우주용 메모리의 자동 오류극복을 위한 오류 정정기 제어 알고리즘 개발

논 문
60-5-21

Development of Error-Corrector Control Algorithm for Automatic Error Detection and Correction on Space Memory Modules

곽 성 우* · 양 정 민†
(Seong Woo Kwak · Jung-Min Yang)

Abstract - This paper presents an algorithm that conducts automatic memory scrubbing operated by dedicated hardwares. The proposed algorithm is designed so that it can scrub entire memory in a given scrub period, while minimally affecting the execution of flight softwares. The scrub controller is constructed in a form of state machines, which have two execution modes - normal mode and burst mode. The deadline event generator and period tick generator are designed in a separate way to support the behavior of the scrub controller. The proposed controller is implemented in VHDL code to validate its applicability. A simple version of the controller is also applied to mass memory modules used in STSAT-3.

Key Words : Single event upset (SEU), Memory scrub, Space electronics, Error detection and correction (EDAC)

1. 서 론

우주용 메모리 모듈은 우주 환경에 의해 독특한 현상을 경험한다. 위성체가 우주 환경에 직면하면 위성체의 전자회로는 우주선(cosmic ray)에 의한 고에너지 입자, 태양으로부터 분출되는 입자, 그리고 지구 자기장에 붙잡혀 있는 여러 에너지 입자들로부터 영향을 받는다. 이 중 SEU(Single Event Upset)는 이들 고에너지 입자에 의해 메모리에 저장된 정보 비트(bit)의 값이 반전되는 현상이다. SEU가 발생한 메모리의 데이터를 복구하지 않으면 메모리에 저장된 비행 소프트웨어(flight software)의 프로그램 코드에 영향을 주어 시스템이 오동작 하는 경우가 발생한다. 특히 위성체의 자세 제어 소프트웨어가 영향을 받을 경우 이와 같은 오동작은 위성체 전체를 위험에 빠트린다. 따라서 우주용 메모리 모듈은 반드시 우주 환경에 의해 발생하는 SEU에 효과적으로 대처할 수 있도록 설계되어야 한다[1]-[3].

SEU에 대처하기 위해서는 SEU가 발생하지 않도록 특수하게 제조된 우주급(space grade) 메모리를 사용하거나, SEU가 발생할 수 있는 상용 메모리를 사용하면서 SEU 탐지 및 복구를 위한 부가적인 하드웨어와 소프트웨어를 추가해야 한다. 첫 번째 방법의 경우 SEU를 원천적으로 제거할 수 있으나 고가의 메모리를 사용해야 된다는 단점이 있다. 따라서 비교적 저비용으로 개발되는 “과학기술위성”과 같은 저궤도 소형위성 또는 대규모 메모리가 사용되는 위성 대용량 메모리 모듈(mass memory unit) 등의 경우 경제적으로 유리한 후자의 방법을 많이 선택한다[4]-[6].

본 연구에서는 별도의 비행 소프트웨어에 의해 수행되는 메모리 스크럽(scrub) 작업을 하드웨어에 의해 자동적으로 수행하는 알고리즘을 개발한다. 본 논문에서 개발된 메모리 스크럽 알고리즘은 비행 소프트웨어의 실행에 가능한 영향을 주지 않으면서 주어진 스크럽 주기 이내에 전체 메모리를 스크럽 하도록 설계된다. 기존 연구 결과와 비교하여 본 기법이 가지는 우수성을 요약하면 다음과 같다.

i) 위성 비행 소프트웨어 개발자는 메모리 스크럽 작업을 위한 별도의 소프트웨어를 개발할 필요가 없고, 주어진 주기 이내에 전체 메모리를 스크럽 하도록 소프트웨어를 설계하여야 하는 부담을 줄일 수 있다.
ii) 메모리 스크럽을 위한 전용 소프트웨어를 실행하기 위해 다른 중요한 비행 소프트웨어의 실행을 희생하는 가능성을 없앨 수 있어 CPU의 효율을 극대화한다. 즉 동일한 성능의 CPU로 더 많은 작업을 가능토록 하여 위성의 고성능화에 일조할 수 있다.
iii) 고가의 우주용 메모리를 사용하는 대신 저비용으로 우주급 메모리에 버금가는 SEU에 대한 강인성을 확보할 수 있어 위성의 개발 비용을 절감하리라 기대된다. 따라서 본 연구는 향후 국내에서 개발될 위성의 저비용 개발 목적에 부합한다.

본 논문의 구성은 다음과 같다. 2장에서는 포와송(Poisson) 분포로 표현되는 메모리 접근 모델에 대해서 기술한다. 3장에서는 본 논문의 주제인 스크럽 제어 시스템을 제안하고 제어기를 설계하는 방법을 기술한다. 4장에서는 제안된 제어기를 VHDL 코드로 구현하고 각 이벤트의 발생을 보여주는 실험을 실시한다. 마지막으로 5장에서 본 논문의 결론을 내린다.

* 정 회 원 : 계명대 전자공학과 부교수
† 교신저자, 정회원 : 대구가톨릭대 전자공학과 부교수
E-mail : jmyang@cu.ac.kr
접수일자 : 2011년 2월 7일
최종완료 : 2011년 4월 13일

2. 메모리 접근 모델

이번 연구에서는 CPU가 메모리에 접근하는 형태가 접근율 λ를 가진 포와송(Poisson) 분포를 따른다고 설정한다. 포와송 분포 모델로부터 CPU의 메모리 접근을 다음과 같은 식을 이용하여 나타낼 수 있다[7].

$a_n(\lambda,t) = (\lambda t)^n e^{-\lambda t}/n!$

여기서 $a_n(\lambda,t)$는 t 시간 내에 n번의 메모리 접근이 있을 확률이다. 따라서 시간 t 구간에서 메모리 접근이 한 번도 없을 확률을 b(t), 한 번이라도 있을 확률을 c(t)라 정의하면 두 확률 식은 다음과 같이 나온다.

$b(t) = e^{-\lambda t}$

$c(t) = 1-e^{-\lambda t}$

이때 b(t)는 CPU가 메모리에 접근한 이후 다음 접근까지의 시간에 대한 확률 분포로 해석할 수 있다. 포와송 분포는 과거의 사건 발생과 관련 없이 현재 확률이 결정되므로 메모리 접근 이후 다음 접근까지의 확률 분포는 가장 최근의 메모리 접근 시각과는 관계가 없으며, 최근 메모리 접근 후 경과한 시간에만 관련된다.

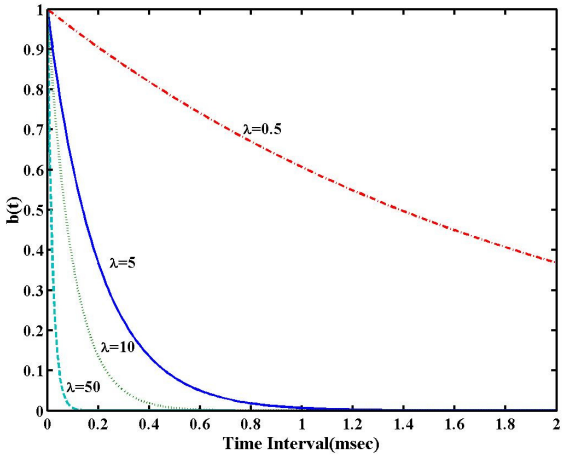

그림 1 포와송 분포를 이용한 메모리 접근 형태 모델링.

Fig. 1 Model of memory access configuration using Poisson distribution.

그림 1은 CPU의 메모리 접근율 λ가 0.5, 5, 10, 50($msec^{-1}$)인 경우의 경과 시간에 따른 확률 b(t)를 보여준다. 이것은 CPU의 메모리 접근 시각의 평균 간격이 각각 1/0.5, 1/5, 1/10, 1/50 msec인 경우의 확률에 해당된다. 메모리 접근 시각 간격이 짧아짐에 따라 연속해서 메모리 접근이 한 번도 없을 확률 b(t)가 시간 경과에 따라 급격히 감소함을 알 수 있다. 즉 메모리의 다음 접근 확률이 커진다. 비행 소프트웨어를 분석하면 평균적인 메모리 접근율을 구할 수 있고, 이를 기반으로 다음 메모리 접근까지의 시간 경과를 확률적으로 구하는 일이 가능하다. 따라서 이것을 기반으로 CPU의 다음 메모리 접근 시각을 확률적으로 예측할 수 있다.

이 모델의 경우 n-bit 카운터를 사용하여 지정된 확률 값을 넘어서는 시각을 간단하게 체크할 수 있어 실제 디지털 회로로 구현하는 데 큰 어려움이 없다. 예를 들어 CPU에 의한 메모리 접근이 없을 확률 값이 $e^{-0.2}$=0.8187이라고 하면 $b(t) = e^{-\lambda t}$이므로 예상 경과 시간 t는 t=0.2/λ로 유도된다. 즉 향후 0.2/λ 동안에는 메모리 접근이 없을 확률이 0.8187이라는 것을 의미한다. 또한 카운터 구동 주파수 f를 이용하면 t에 해당하는 카운터 값 n은 n=0.2f/λ로 구해진다.

3. 스크럽 제어기 설계

3.1 상태 머신 및 상태 제어기 설계

메모리 스크럽 알고리즘의 기본 개념은 비행 소프트웨어의 메모리 요구를 방해하지 않으면서 주어진 주기 이내에 전체 메모리를 스크럽하는 것이다. 이 목표를 달성하기 위하여 본 연구에서는 그림 2와 같은 메모리 스크럽 제어 시스템(memory scrub control system)을 설계하고자 한다. 스크럽 제어기의 입력은 전체 메모리 크기(total memory size), 설정된 메모리 스크럽 주기(memory scrub period), 그리고 비행 소프트웨어의 실행에 따른 CPU의 메모리 요구(CPU memory request)이며, 출력은 메모리 스크럽을 실제 실행하는 작업(memory scrub)이다. 스크럽 제어기는 그림과 같이 비행 소프트웨어 모델(flight s/w model), 상태 제어기(state controller), 그리고 상태 머신(state machine) 등 3개의 블록으로 구성된다.

상태 제어기는 비행 소프트웨어 모델을 기반으로 "CPU memory request" 신호가 발생될 빈도와 패턴을 확률적으로 예측한다. 상태 제어기는 제어 가능 이벤트 신호를 이용하여 상태 머신을 제어함으로써 CPU의 메모리 접근 가능성을 높이고, 주어진 메모리 스크럽 주기 이내에 스크럽 작업을 완료할 확률을 최대화 하도록 설계된다.

상태 머신은 스크럽 제어기의 운용을 나타내기 위한 것으로 스크럽 제어기의 각 상태를 나타낸다. 그림 3은 본 논문에서 사용하는 상태 머신의 흐름도이다. "Memory Scrub" 상태에서는 제어기가 실제 메모리 스크럽 작업을 수행하며, "Memory Access"는 CPU(또는 외부 기기)가 메모리에 접근하는 상태를 가리킨다. "Forced Scrub"은 CPU 또는 외부로부터의 메모리 접근을 제한하고 오로지 스크럽 작업만을 실행하는 상태이다. 이 상태는 외부로부터의 메모리 접근 때문에 스크럽 작업이 오랜 시간 동안 중단되어 메모리 데이터가 SEU에 의해 손상될 가능성이 매우 높을 경우 강제로 외부 메모리 접근을 제한하고 스크럽 작업을 우선적으로 실행하는 경우에 사용된다. "Idle" 상태는 CPU 등 외부 기기의 메모리 접근이 없고 스크럽 주기 이내에 모든 스크럽 작업을 완료하여 다음 스크럽 주기 시작 시간까지 대기하는 상태이다.

상태 머신의 상태를 천이하게 하는 이벤트(event) 종류와 설명은 표 1에 나와 있다. mem_request와 mem_release는 CPU 또는 외부 기기의 메모리 접근 또는 해지를 나타내는 이벤트로서 CPU나 외부 기기의 요구에 의해 발생되므로 제어 불가능하다. 한편 scrub_begin과 scrub_end, 그리고 forced_scrub은 각각 스크럽 시작, 스크럽 종료, 강제 스크럽 실행을 의미하는 이벤트들이며 모두 제어 가능하다.

상태 제어기는 표 1에 나와 있는 세 가지 제어 가능 이벤트 신호를 이용하여 상태 머신을 제어하며 제한된 시간(데드라인) 이내에 전체 메모리를 모두 스크럽 하도록 설계된다.

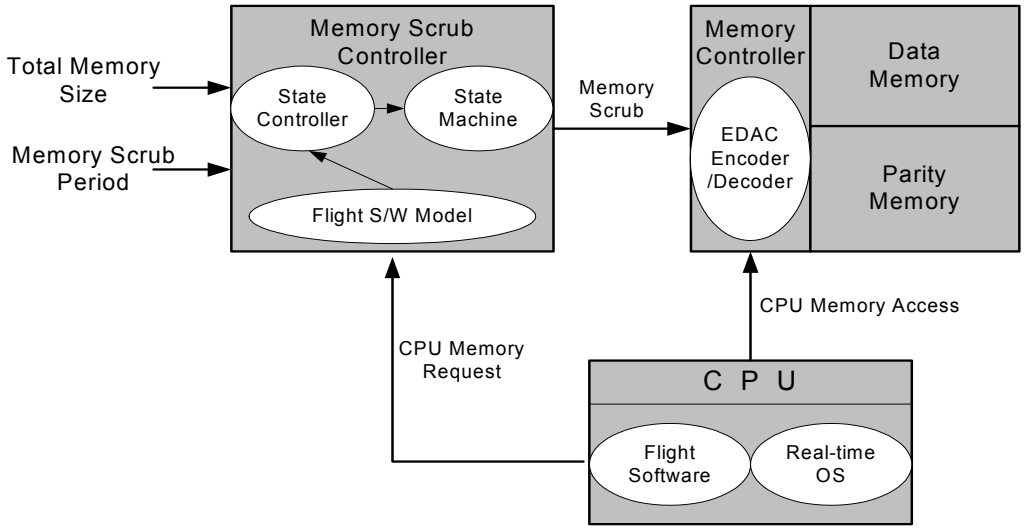

그림 2 메모리 스크럽 제어 시스템 개념도.
Fig. 2 Basic configuration of memory scrub control system.

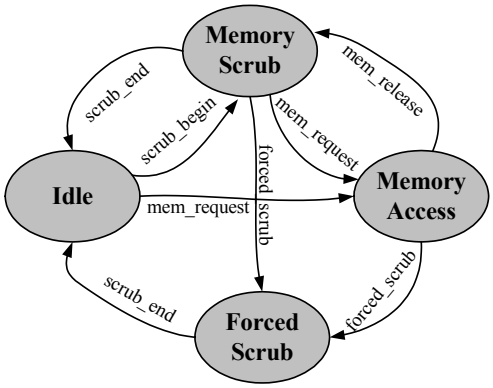

그림 3 스크럽 제어기를 위한 상태 머신.
Fig. 3 State machine for memory scrub controller.

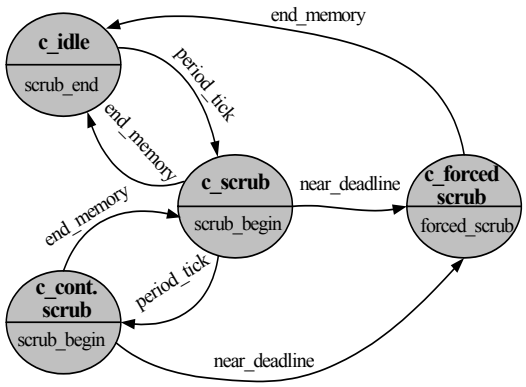

그림 4 상태 머신을 위한 상태 제어기.
Fig. 4 State controller for state machine.

표 1 상태 머신에 사용되는 이벤트의 종류.
Table 1 Events used in the state machine.

	이벤트 종류	설명
제어 불가능 이벤트 (uncontrollable event)	mem_request	CPU 또는 외부기기의 메모리 요구
	mem_release	CPU 또는 외부기기의 메모리 해지
제어 가능 이벤트 (controllable event)	scrub_begin	스크럽 시작
	scrub_end	스크럽 종료
	forced_scrub	스크럽 작업 강제 실행

그림 4는 이번 연구에서 제안한 상태 제어기이다. 상태 제어기는 “c_idle”, “c_scrub”, “c_cont. scrub”, 그리고 “c_forced scrub” 등 4개의 상태를 가진 무어 머신(Moore machine)으로 구성된다. 각 상태에서는 표 1의 제어 가능 이벤트 신호 scrub_begin, scrub_end, forced_scrub 중 하나를 발생시킨다. “c_scrub”은 스크럽 작업을 실행하는 상태를 나타내고, “c_cont. scrub”은 현재 주기 이내에 전체 메모리에 대한 스크럽 작업이 완료되지 못하여 다음 주기에도 스크럽 작업을 계속하는 상태를 가리킨다. “c_scrub”과 “c_cont. scrub” 상태에서는 scrub_begin 이벤트를 발생시켜 스크럽 작업을 계속하도록 한다. “c_forced scrub” 은 스크럽 제한 시간이 가까워졌는데도 불구하고 현재 진행 중인 전체 메모리에 대한 스크럽 작업이 완료되지 못했을 때 강제로 CPU 또는 외부 기기의 메모리 접근을 제한하고 오직 스크럽 작업만을 강행하도록 하는 상태이다. 이 상태에서는 forced_scrub 이벤트를 발생시켜 그림 3의 상태 머신이 “Forced Scrub” 상태로 천이하게 한다. “c_idle”은 스크럽 작업이 주기 이내에 완료된 상태를 나타내며 scrub_end 이벤트를 발생시켜 스크럽 작업을 종료시킨다.

상태 제어기는 period_tick, near_deadline, end_memory 이벤트에 의해 구동되고 상태 변화가 이루어진다. 이들 이벤트에 대한 설명을 표 2에 나타내었다. period_tick, near_deadline 이벤트는 매 스크럽 주기마다, 또는 데드라인

근처에서 펄스 신호를 발생시키는 회로(타이머 회로)로부터 만들어 진다. period_tick, near_deadline 이벤트를 생성하는 펄스 발생기는 다음 절에서 자세히 설명한다. end_memory 이벤트는 스크럽 해야 하는 전체 메모리의 마지막 주소(address)에 도달했을 때 발생하는 이벤트이다. 스크럽은 메모리의 시작 주소부터 마지막 주소까지 차례로 행해지고, 마지막 주소에 도달하면 다시 처음 주소로 되돌아가 이를 반복한다.

표 2 상태 제어기의 천이에 사용되는 이벤트의 종류.
Table 2 Events used in the transition of the state controller.

이벤트 종류	설명
period_tick	매 스크럽 주기마다 발생되는 타이머 이벤트
near_deadline	시간 제한(데드라인)에 근접했음을 알려주는 이벤트
end_memory	메모리의 마지막 주소(address)에 도달했음을 나타냄

3.2 주기 및 데드라인 근접 이벤트 발생기 설계

period_tick 이벤트는 메모리 스크럽 제어 시스템의 매 스크럽 주기마다 발생하는 타이머 이벤트이다. 상태 제어기는 이 이벤트를 이용하여 스크럽 작업을 시작하고, 전체 메모리에 대한 스크럽이 완료되면 "Idle" 상태로 천이한 후 다음 period_tick이 발생하면 스크럽 작업을 재개한다. 따라서 period_tick마다 전체 메모리에 대한 스크럽 작업이 반복된다. 다시 말하면 각 메모리 셀이 스크럽되는 주기가 바로 period_tick의 주기와 같다. period_tick 이벤트는 그림 5에서와 같이 타이머를 이용하여 설계하였다. 주기 설정 값(period set)은 외부에서 설정 가능하도록 하였다. 주기 설정 값에 해당하는 시각마다 이벤트 발생기에서 펄스(pulse)가 발생하는데 이것이 period_tick 이벤트이다.

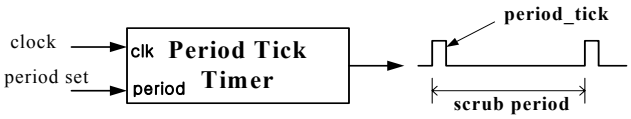

그림 5 period_tick 이벤트 발생기.
Fig. 5 Event generator for period_tick.

near_deadline 이벤트는 스크럽 작업이 완료되어야 하는 최대 시간제한(데드라인)에 근접하였음을 알려주는 이벤트이다. 일반적으로 스크럽 주기는 데드라인보다 짧게 설정된다. CPU 등 외부 기기의 메모리 접근이 빈번이 일어날 경우 스크럽 작업은 지연되고 몇 개의 스크럽 주기를 넘길 수 있지만 데드라인을 넘겨서는 안 된다. near_deadline은 scrub_begin 이벤트로 스크럽 작업이 시작된 후 경과한 시간이 실제 데드라인(actual deadline)에 도달하였을 때 발생한다. 외부 기기의 메모리 요구가 없을 때 전체 메모리를 한번 스크럽 하는 데 걸리는 실제 시간을 t_scrub라고 하면,

actual deadline = deadline − t_scrub

로 계산된다.

그림 6은 구현된 near_deadline 이벤트 발생기를 나타낸다. scrub_begin 이벤트에 의해 발생기 내부 타이머가 리셋(reset) 되고 시작(start)되며, scrub_end 이벤트가 발생하면 타이머는 정지(stop)된다. 타이머가 리셋 되어 동작된 후 actual deadline 시간이 경과할 때까지 scrub_end가 발생하지 않으면 near_deadline 펄스가 발생한다. near_deadline 이벤트가 발생하면 상태 제어기는 "c_forced scrub" 상태로 천이하여 forced_scrub 이벤트를 발생시키고, 이는 상태 머신을 "Forced Scrub" 상태로 천이시켜 외부 메모리 요구와 상관없이 오직 스크럽 작업만을 수행하도록 한다.

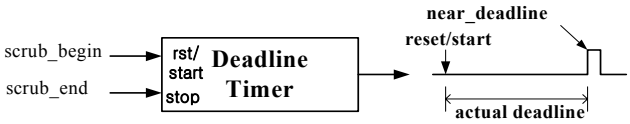

그림 6 near_deadline 이벤트 발생기.
Fig. 6 Event generator for near_deadline.

그림 7은 매 주기마다 발생하는 period_tick에 따라 스크럽 작업이 수행되는 형태를 보여준다. 매 period_tick마다 새로 스크럽 작업을 수행하고 다음 주기 이전에 전체 메모리에 대한 스크럽 작업이 완료되면 "Idle" 상태로 천이하여 대기한다. 스크럽 중간에는 CPU 등 외부 기기의 메모리 요구에 의해 스크럽 작업이 선점(preemption) 당할 수 있다. 외부 메모리 요구가 스크럽 작업보다 우선순위가 높으므로 외부 메모리 요구를 먼저 처리하고 나머지 여유 시간에 스크럽 작업을 수행한다.

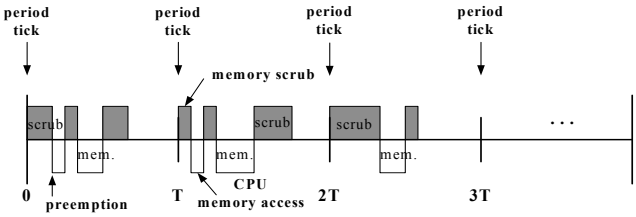

그림 7 period_tick을 이용한 스크럽 작업 수행.
Fig. 7 Scrubbing procedure using period_tick event.

그림 8은 외부 메모리 요구 시간이 많아서 period_tick 주기 이내에 스크럽 작업을 완료하지 못하여 다음 주기를 넘어서까지 계속되는 경우를 보여준다. 이때는 다음 주기가 시작되는 시점(그림 8의 nT 시점)에 period_tick 이벤트에 의해 상태 제어기의 상태가 "c_cont. scrub" 상태로 천이하고, 이전 주기부터 계속된 스크럽이 완료됨을 알려주는 end_memory 이벤트가 발생하면 "c_scrub" 상태로 천이하여 다음 스크럽 작업을 바로 시작한다.

그림 9는 스크럽 작업이 데드라인에 근접할 때까지 완료되지 못하여 near_deadline 이벤트가 발생하는 경우의 스크럽 형태를 보여준다. 스크럽 시작 후 실제 데드라인 동안 스크럽 작업이 완료되지 못하면 near_deadline 이벤트가 발생하고 이는 상태 머신을 "Forced Scrub" 상태로 천이하게 하여 외부 메모리 요구를 제한하고 스크럽 작업만을 수행하도록 한다.

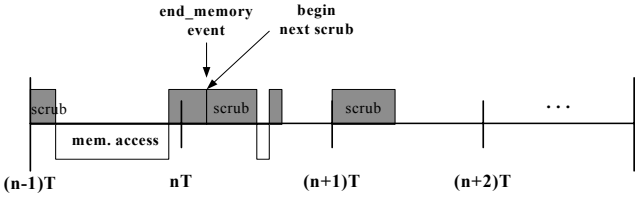

그림 8 주기를 넘겼을 때 지속적 스크럽 형태.
Fig. 8 Continuation of scrubbing exceeding a period.

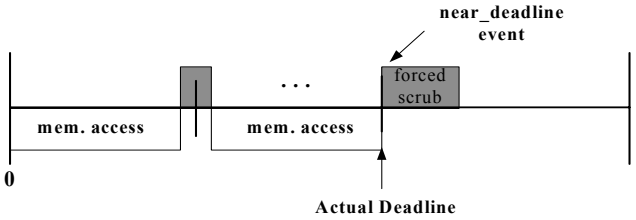

그림 9 near_deadline 이벤트 발생 및 강제 스크럽 형태.
Fig. 9 Generating near_deadline event with forced scrubbing.

3.3 메모리 스크럽 과정

스크럽 제어기의 상태 머신이 “Memory Scrub” 상태에 있으면 메모리 스크럽 작업을 수행한다. 메모리 스크럽 작업은 normal과 burst 모드 등 두 가지 모드가 가능하다. normal 모드에서는 한 워드(word)씩 메모리 데이터(패리티 비트 포함)를 읽어(memory read)와 EDAC(error detection and correction) 로직(logic)을 통과시킨다. EDAC 로직을 통과하면서 메모리에서 발생한 오류는 정정되고 출력은 오류 없는 값이 된다. 하지만 실제 메모리에는 오류가 정정되지 않은 데이터가 그대로 남아 있으므로 EDAC 회로의 출력 값, 즉 오류가 정정된 값을 한 번 읽은 번지(address)에 다시 쓰는(memory write) 과정을 거친다. 이것이 메모리의 오류를 정정하는 스크럽 과정이다.

그림 10은 normal 모드에서의 메모리 스크럽 과정을 보여준다. memory write 후에는 다음 메모리의 스크럽을 위해 메모리 번지 값을 증가 시키고(increment memory address), 증가시킨 번지 값이 메모리의 최종 번지(memory end address)를 넘어서는지 체크한다. memory end address 보다 크면 메모리 전체에 대한 스크럽이 끝났음을 알리는 end_memory 이벤트를 발생시키고 다음 스크럽 번지를 메모리 시작 번지로 설정한다(next addr <= start addr).

normal 모드에서는 외부 기기의 메모리 요구(memory request)에 빠르게 대응하기 위하여 memory read 후 외부 기기의 메모리 요구가 있는지를 체크한다. 메모리 요구가 있을 시에는 스크럽 작업을 중단하고 메모리 접근을 먼저 처리한다. 이것은 그림 3의 상태 머신에서 “Memory Scrub” 상태에서 mem_request 이벤트 발생 시 상태 머신이 “Memory Access” 상태로 빠르게 천이하게 함으로써 mem_request에 대한 지연시간(latency)을 줄이기 위함이다.

burst 모드는 burst 카운터에 설정된 분량의 메모리를 연속으로 스크럽 할 수 있는 기능이다. burst 모드 스크럽 도중 외부 기기의 메모리 요구는 무시되므로 burst 스크럽이 끝날 때까지 외부로부터의 메모리 접근이 지연된다. 그림 11은 burst 모드의 스크럽 과정을 나타낸다. burst 카운터가

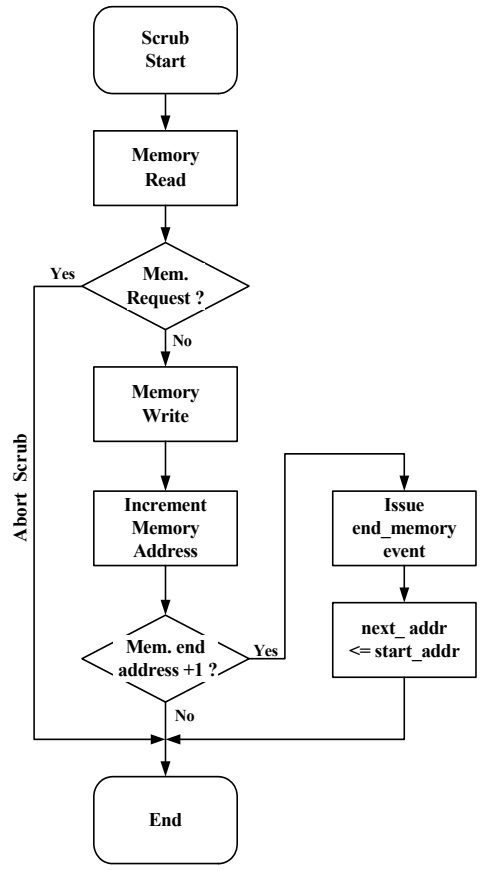

그림 10 normal 모드에서의 스크럽 과정.
Fig. 10 Scrubbing procedure in normal mode.

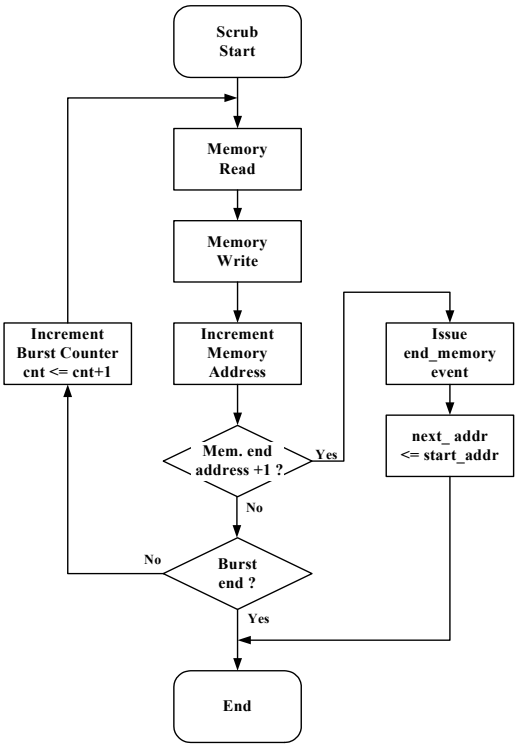

그림 11 burst 모드의 스크럽 과정.
Fig. 11 Scrubbing procedure in burst mode.

끝나거나 메모리 마지막 번지의 스크럽이 종료되면 burst 모드 스크럽이 종료된다. 이 모드는 메모리 접근 형태 모델과 결합하여 향후 일정 시간 동안 외부 기기의 메모리 접근 가능성이 낮은 것으로 판단될 때 적용된다. 예를 들어 메모리 접근 형태 모델로부터 향후 10번의 burst cycle 동안 외부 기기의 메모리 접근이 없을 확률이 0.9 이상으로 나왔다고 하자. 이 경우 burst 카운터 값을 10으로 설정함으로써 10번의 스크럽 작업을 연속해서 처리할 수 있다. 또한 이 방식은 전체 메모리에 대한 스크럽 시간을 줄이는 효과를 낸다.

4. 모의실험

앞 장에서 설계한 스크럽 제어기를 VHDL로 실제 구현하고 FPGA에 탑재하여 동작을 확인하였다. 그림 12와 그림 13은 FPGA 내에서 동작하는 스크럽 제어기의 상태와 이벤트 신호를 외부로 출력하여 오실로스코프로 측정한 결과이다. 각 그림 상단에서 3~4번째 신호 Machine State[1], Machine State[0]은 그림 3의 머신의 상태를 나타내는 2 bit 값이다(00:Idle, 10:Memory Access, 01:Memory Scrub, 11:Forced Scrub). 하단의 두 신호 Controller State[1], Controller State[0]은 그림 4의 상태 제어기를 나타내는 2 bit 값이다(00:c_idle, 10:c_scrub, 01:c_cont. scrub, 11:c_forced scrub).

그림 12는 스크럽 주기의 시작을 알려주는 period_tick 이벤트(그림의 제일 상단 신호)가 발생한 후 스크럽 제어기의 동작을 보여준다. 4μsec에서 period_tick 이벤트가 발생하면 상태 제어기는 “c_idle”에서 “c_scrub”로 천이하고 scrub_begin 신호를 발생시킨다(그림 12 하단 3개 신호 참조). 이때 상태 머신은 외부로부터 메모리 요구 신호(mem_request)에 의해 “Memory Access” 상태에 있다. 메모리 요구(mem_request)를 처리하는 것이 우선이므로 scrub_begin 이벤트가 상태 머신의 상태를 “Memory Access”에서 “Memory Scrub”로 천이시키는 시점은 mem_request 신호가 끝나는 7μsec 부근이다. 이후 mem_request가 없는 구간에서는 “Memory Scrub” 상태에서 스크럽 작업을 수행하며, mem_request가 있으면 “Memory Access” 상태로 천이한다(12~20μsec 구간 참조).

그림 13은 near_deadline 이벤트(그림의 제일 상단 신호)가 발생한 후 스크럽 제어기의 동작을 나타낸다. 4μsec에서 near_deadline 이벤트가 발생한 후 상태 제어기는 곧바로 강제 스크럽 상태(“c_forced scrub”)로 천이하여 forced_scrub 이벤트를 발생시킨다. forced_scrub 이벤트는 mem_request가 있음에도 불구하고 상태 머신의 상태를 “Memory Access” 상태에서 “Forced Scrub” 상태로 천이시켜 전적으로 스크럽 작업을 수행하도록 한다. 메모리 끝에 도달했음을 나타내는 end_memory 이벤트가 발생할 때까지 이 상태는 계속 유지된다(4~10μsec 구간). 또 강제 스크럽 작업이 끝난 후 외부 메모리 요구를 수행한다(14μsec 이후 구간).

본 연구에서 개발된 스크럽 제어기의 단순화된 버전은 과학기술위성 3호 대용량 메모리 모듈에 적용되었다. 대용량 메모리 모듈의 비행 모델(flight model)에 대한 기능 및 환경 시험에서 스크럽 제어기의 정상 동작 여부와 실용성을 확인할 수 있었다.

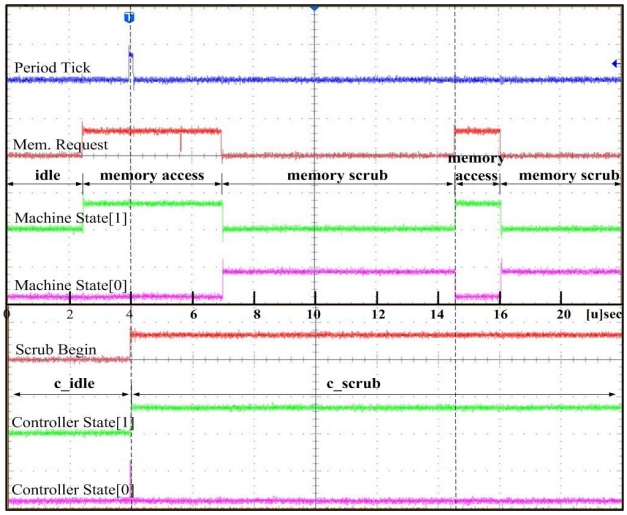

그림 12 Period tick에 의한 스크럽 제어기 동작.
Fig. 12 Operation of scrub controller induced by period tick.

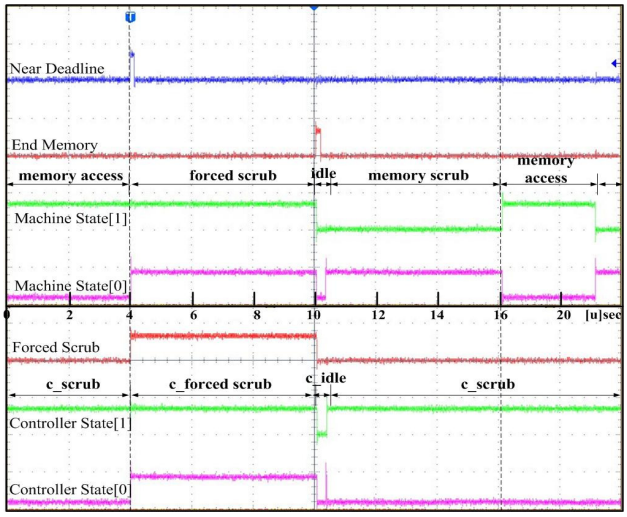

그림 13 Near deadline 이벤트에 의한 스크럽 제어기 동작.
Fig. 13 Operation of scrub controller induced by near deadline event.

5. 결 론

본 연구에서는 우주용 메모리에서 발생하는 SEU의 누적을 방지하기 위하여 자동으로 오류를 탐지하고 복구하는 스크럽 알고리즘을 개발하고 이를 VHDL와 FPGA로 구현하였다. 제안된 스크럽 제어 시스템에서 비행 소프트웨어의 메모리 접근 형태는 포아송 모델로 표현되었고, 제어기는 CPU의 메모리 접근을 고려하여 스크럽 작업을 수행한다. 스크럽 알고리즘은 상태 머신과 이것을 제어하기 위한 상태 제어기에 의해 수행되도록 구성되며, 주어진 주기 이내에 전체 메모리의 스크럽이 완료되는 실시간 조건을 만족하도록 하였다. VHDL과 FPGA로 구현된 스크럽 제어기가 보이는 각 이벤트들의 발생 및 처리 과정은 제안된 제어 시스템의 우수성을 입증한다.

본 연구에서 개발된 고에너지 입자에 의한 SEU 자동 극

복 기술은 우주용 메모리뿐 아니라 우주 환경과 비슷하게 방사능 영향을 크게 받는 원자력 발전소 내부의 제어 컴퓨터 메모리 및 원자로용 로봇 제어 컴퓨터의 설계에도 동일하게 적용될 수 있을 것이다.

감사의 글

이 논문은 2008년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (KRF-2008-313-D00176)

참 고 문 헌

[1] S. Yoshioka, H. Kamimura, M. Akiyama, M. Nakamura, T. Tamura, and S. Kuboyama, “A radiation-hardened 32-bit mirco-processor based on the commercial CMOS process,” IEEE Transactions on Nuclear Science, vol. 41, no. 6, pp. 2481-2486, 1994.

[2] T. P. Ma and P. V. Dressendorfer, Ionizing Radiation Effects in MOS Devices and Circuits, John Wiley & Sons, 1989.

[3] 곽성우, 류상문, 박홍영, 오대수, 유관호, 최병재, 김병국, “과학위성1호 탑재 컴퓨터의 설계 및 구현”, 한국항공우주학회지, 제31권 4호, pp. 105-111, 2003.

[4] 곽성우, 박홍영 “과학기술위성 1호 탑재 컴퓨터에서의 SEUs 극복을 위한 메모리 운용 및 해석”, 한국항공우주학회지, 제32권 1호, pp. 98-105, 2004.

[5] K. H. Kim, H. S. Kim, J. H. Park, K. H. Park, and S. D. Choi, “Design and implementation of the small satellite on-board computer system: KASCOM,” Journal of Astronomy and Space Sciences, vol. 13, no. 2, pp. S52-S66, 1996.

[6] C. I. Underwood, “In-orbit radiation effects monitoring on the UoSAT satellite,” 4th Annual AIAA/USU Conference on Small Satellite, 1990.

[7] C. M. Krishina and K. G. Shin, Real-Time Systems, McGraw-Hill, 1997.

저 자 소 개

곽 성 우 (郭 成 祐)

1970년 3월 10일생. 1993년 2월 한국과학기술원 전기 및 전자공학과 졸업(학사). 1995년 2월 한국과학기술원 전기 및 전자공학과 졸업(석사). 2000년 한국과학기술원 전기 및 전자공학과 졸업(공박). 2000년～2002년 인공위성연구센터 선임연구원, 연구교수. 2003년～현재 계명대학교 전자공학과 부교수. 주관심분야: 실시간 시스템, 비동기 시스템 제어, 우주용 디지털 시스템 설계 등.

Tel : 053-580-5926

Fax : 053-580-5165

E-mail : ksw@kmu.ac.kr

양 정 민 (楊 正 敏)

1971년 3월 31일생. 1993년 2월 한국과학기술원 전기 및 전자공학과 졸업(학사). 1995년 2월 한국과학기술원 전기 및 전자공학과 졸업(석사). 1999년 2월 한국과학기술원 전기 및 전자공학과 졸업(공박). 1999년 3월～2001년 2월 한국전자통신연구원 컴퓨터·소프트웨어연구소 선임연구원. 2001년 3월～현재 대구가톨릭대학교 전자공학과 부교수. 주관심분야: 비동기 순차 머신 제어, 실시간 시스템 등.

Tel : 053-850-2736

Fax : 053-850-2704

E-mail : jmyang@cu.ac.kr