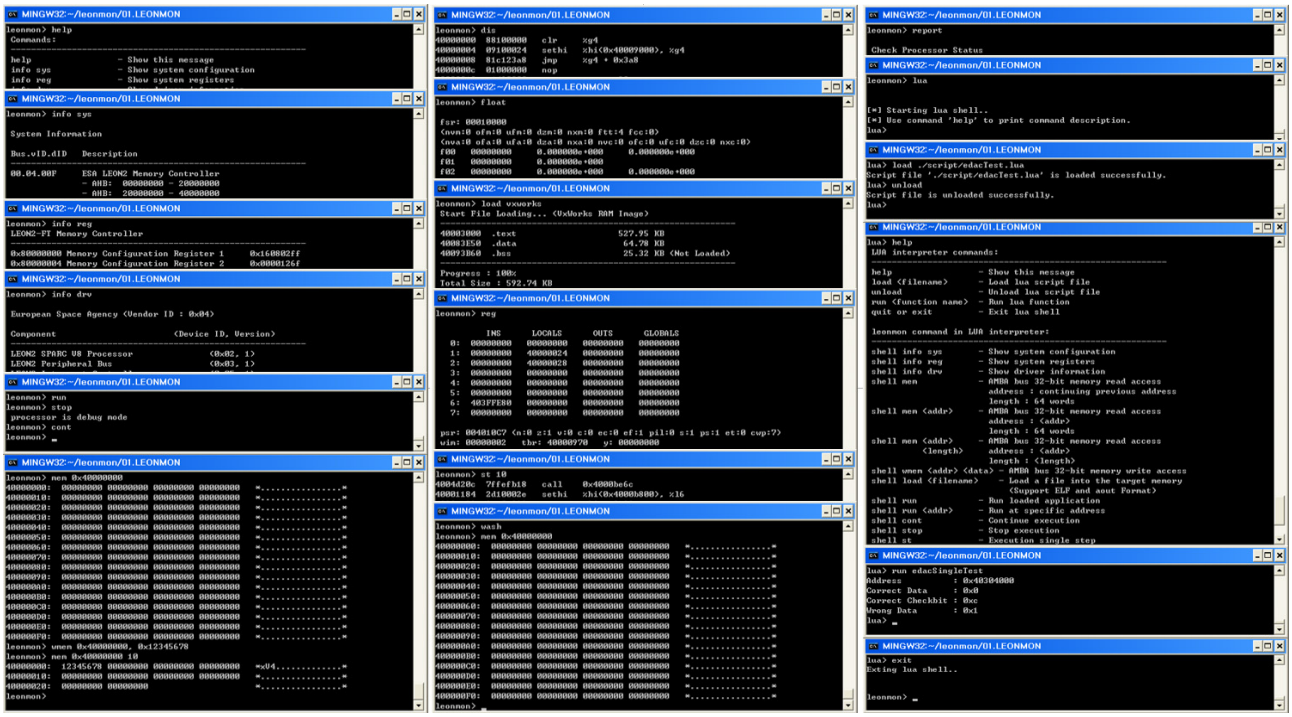

그림 7. LEONMON Commands 시험 결과

가 발생했을 경우에 VxWorks에 연결된 ISR이 실행된다. ISR에서는 Interrupt를 Disable하고 Single Bit EDAC Error Correction을 수행하며, Cache로 인한 ISR문제를 해결하기 위해 Data Cache를 Flush한다.

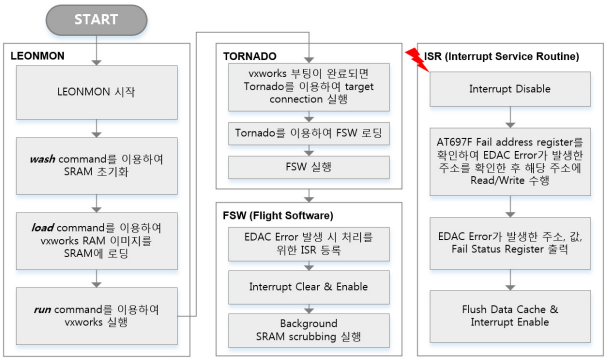

그림 6 Execute Simulated FSW on NGSCB

NGSCB에서 모의 위성탑재소프트웨어의 실행이 완료되면 LUA Script를 이용하여 고장을 주입한다. 먼저 LEONMON에서 *lua* command를 이용하여 LUA Shell을 실행시키고, *load* command로 LUA Script를 로드한다. 로드가 완료되면 *run* command를 이용하여 script를 실행한다. 사용자는 Single Bit EDAC Error를 발생시키려는 주소로부터 데이터를 read하여 데이터 또는 Check Bit의 값을 임의로 1bit 변경함으로써 Error를 발생 시킬 수 있다. AT697F MCFG3의 Diagnostic write bypass bit를 1로 설정하고, 데이터와 Check Bit를 설정한 후 다시 0으로 설정하여 Single Bit EDAC Error를 발생시킨다.

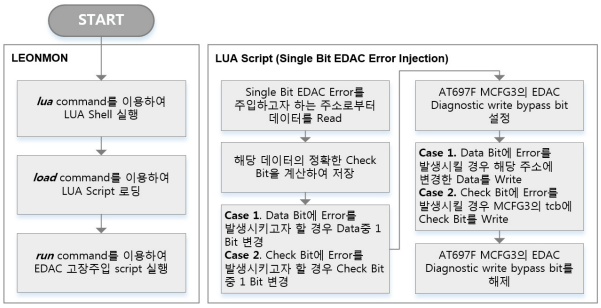

그림 8. EDAC Fault Injection using LUA Script

LUA Script를 이용하여 Single Bit EDAC Error를 주입하면 NGSCB에서 실행되는 모의 위성탑재소프트웨어가 해당 주소를 read할 때 EDAC Error를 탐지할 수 있다. EDAC Error는 AT697F Hardware Error(Trap Type: 0x11)에 속하며 해당 Trap 발생 시 VxWorks에 등록된 ISR이 정상적으로 수행되어 Single Bit EDAC Error를 Correction한 것을 확인할 수 있다.

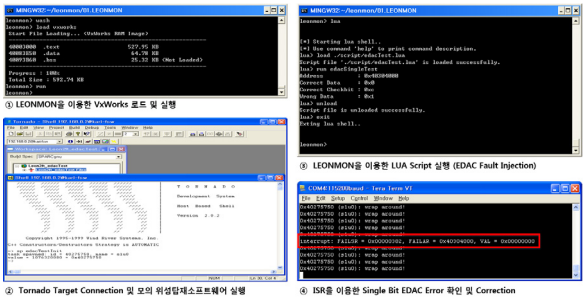

그림 9. LUA Script를 이용한 EDAC Error 시험결과

V. 결 론

본 논문에서는 LEON2FT기반의 AT697F를 적용한 NGSCB에서 시스템 모니터링 및 디버깅 기능을 지원하고, LUA를 이용하여 실시간으로 고장을 주입할 수 있는 시스템을 설계 및 구현하였다. 프로세서 모니터링 및 디버깅 기능이 기존에 사용하던 상용 프로그램과 동일함을 확인하였다. 또한 NGSCB에서 모의 위성탑재소프트웨어를 실행하고, LUA Script를 이용하여 실시간으로 Single Bit EDAC Error를 주입하였을 때 NGSCB가 정상적으로 Error를 처리하는 것을 확인하였다. 향후에는 프로세서의 모니터링 및 디버깅 기능을 강화하고, LUA Script를 이용하여 NGSCB의 PCI, UART, Timer, 1553B, SpW등에 실시간으로 고장을 주입할 수 있는 Script의 개발이 필요하다.

참 고 문 헌

[1] Jong-Wook Choi, Yee-Jin Cheon, "Study of Next Space Processor for Development of Flight Software", KSAS, pp. 809-814, 2012

[2] Jong-Wook Choi, Byeong-Gyu Nam, "System Software Design and Simulation for LEON2-FT Processor based on PCI", KOSST, vol. 8, no. 1, pp. 54-60, 2013

[3] Jae-Yeop Jeong, Jong-Wook Choi, Yee-Jin Cheon, "The Design and Implementation of Processor Debug Monitor for Next Generation Spacecraft Computer Board", KSAS, 2014

[4] Yunki Lee, Jihoon Kim, "A Conceptual Study on Standard Architecture Design for the Next Generation Satellite OBC", KSAS, pp. 1018-1024, 2013

[5] Aeroflex, "GRMON User's Guide", pp. 1-79, 2013

[6] Aeroflex, "GRMON2 User's Guide", pp. 1-203, 2014

[7] ATMEL, "Rad-Hard 32-bit SPARC V8 Processor AT697F", pp. 1-175, 2011

[8] SPARC International, Inc., "The SPARC Architecture Manual Version 8", pp 1-295

[9] http://www.lua.org, "Lua 5.2 Reference Manual", 2013

[10] Kurt Jung, Aaron Brown, "Beginning Lua Programming", pp. 1-673, 2007

저자

정 재 엽(Jae-Yeop Jeong) **정회원**

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

<관심분야> : 임베디드시스템, 실시간운영체제

최 종 욱(Jong-Wook Choi) **정회원**

- 1999년 2월 : 경북대학교 전자공학 학사졸업
- 2001년 2월 : 경북대학교 전자공학 석사졸업
- 2000년 12월 ~ 현재 : 한국항공우주연구원 위성탑재소프트웨어팀

<관심분야> : 시뮬레이터, 실시간운영체제

천 이 진(Yee-Jin Cheon)

- 1993년 2월 : 경북대학교 전자공학 학사졸업
- 1995년 2월 : 경북대학교 전자공학 석사졸업
- 2010년 2월 : 한국과학기술원 전기 및 전자공학 박사졸업
- 1995년 3월 ~ 현재 : 한국항공우주연구원 위성탑재소프트웨어팀

<관심분야> : 실시간제어, 비선형 추정 알고리즘, Fail-safe 알고리즘