위성 시뮬레이터 개발을 위한 ERC32 프로세서 기반의 가상화 시스템 개발

최종욱*, 신현규*, 이재승*, 천이진* 정회원

Virtualized System Development Based on ERC32 Processor for Satellite Simulator

Jong-Wook Choi*, Hyun-Kyu Shin*, Jae-Seung Lee*, Yee-Jin Cheon* *Regular Members*

요 약

위성 탑재소프트웨어를 개발하는 과정에서 프로세서 에뮬레이터와 위성 시뮬레이터는 핵심 개발 툴로서, 탑재소프트웨어 개발/검증 전반에 사용하며 실제 하드웨어를 대체할 수 있는 수준까지 활용이 가능하다. 현재 한국항공우주연구원에서 개발하는 저궤도 위성의 경우 ERC32 프로세서를 사용하며 Aeroflex Gaisler에서 판매하는 TSIM-ERC32 에뮬레이터를 사용하여 탑재소프트웨어 시뮬레이터를 개발하여 탑재소프트웨어 개발 및 테스트에 사용하였으나, 실제 위성 시뮬레이터를 개발하는 과정에서 에뮬레이터 코어를 개발자가 원하는 방식으로 수정 및 변경할 수 없는 문제와 위성 시뮬레이터 연동 시 인터페이스를 쉽게 구현할 수 없는 문제가 발생한다. 본 논문에서는 이러한 문제들을 해결하기 위해 ERC32 코어를 정확히 에뮬레이션 할 수 있는 인터프리트 방식의 Cycle True 에뮬레이터 개발 방법에 대해서 기술하며 에뮬레이터를 이용한 RTOS 기반의 소프트웨어 개발 및 디버깅 환경에 대해서 설명한다.

Key Words : ERC32, emulator, laysim-erc32, TSIM-ERC32

ABSTRACT

During the development of flight software, the processor emulator and satellite simulator are essential tools for software development and verification. SWT/KARI developed the software-based spacecraft simulator based on TSIM-ERC32 processor emulator from Aeroflex Gaisler. But when developing flight software using TSIM-ERC32, there are much limitation for understanding of exact behavior of ERC32 processor, and it is impossible to change or modify the emulator core to develop the satellite simulator. To reslove this problem, this paper presents the development of new cycle-true ERC32 emulator as laysim-erc32 and describes the software development and debugging method on VxWorks/RTEMS RTOS.

I. 서 론

위성 탑재소프트웨어를 개발하는 과정에서 프로세서 에뮬레이터와 위성 시뮬레이터는 핵심 개발 툴로서, 단위 테스트, 통합 테스트, 검증 시험 및 시스템 테스트 등의 개발/검증 전반에 사용되며 실제 하드웨어를 대체할 수 있는 수준까지 활용이 가능하다. 현재 한국항공우주연구원(항우연)에서 개발 중인 저궤도 위성의 탑재컴퓨터 프로세서는 Astrium/ESA에서 개발된 SPARC v7 기반의 MCM-ERC32SC 프로세서를 사용하며, 프로세서 에뮬레이터의 경우 Aeroflex Gaisler에서 상용으로 판매하고 있는 TSIM-ERC32 프로세서 에뮬레이터[1]를 사용한다. TSIM-ERC32 프로세서 에뮬레이터를 기반으로 저궤도 위성 시뮬레이터가 개발되어 탑재소프트웨어 개발/검증, 시스템 테스트 및 지상운영 등에서 사용하고 있다. TSIM-ERC32 프로세서 에뮬레이터를 이용하여 시뮬레이터를 개발 할 경우 에뮬레이터 코어를 개발자가 원하는 방식으로 수정 및 변경할 수 없는 문제와 위성 시뮬레이터와 연동 시 인터페이스를 쉽게 구현할 수 없는 어려움

*한국항공우주연구원 위성비행소프트웨어팀 (jwchoi@kari.re.kr, hkshin@kari.re.kr, jslee@kari.re.kr, yjcheon@kari.re.kr)
접수일자 : 2011년 3월 18일, 수정완료일자 : 2011년 4월 4일, 최종게재확정일자 : 2011년 4월 25일

이 있다. 또한 국내 인공위성 개발 시 ERC32 프로세서를 사용할 경우 지속적으로 TSIM-ERC32 프로세서 에뮬레이터를 구매해야하는 재정적 부담도 가지게 된다.

본 논문에서는 이러한 문제들을 해결하기 위해 ERC32 코어를 정확히 에뮬레이션 할 수 있는 인터프리트 방식의 Cycle True 에뮬레이터 개발 방법에 대해서 기술하며 개발된 에뮬레이터 상에서 탑재소프트웨어 개발 및 디버깅 방법에 대해서 설명한다. 또한 현재까지 개발된 ERC32 에뮬레이터에 대해서 기술한다.

Ⅱ. ERC32 프로세서 및 에뮬레이터 개발 현황

1. ERC32 프로세서 아키텍처

ERC32 프로세서는 SPARC v7 기반의 IU(Integer Unit), 수치연산을 위한 MEIKO FPU(Floating Point Unit), 그리고 모든 주변장치와 메모리 관리, 오류 검출 등을 담당하는 MEC(MEmory Controller)로 구성된다[2].

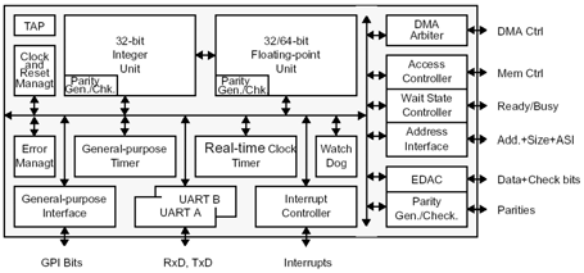


그림 1. ERC32 Processor Architecture

IU는 프로세서의 모든 운영을 담당하며 정수 연산, 메모리 로딩, PC(Program Counter) 유지 및 FPU 명령 수행/관리를 담당하며 IU 제어를 위해 PSR(Processor State Register), WIM(Window Invalid Mask), TBR(Trap Base Register), Y(Multiply/Divide Register), PC/nPC 시스템 레지스터를 가지고 있으며 내부연산을 위해 136개의 Window Register를 내재하고 있다. Window Register는 SAVE, RESTORE 명령을 수행하거나 trap이 발생할 경우 Window Register 전환이 발생하고, 전환된 Window가 WIM과 일치할 경우 window underflow/overflow trap이 발생한다. Trap이 발생할 경우 IU는 ERC32 trap operation에 따라 처리하게 된다.

FPU는 single/double precision을 지원하며 IEEE-754 포맷을 따르고 있다. IU와 동시 수행을 하기 위해 내부적으로 IU와 동일한 pipeline 구조를 가지고 있으며, FPU의 상태와 exception 정보를 가지고 있는 FSR(FPU Status Register), 현재 수행중인 FPU 명령과 주소를 저장하고 있는 FQ(FPU Queue Register) 그리고 32개의 연산레지스터를 제공한다.

MEC의 경우 메모리 크기, waitstate 등을 관리하며 3개의 타이머와 인터럽트 컨트롤러, 2개의 UART 등을 제어하며 오류 검출 등을 관리하며 31개의 시스템 레지스터를 통해서 운영된다.

2. ERC32 에뮬레이터 개발 현황

ERC32 프로세서를 지원하는 에뮬레이터의 경우 표 1과 같이 4개 에뮬레이터는 모두 ESA 관련 업체에서 개발이 되었으며, 항우연 위성비행소프트웨어팀(SWT)에서는 QEMU 0.11.1을 기반으로 한 QEMU laysim-erc32[3]와 이번 논문에서 기술하는 GUI 기반의 Cycle True 에뮬레이터인 laysim-erc32를 개발 하였다.

표 1. ERC32 Processor Emulator Status

Emulator	Type	Supplier	Remark
TSIM-ERC32	Instruction Level	Aeroflex Gaisler	- Cycle True - Used for most ESA Projects - Used for KOMPSAT-3/5 Flight Software Simulator
SimERC32	Instruction Level	Astrium/ CNES	- Astrium Internal (SIMERC32 Emulator in SIMIX) - Gaia Real-Time Simulator
ESOC Simulator	Instruction Level	VEGA/ ESOC	- Used for most ESOC/ESA ground system
QERx	Dynamic Translation	FFQTECH SciSys	- Based on QEMU 0.9.1 - Galileo Constellation Operation Simulator
QEMU laysim-erc32	Dynamic Translation	SWT/KARI	- Based on QEMU 0.11.1 - S/W development on VxWorks/RTEMS
laysim-erc32	Instruction Level	SWT/KARI	- Window & Linux Platform - Source Level Debugging and Cycle True - Used for KOMPSAT-3/5 Ground Operation Simulator

에뮬레이터에서 프로세서를 모사하는 방법은 크게 각 단계별로 타깃 프로세서의 명령어를 실시간으로 해석(Interpretation)하는 방식과 실시간으로 호스트 컴퓨터 코드로 변경하는 동적 변환(Dynamic Translation) 방식으로 나뉜다. 해석 방식은 타깃 수행 코드를 호스트 수행 코드로 매번 변경해야 하는 큰 오버헤드가 항상 존재하며 타깃 프로세서의 시스템 클록이 높을 경우 실시간 성능을 만족할 수 없는 문제가 있는 반면 개발이 용이하고 타깃 프로세서의 사이클과 동일하게 에뮬레이션 할 수 있는 큰 장점이 있다. 동적 변환방식은 블록 단위로 실시간으로 타깃 프로세서의 명령을 호스트 컴퓨터로 변경한 뒤 동일한 블록이 수행될 경우 기 컴파일 된 명령들을 수행함으로서 에뮬레이터의 성능적인 면에서는 가장 좋으나 타깃 프로세서의 사이클을 동일하게 에뮬레이션 할 수 없는 어려움이 있다. laysim-erc32의 경우 해석 방식의 에뮬레이션 기법을 사용하여 실제 타깃 프로세서와 동일한 사이클을 제공하고 있으며 개발자들의 편이성을 위해 GUI 기반으로 실시간 디버깅 환경을 제공 한다. 그리고 QEMU laysim-erc32의 경우 QEMU의 동적 변환기를 사용하여 개발된 ERC32 프로세서 에뮬레이터이며 앞에서 얘기한 해석 방식의 에뮬레이터 보다 10배 이상 높은 성능을 보여주나 cycle true로 에뮬레이션 할 수 없으며 time constrained 속성을 가지고 있는 코드의 검증에 어려움을 가지고 있다.

Ⅲ. GUI 기반의 Cycle-True ERC32 프로세서 에뮬레이터

1. laysim-erc32 에뮬레이터 아키텍처

현재 개발 중인 “GUI 기반의 Cycle True ERC32 프로세

서 에뮬레이터 laysim-erc32"는 리눅스 및 윈도우에서 동일하게 개발/실행 할 수 있으며 GCC와 GUI를 위해 GTK 라이브러리를 사용하여 개발 되었다.

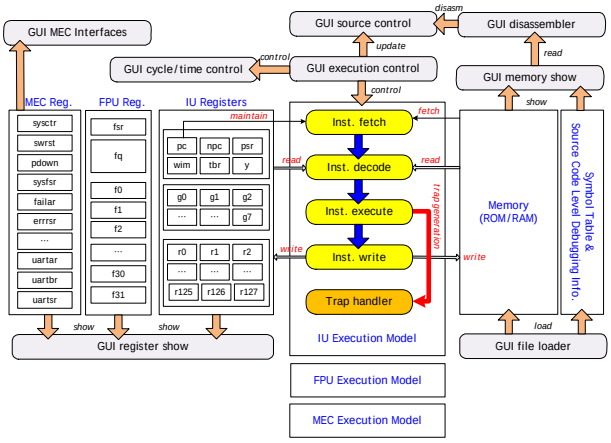


그림 2. laysim-erc32 Emulator Architecture

layim-erc32는 크게 7 부분의 나뉜다. 먼저 ERC32 실행 파일을 메모리 영역으로 로딩하고, 포맷(a.out/elf 포맷)에 맞게 심벌 정보와 디버깅 정보를 분석/적재하는 File Loader Module, 소스 코드와 함께 디스어셈블러 형태로 GUI 화면에 출력하는 Source/Disassembler Module, IU의 명령어를 수행하는 IU Execution Module, FP 명령어를 수행하는 FPU Execution Module, MEC operation을 담당하는 MEC Execution Module, trap/interrupt를 처리하는 Trap/Interrupt Module 그리고 watch/breakpoint, 실시간 레지스터 업데이트, control 등을 담당하는 GUI Control Module로 구성된다.

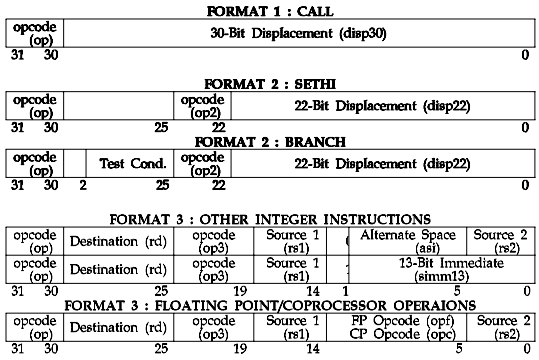
2. File Loader Module

laysim-erc32에 로딩 될 수 있는 파일 포맷은 VxWorks 5.4에서 빌드 시 생성되는 a.out 포맷과 LECCS/RCC에서 빌드 될 때 생성되는 elf 포맷 그리고 심벌 및 디버깅 정보가 없는 binary 포맷을 지원한다. GUI 환경에서 ERC32 실행 파일을 로딩 할 경우 해당 파일의 포맷을 분석한 뒤 해당 포맷에 맞는 로더가 수행되며, 로더에서는 심벌 정보와 디버깅 정보를 추출 하고 ERC32 메모리에 text/data 세그먼트를 복사하게 된다. 만약 RAM에서 실행되는 이미지일 경우 stack/frame pointer를 RAM에서 수행될 수 있도록 설정한다.

3. Source/Disassembler Module

File Loader Module을 통해 로딩 된 ERC32 실행 코드는 디버깅 정보와 매칭 되는 C 소스 코드가 있을 경우 mixed 모드로 보여주며 매칭 되는 C 소스 코드가 없을 경우나 디버깅 정보가 없을 경우 어셈블러 형태로 GUI 화면에 출력한다. GUI 화면에 어셈블러 형태로 보여줄 때 개발자들의 편이성을 위해 SPARC에서 제안하는 "Suggested Assembly Language Syntax"[4]를 따르고 있다. ERC32 SPARC v7에서 제공되는 명령은 크게 Load and Store Instruction, Arithmetic/Logical/Shift Instruction, Control Transfer Instruction, Read/Write Control Register Instruction, FP/CP Instruction으로 구성되며, 그중 실제 ERC32에서 사용할 수 있는 명령어는 171개이며 각 명령은 표 2와 같이 크게 3개의 포맷으로 구성된다[3].

표 2. ERC32 SPARC v7 Instruction Format



코드의 실행 여부를 추적할 수 있도록 수행되지 않은 코드의 주소는 검은색으로 표시가 되고 수행된 코드의 주소는 파란색으로 표시 되며, 현재 수행되는 코드의 경우 붉은색으로 표시되며 추후 code coverage 기능을 구현하여 C 소스 코드 레벨에서의 trace가 가능 하도록 개발 할 예정이다.

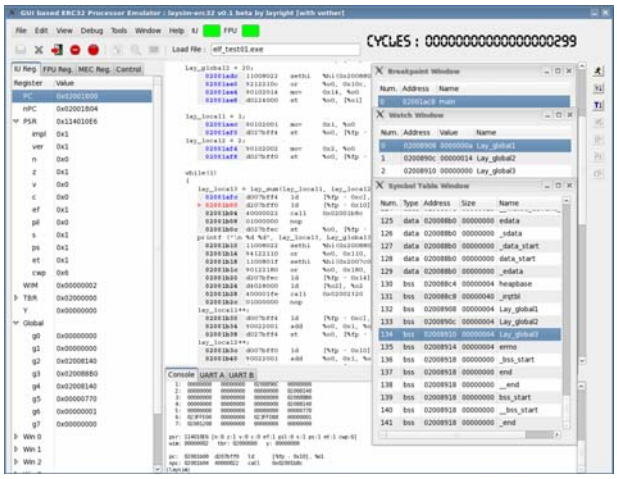


그림 3. laysim-erc32 Source/Disassembler

4. IU Execution Module

ERC32의 실제 SPARC 명령을 수행하는 IU Execution Module은 하나의 thread로 동작하며, GUI에서 run, stop, step 등의 명령에 의해 컨트롤 된다. IU Execution Module은 IU의 4단계 pipeline에 맞게 명령어를 Fetch → Decode → Execute → Write 단계로 실행되며, 전체적으로 그림 4와 같은 흐름으로 수행된다.

Fetch 단계에서는 현재 PC/nPC에 있는 주소를 통해 명령어를 메모리에서 가져온다. 만약 PC/nPC에 설정된 주소를 접근 할 수 없을 경우 trap이 발생한다. 다음 단계로 현재 처리해야 할 인터럽트가 있는지를 확인하고, 인터럽트를 수행 할 수 있는 조건(PSR의 trap이 enable되어 있고 수행해야 할 인터럽트 레벨이 pil보다 작을 경우 등) 인지를 확인한 뒤 TBR을 업데이트하고 해당 인터럽트를 처리한다. Decode 단계에서는 해당 명령어를 SPARC v7 명령어 레벨로 분석한 뒤 해당 명령어를 에뮬레이션하는 함수를 호출한다. Execute/Write 단계에서는 Decode 단계에서 호출된 에뮬레이션 함수가 수행되며, 명령어 수행 시 필요한 레지스터와 메모리를 읽고 그 결과를 레지스터/메모리에 저장한다. 수치연산을 처리하는 FPU 연산일 경우 FPU Execution Module을 통해서 처리하게 되며, 각 명령어 수행 단계마다 privilege, align, trap condition 등을 확인하게 되고 예외사항 발생 시 해당 trap을 설정한다. 그리고 해당 명령어의 cycle을 계산하며, 추후 MEC Execution Module에서 시스템 클록과 관련된 타이머 레지스터를 업데이트 한다. Execute/Write 단계에서 trap이 발생한 경우 trap handler에서 해당 trap을 ERC32 trap operation에 따라 처리하며 만약 trap을 처리할 수 없을 경우 ERC32는 Error mode로 천이하고 MEC의 에러와 관련된 레지스터들을 업데이트 한다. MEC Execution Model에서는 다양한 타이머와 UART 등을 처리하고 마지막으로 GUI를 업데이트 하게 된다.

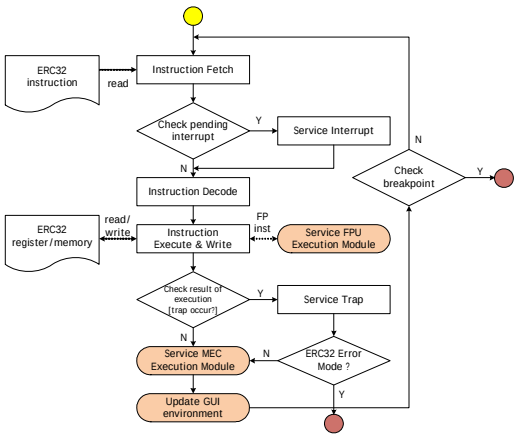


그림 4. laysim-erc32 IU Execution Module Flow

5. FPU Execution Module

FPU의 경우 IEEE-754 포맷을 사용하기 때문에 호스트 컴퓨터인 x86 머신의 자원을 이용하여 수치연산을 수행하고, 그 결과를 FPU 레지스터에 반영한다. FPU 예외사항이 발생할 경우 먼저 호스트 컴퓨터의 예외사항을 정확히 처리한 뒤 그 상태 정보를 FSR에 반영한다. ERC32 FPU의 경우 한번에 하나의 FP 명령어를 수행할 수 있기 때문에 연속적으로 FP 명령어가 수행될 경우 먼저 수행된 FP 명령어가 끝날 때까지 FP Queue에 다음 명령어가 저장되고 FSR의 qne가 1(not empty)로 설정되며 동시에 IU의 수행도 FP Queue가 empty가 될 때까지 freeze된다. 이러한 FPU 처리 방식 때문에 IU 보다 cycle 계산방법이 복잡하며 IU/FPU 모두 그 앞의 연산에서 수행된 레지스터의 결과가 현재 수행하는 명령어의 소스 레지스터로 사용될 경우 H/W interlock이 발생하며 추가적인 cycle이 소요된다. 현재 laysim-erc32에서는 이러한 H/W interlock까지 모두 실제 하드웨어와 동일하게 구현되어 있다. FPU의 경우 IU 운영 모드(Reset, Run, Error mode)와 달리 정상적으로 수행되는 Execution 모드에서 예외사항(divide by zero, overflow/underflow 등)이 발생할 경우 Pending Exception 모드로 천이하게 되지만 IU에서는 즉각적으로 FPU의 에러사항을 인지할 수 없다. 이후 IU에 의해 다시 FP 명령어를 수행할 때 비로소 Exception 모드로 천이하게 되고 IU에 의해 trap 8 (FPU exception)이 발생하게 된다. 탑재소프트웨어에서 정확히 FPU exception을 처리할 경우 다시 FP Queue는 empty로 전환되고 FPU는 Execution 모드로 천이되어 정상적으로 FP 명령어를 수행할 수 있다.

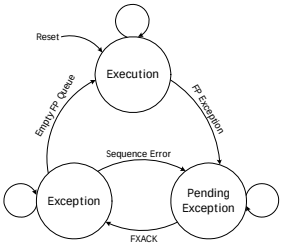


그림 5. laysim-erc32/ERC32 FPU Operation Mode

6. MEC Execution Module

ERC32 MEC는 메모리/인터럽트/IO 컨트롤러, 2개의 UART, 3개의 타이머, fault detection/injection, 그리고 소프트웨어에서 액세스할 수 있는 레지스터들로 구성된다. laysim-erc32에서는 ERC32 MEC의 모든 기능을 구현하였으며 실제 하드웨어와 동일하게 소프트웨어에서 접근/사용 할 수 있다. MEC 레지스터의 경우 supervisor 모드에서만 write/read할 수 있는 레지스터를 user 모드에서 액세스할 경우 trap이 발생한다.

메모리 컨트롤러의 경우 laysim-erc32의 RAM/ROM의 크기, waitstate를 설정하며 소프트웨어가 미구현 된 영역을 액세스할 경우 trap 1 (instruction access) / trap 9 (data access)가 발생하며, waitstate는 추가적인 cycle을 소요하게 된다. ERC32는 15개의 외부/내부 인터럽트를 제공하며 내부 인터럽트들은 laysim-erc32에서 MEC 운영 시 타이머/UART 등에서 사용하며 외부 인터럽트의 경우 외부 모듈(예를 들어 VASI 모듈 로딩) 사용 시 해당 인터럽트가 발생한다. IO 컨트롤러의 경우 ERC32에서 제공되는 4개의 I/O 영역을 사용할 수 있도록 크기와 waitstate를 설정하는 부분이며 I/O 영역에 다양한 디바이스를 등록

할 수 있으며 현재 항우연의 저궤도 위성 시뮬레이터 개발을 위해 SpaceWire, PM32 FPGA controller, VASI 등을 구현하였다[5]. 타이머의 경우 ERC32에서 제공되는 RTC (Real-Time Clock Timer), GPT (General-purpose Timer), WDT (Watchdog Timer) 3개의 타이머를 실제 하드웨어와 동일한 방식으로 laysim-erc32에서 제공하며 IU/FPU 명령어 수행 시 소요된 cycle에 맞게 타이머의 scaler/count가 감소하며 timeout이 발생할 경우 해당 인터럽트가 발생하고 IU Execution Module에서 처리하게 된다. 두개의 UART를 통해 외부와 통신을 수행하게 되며 laysim-erc32 GUI에서 제공되는 UART A/B console을 통해 유저의 입력 및 다양한 정보를 출력할 수 있다. UART 인터럽트는 매 byte단위로 입출력 시 인터럽트 4/5가 발생하며 에러 발생 시 인터럽트 7이 발생한다. 그림 6은 laysim-erc32에서 RTC에 의한 인터럽트와 UART 관련 수행 결과를 보여준다.

표 3. laysim-erc32/ERC32 Interrupts

Interrupt	tt	Comments
Watchdog timeout	0x1F	level=15	WDT는 Interrupt Mask Register를 통해서 masking할 수 없으며, IWDE가 1(active)인 경우에만 동작하게 된다. WDTTimeout이 발생하면 IRQ 15가 발생하게 되고, WDTResetTimeout안에 WDOGTR를 재설정 하지 않을 경우 WDTreset이 발생하여 laysim-erc32를 리셋 시킨다.
External INT 4	0x1E	level=14	외부 인터럽트 4번으로 laysim-erc32에서는 사용하지 않음 외부 VASI 모듈을 연결시 VASI RTC Cycle/Slot 인터럽트 사용
Real time clock timer	0x1D	level=13	ERC32 RTCTimeout이 발생하면 IRQ 13을 발생시킨다.
General-purpose timer	0x1C	level=12	ERC32 GPTTimeout이 발생하면 IRQ 12를 발생시킨다.
External INT 3	0x1B	level=11	외부 인터럽트 3번으로 laysim-erc32에서는 사용하지 않음 외부 VASI 모듈을 연결시 VASI Error 인터럽트로 사용
External INT 2	0x1A	level=10	외부 인터럽트 2번으로 laysim-erc32에서는 사용하지 않음 외부 VASI 모듈을 연결시 VASI Nominal 인터럽트로 사용
DMA timeout	0x19	level=9	DMA session exceeds permitted time
DMA access error	0x18	level=8	DMA performs an access error, access violation or illegal access
UART Error	0x17	level=7	ERC32 UART 통신 중 overrun/parity/frame 에러가 발생할 경우 IRQ 7이 발생한다.
Correctable memory error	0x16	level=6	SRAM Single Bit EDAC Error가 발생하면 IRQ 6이 발생하게 된다.
UART A Rx/Tx	0x15	level=5	ERC32 UART A 통신은 Byte단위로 Rx/Tx 인터럽트가 발생한다.
UART B Rx/Tx	0x14	level=4	ERC32 UART B 통신은 Byte단위로 Rx/Tx 인터럽트가 발생한다.
External INT 1	0x13	level=3	외부 인터럽트 1번으로 laysim-erc32에서는 사용하지 않음 외부 모듈 연결시 특정 인터럽트로 사용 가능
External INT 0	0x12	level=2	외부 인터럽트 0번으로 laysim-erc32에서는 사용하지 않음 외부 모듈 연결시 특정 인터럽트로 사용 가능
Masked hardware errors	0x11	level=1	ERC32의 H/W error와 발생했을 때 maske되어 있으면 IRQ 1 발생한다.

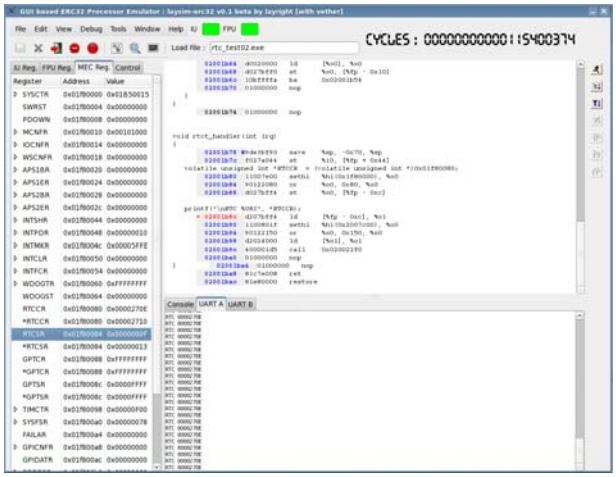


그림 6. laysim-erc32 RTC/UART Test

7. Trap/Interrupt Handling Module

ERC32는 크게 최초 파워가 들어간 상태인 Reset mode, 정상적으로 명령을 수행하는 Run mode, 그리고 수행 중 에러가 발생한 경우 더 이상 수행을 하지 못하고 halt하게 되는 Error mode로 구분된다. ERC32에서 명령어를 수행 중 발생할 수 있는 trap은 크게 synchronous trap과 asynchronous trap (interrupt)로 나뉜다. Synchronous trap중 window overflow/underflow trap의 경우 소프트웨어에서 기본적으로 trap handler를 제공하기 때문에 정상적으로 처리되지만, 나머지 synchronous trap (예를 들어 UNIMP, privilege, align error)이 발생할 경우 대부분 소프트웨어에서 처리하지 못하기 때문에 결국 Error mode로 천이된다. laysim-erc32에서도 소프트웨어에서 synchronous trap을 처리하지 못할 경우 동일하게 Error mode로 천이하고 더 이상 명령어를 수행하지 않는다. Asynchronous trap (interrupt)의 경우 앞의 표 3과 같이 15개의 인터럽트에 의해서 발생하며 인터럽트 발생 시 synchronous trap이 없을 경우 IU에 의해서 처리된다. 이러한 trap 처리는 IU Execution Module에서 각 명령어를 수행하는 단계별로 trap 조건이 발생했는지 확인하고 trap 조건일 경우 ERC32 프로세서 trap operation과 동일하게 아래 표 4와 같은 순서로 처리한다. 만약 trap이 disable된 상태에서 synchronous trap이 발생할 경우 ERC32는 Error mode로 천이한다.

표 4. laysim-erc32/ERC32 Trap Operation

1) Further traps are disabled (asynchronous traps are ignored; synchronous traps force an error mode).
2) The S bit of the PSR is copied into the PS bit; the S bit is then set to 1.
3) The CWP is decremented by one (modulo the number of windows) to activate a trap window.
4) The PC and nPC are saved into r[17] and r[18], respectively, of the trap window.
5) The tt field of the TBR is set to the appropriate value.
6) If the trap is not a reset, the PC is written with the contents of the TBR and the nPC is written with TBR + 4. If the trap is a reset, the PC is set to address zero and the nPC to address four.

8. GUI Control Module

laysim-erc32는 GUI Control Module에 의해 single step mode 및 run mode로 수행 할 수 있다. Single step mode에서는 각 명령의 수행 결과에 따른 레지스터와 명령 수행 결과 등을 실시간으로 trace할 수 있으며 기존 GDB 등에서 제공하는 다양한 breakpoint, watch, run to cursor, set PC here 등의 기능을 수행할 수 있으며, run mode에서는 실제 ERC32 하드웨어와 동일하게 수행되며 수행 중 breakpoint가 설정된 주소와 PC가 일치 할 경우 single step mode로 변경되고 마지막 프로세서 정보가 GUI 화면에 업데이트 된다. Run mode에서는 매번 GUI를 업데이트 할 경우 성능 저하가 발생하기 때문에, 실시간으로 ERC32 레지스터를 GUI 화면에서 확인 할 수 없으나 watch 기능을 통해서 모니터링 하는 변수/주소들은 계속 trace 할 수 있으며, console 창을 통해 수행 중인 메모리와 레지스터의 정보들을 확인할 수 있다. 또한 수행 중 예외사항이 발생하여 ERC32가 Error mode로 천이하게 되면 single step mode로 변경 된다. GUI cycle/time

control을 통해서 현재 수행중인 명령어의 개수와 cycle을 확인할 수 있으며 추후 performance report를 통해 CPU performance, Real-time performance, Simulation performance 등을 리포팅할 수 있게 개발할 예정이다.

Ⅳ. laysim-erc32에서의 소프트웨어 개발 및 디버깅

1. VxWorks/RTEMS 기반의 소프트웨어 개발 환경

기존 TSIM-ERC32 프로세서 에뮬레이터를 이용하여 개발된 시뮬레이터와 동일하게 laysim-erc32에서도 VxWorks 5.4/RTEMS RTOS 기반의 소프트웨어를 변경 없이 실제 하드웨어와 동일하게 로딩 및 수행 할 수 있다.

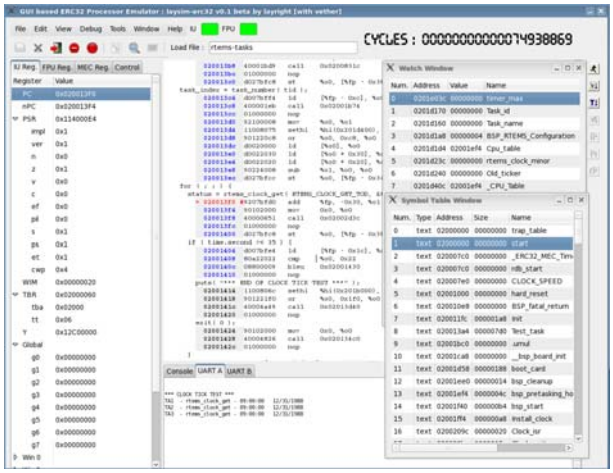


그림 7. S/W Development based on RTEMS

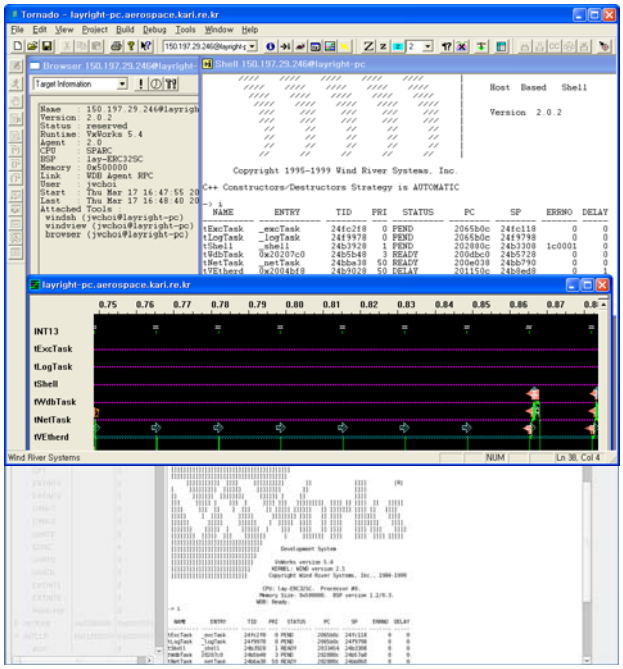


그림 8. S/W Development based on VxWorks/Tornado

그림 7은 RTEMS 기반의 소프트웨어 개발 환경을 보여주며, 기존 TSIM-ERC32에서는 GDB/DDD를 사용하였으나 laysim-erc32에서는 내재된 디버깅 환경을 이용하여 C 소스 레벨 디버깅을 할 수 있으며 실시간으로 변수/메모리를 추적할 수 있다. 또한 control console을 통해 실시간으로 레지스터 정보 및 메모리 정보 등을 확인할 수 있다. 그림 8은 Windows기반의 VxWorks/Tornado에서의 소프트웨어 개발 환경을 보여준다. laysim-erc32에서 가상 네트워크를 통해 Tornado의 Target Server/WDB와 연동되어 소프트웨어 모듈 다운로딩 및 디버깅을 수행할 수 있으며 WindView를 통한 각 태스크의 수행시간 및 자원의 사용을 확인할 수 있다.

2. TSIM-ERC32 vs laysim-erc32

TSIM-ERC32 프로세서 에뮬레이터와 기존 SWT에서 개발된 QEMU laysim-erc32, 이번 논문에서 기술한 GUI 기반의 Cycle True 에뮬레이터인 laysim-erc32를 전체적으로 표 5에 정리하였다. 에뮬레이터 코어의 성능을 비교하기 위해 Fedora 8 Linux(Intel Dual Core E8400 3GHz)에서 Dhrystone 400,000번 수행시간을 절대시간으로 측정/비교한 결과 동적 변환기를 사용하는 QEMU가 7배 이상 빠른 성능을 보여주며 laysim-erc32의 경우 TSIM-ERC32보다 0.6배의 성능을 보여주나 실제 ERC32 H/W보다는 4배 이상의 성능을 보여준다.

표 5. Comparison of ERC32 Processor Emulator

	TSIM-ERC32	QEMU laysim-erc32	GUI laysim-erc32
Emulation Method	Interpreter	Dynamic trans.	Interpreter
Performance with Dhrystone [TSIM is x8 times faster than real ERC32]	factor 1 [factor 1.65]	factor 7.3 [factor 12.0]	factor 0.6 [factor 1]
Timing Accuracy	Cycle True	Absolute Time	Cycle True
I/O Emulation	Loadable modules for user-define I/O	Implemented in Source Level	Implemented in Source Level
Standalone Debugging [Debugging after simulation stop]	Yes	No	Yes
Realtime register/variable monitoring	No	registers/memory view	R.T variable trace/ memory&register view
Code Coverage Monitoring	Yes	Not yet	Yes
Performance Report	Yes	No	Not yet
Breakpoint (standalone)	Unlimited	No	Unlimited
GDB Connection	Support	Support	No [Standalone Debugger]
EDAC Emulation (only TSIM2-ERC32)	No (Yes)	No	No
RTEMS/VxWorks Awareness	Yes/No	No	Future Work
Support Formats	S-rec, elf, a.out (no binary)	elf, a.out, binary (no S-rec)	elf, a.out, binary (no S-rec)
ERC32 Completeness	100%	80% [No Fault detection/injection]	Aim for TSIM-ERC32 (No EDAC Emul.)
License	11,000€	Open-Source [in-house]	In-house

Ⅴ. 결 론

본 논문에서는 GUI 기반의 Cycle True ERC32 프로세서 에뮬레이터 laysim-erc32의 개발 방법과 laysim-erc32를 이용한 소프트웨어 개발 및 디버깅 방법에 대해서 기술하였다. 현재 개발된 laysim-erc32는 ERC32의 모든 기능을 실제 하드웨어와 동일하게 모사하며 실제 하드웨어에서 동작하는 VxWorks/RTEMS를 수정 없이 로딩 및 수행할 수 있으며 가상 네트워크를 통해 Tornado상에서 WDB를 이용한 실시간 디버깅 및 모니터링을 수행할 수 있다. 또한 GDB/DDD 없이 에뮬레이터 레벨에서 C 소스코드 디버깅이 가능하며 실시간으로 변수/메모리를 모니터링 할 수 있다. 현재 laysim-erc32의 경우 instruction level test 및

operation test가 모두 수행되었으며 지상국 Operation Simulator를 개발하기 위한 핵심 코어로 사용되고 있으며 추후 SWT/KARI에서 개발하는 에뮬레이터 및 시뮬레이터의 코어로 사용될 예정이다.

참 고 문 헌

[1] Gaisler Research, “TSIM Simulator User’s Manual v1.3.8,”, January, 2006.

[2] ATMEL, "TSC695F SPARC 32-bit Space Processor User Manual,", December, 2003.

[3] Jong-Wook Choi, "Development of High performance ERC32 Processor Emulator based on Dynamic Translation Emulation Method," 2010 Joint Conference on Satellite Communications, pp, 91-96, 2010.

[4] SPARC International Inc., "The SPARC Architecture Manual Version 8,", 1992.

[5] 최종욱, 신현규, 이재승, 천이진, "소프트웨어 기반의 위성 시뮬레이터를 이용한 위성 탑재소프트웨어 개발 및 검증 방안", 통신위성우주산업연구회논문지, 제5권, 2호, pp. 1-7, 2010.

저 자

최 종 욱 (Jong-Wook Choi) 정회원

1999년 2월 : 경북대학교 전자공학 학사졸업
2001년 2월 : 경북대학교 전자공학 석사졸업
2000년 12월~현재 : 한국항공우주 연구원 위성비행소프트웨어 팀

<관심분야> 시뮬레이터, 실시간운영체제

신 현 규 (Hyun-Kyu Shin) 정회원

2001년 2월 : 한국과학기술원 전산학 학사졸업
2007년 2월 : 한국과학기술원 소프트웨어공학 석사졸업
2007년 12월~현재 : 한국항공우주 연구원 위성비행소프트웨어 팀

<관심분야> Java, 실시간운영체제, 개발프로세스

이 재 승 (Jae-Seung Lee) 정회원

1999년 2월 : 경북대학교 전자공학 학사졸업
2001년 2월 : 경북대학교 전자공학 석사졸업
2000년 12월~현재 : 한국항공우주 연구원 위성비행소프트웨어 팀

<관심분야> 검증시험, 위성명령처리

천 이 진 (Yee-Jin Cheon) 정회원

1993년 2월 : 경북대학교 전자공학 학사졸업
1995년 2월 : 경북대학교 전자공학 석사졸업
2010년 2월 : 한국과학기술원 전기 및 전자공학 박사졸업
1995년 3월~현재 : 한국항공우주 연구원 위성비행소프트웨어 팀

<관심분야> 실시간 제어, 비선형 추정 알고리즘, Fail-safe 알고리즘