Journal of Korea Multimedia Society Vol. 18, No. 6, June 2015(pp. 723-733)
http://dx.doi.org/10.9717/kmms.2015.18.6.723

STM32F4 기반의 실시간 임베디드 시스템의 가동시간 향상을 위한 긴 꼬리 와치독 타이머 기법

최하연[†], 윤지완[††], 박서연[†††], 김예솔[††††], 박상수[†††††]

Long-Tail Watchdog Timer for High Availability on STM32F4-Based Real-Time Embedded Systems

Hayeon Choi[†], Jiwan Yun[††], Seoyeon Park[†††], Yesol Kim[††††], Sangsoo Park[†††††]

ABSTRACT

High availability is of utmost importance in real-time embedded systems. Temporary failures due to software or hardware faults should not result in a system crash. To achieve high availability, embedded systems typically use a combination of hardware and software techniques. A watchdog timer is a hardware component in embedded microprocessors that can be used to automatically reset the processor if software anomalies are detected. The embedded system relies on a single watchdog timer, however, can be permanently disabled if the timer is not properly configured, e.g. falling into an indefinite loop. STM32F4 provides two different types of watchdog timer in terms of timing accuracy and robustness. In this paper, we propose a hybrid approach, called long-tail watchdog timer, to utilize both timers to achieve self-reliance in embedded systems even though one of timers fails. Experimental results confirm that the proposed approach successfully handles various failure scenarios and present performance comparisons between single watchdog timer and hybrid approach in terms of configuration parameters of watchdog timers in STM32F4, counter value and window size.

Key words: Watchdog Timer, STM32F4, High Availability, System Failure, Hard Reset

1. 서 론

임베디드 시스템은 특정 기능을 반복적으로 수행하는 컴퓨터 시스템으로 대부분의 임베디드 시스템은 기능의 수행이 주어진 시간 내에 완료되어야 하는 실시간 제약을 갖는다. 오늘날의 임베디드 시스템은 유연하게 설계되어 다양한 요구사항을 만족시킬 수 있으며 일상생활에 쓰이는 기기들을 제어하는데 사용된다[1-3]. 크기가 작은 임베디드 시스템은 스마트 안경, 스마트워치 등의 웨어러블 기기로 대표되는 사물 인터넷(IoT)을 포함하여 다양한 산업분야에 적용되고 있다[4,5]. 이러한 응용에 탑재되는 임베디드 시스템은 크기가 작고 사용자가 요구하는 정보 등을 정확하게 지속적으로 제공하는 것이 필수적이다. 지

※ Corresponding Author : Sangsoo Park, Address: (120-750) 52 Ewhayeodae-gil, Seodaemun-gu, Seoul, TEL : +82-2-3277-6643, FAX : +82-2-3277-2306, E-mail : sangsoo.park@ewha.ac.kr
Receipt date : Mar. 10, 2015, Revision date : Apr. 2, 2015
Approval date : Apr. 13, 2014
[†] Dept. of Computer Science and Engineering, Ewha Womans University (E-mail : hayeon.choi@ewhain.net)
[††] Dept. of Computer Science and Engineering, Ewha Womans University (E-mail : ctladykiki@nate.com)
[†††] Dept. of Computer Science and Engineering, Ewha Womans University (E-mail : seoyeon9249@naver.com)
[††††] Dept. of Computer Science and Engineering, Ewha Womans University (E-mail : yaesol91@gmail.com)
[†††††] Dept. of Computer Science and Engineering, Ewha Womans University
※ This work was supported by the National Research Foundation of Korea Grant funded by the Korean Government(NRF-2014S1A5B6037290).

속적인 기능의 수행, 즉 높은 가동시간을 제공하기 위해서 부가적인 하드웨어를 추가하는 것이 어려우며, 소프트웨어 코드의 크기와 성능에 민감하다는 특징을 갖고 있다.

일시적인 결함이 발생하더라도 시스템의 정상적인 수행이 지속되거나 주어진 시간 이내에 정상적인 수행이 가능한 상태로 복구되는 시스템을 결함 감내 시스템(Fault-tolerant system)이라 한다. 이러한 결함 감내 시스템에서 결함을 복구 방법으로는 이중화 시스템(Fault-tolerance by replication), 동작 지속(No Single Point of Repair), 고장 분리(Fault Isolation to the Failing Component), 고장 전파 억제(Fault Containment) 기법 등이 있다[6,7]. 이러한 방법들은 추가적인 하드웨어나 소프트웨어가 필요하기 때문에 제한된 자원을 갖는 소형 임베디드 시스템에는 적용하기 어렵다.

대부분의 상용 임베디드 프로세서는 높은 가동시간을 지원하기 위해서 와치독 타이머(WDG: Watchdog Timer)가 내장되어 있다[8,9]. STM32F4[10] 임베디드 프로세서는 ARM사의 Cortex-M4[11] 코어 기반의 소형 임베디드 시스템을 위한 임베디드 프로세서로서 다양한 사물인터넷 기기의 핵심요소로 적용되고 있다. STM32F4는 시간 정밀도와 결함 복원도에서 서로 다른 특성을 갖는 두 개의 WDG인 IWDG(Independent WDG)와 WWDG(Window WDG)를 내장하고 있다.

일반적인 WDG는 주기적인 타이머로 동작되며 완료되기 전에 갱신하지 않으면 시스템을 리셋 하는 기능을 갖고 있다. 즉, 시스템이 일시적인 결함으로 정상적으로 수행이 되지 않을 경우 WDG를 갱신하는 루틴을 수행하지 못하고 타이머의 주기에 도달하게 되면 시스템을 리셋하여 시스템 초기의 정상적인 수행이 가능한 상태로 복구한다. 그러나 WDG 소프트웨어를 적절히 구현하지 않으면 결함이 발생하였을 때 복구하는 한계가 있다. 예를 들어 WDG를 갱신하는 코드가 무한히 반복되고 소프트웨어 코드는 실행되지 못하는 경우 WDG는 계속 갱신이 되기 때문에 결함을 감지하고 복구할 수 없게 된다.

본 논문은 이와 같이 STM32F4의 WDG를 사용했을 때 결함이 발생할 수 있는 다양한 시나리오에 대해서 특성이 다른 두 개의 WDG를 결합하여 사용하도록 하여 빠르게 결함을 복구하면서 보다 다양한 상황에서 복구가 가능한 소프트웨어 기반의 가동시간 향상 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서 소형 임베디드 시스템에서 결함을 발견하고 가동시간을 높이는 관련 연구를 기술하고, WDG를 활용한 기존 연구와 비교한다. 3장에서는 본 논문에서 제안한 하이브리드 방식의 긴 꼬리 와치독 타이머에 대해 기술한다. 4장에서는 제안된 기법이 다양한 시나리오에 대해 복구가 가능한지 실험을 통해 보이고 시스템의 복구 성능에 영향을 주는 WDG 파라미터 값에 따른 실험 결과를 제시한다. 마지막으로 5장에서 본 논문의 결론을 내린다.

2. 관련 연구

임베디드 시스템에서 결함을 발견하고 가동시간을 높이는 관련 연구에서 [12]의 연구에서는 실시간 운영체제 기반으로 동작하는 시스템에서 응용을 수행하기 위한 태스크 외에 대기 중인 있는 태스크를 추가하여 수행 중인 태스크에 결함이 발생하였을 때 이를 대체하여 복구하는 기법을 제안하였다. 실시간 시스템에서 한 태스크에서 결함이 발생했을 때 이를 복구하기 위해서 태스크를 다시 시작하게 되면 다른 태스크의 시간 제약성까지 영향을 미칠 수 있다. 이러한 문제점을 극복하기 위해 시스템에 유휴시간을 제공하여 오류가 발생한 태스크 대신 대기 중인 태스크를 스케줄링 할 수 있도록 확장한 RM(Rate-Monotonic) 스케줄링 기법을 적용하였다.이와 같이 대기 중인 태스크를 추가하여 결함이 발생하였을 때 대체하여 스케줄링하는 여러 연구가 있지만 CPU, 메모리 등의 자원 면에서 오버헤드가 높은 단점이 존재한다.

[13]에서는 기존의 WDG의 하드웨어에서 너무 빨리 시스템을 리셋 시킬 가능성이 있는 표준 WDG와 정해진 시간 내에서만 리셋이 가능한 윈도우 기반 WDG에서의 단점을 개선하여 하드웨어적으로 구현한 순서화된 WDG (Sequenced WDG)를 제안하였다. [14]에서는 일반적으로 시스템에서 단일 WDG가 사용되기 때문에 여러 태스크가 수행되는 실시간 운영체제 기반의 시스템에서 다중의 WDG를 사용하여 결함이 발생한 태스크를 추적하여 신뢰성을 높이는 기법을 제안하였다. 또한 단일 WDG를 사용하는 시스템에서 해당 타이머가 오동작하는 경우에 이를 감

지하여 WDG 자체를 초기화할 수 있는 WDG의 WDG를 구현하는 기법이 제안되기도 하였다[15].

이와 같이 스케줄링 기법으로 가동시간을 향상 시키는 연구는 추가적인 하드웨어 혹은 소프트웨어 코드가 필요하기 때문에 자원이 제한된 소형 임베디드 시스템에 적합하지 않다. [14]는 표준과 윈도우 기반의 WDG의 단점을 개선하는 점은 본 논문의 접근 방법과 유사하지만 새로운 형태의 단일 WDG를 제안하는 연구이기 때문에 상용 임베디드 프로세서에 적용하는데 한계가 있다.

3. 긴 꼬리 와치독 타이머 기법

STM32F4 임베디드 프로세서는 서로 다른 특성을 갖는 IWDG(Independent Watchdog)와 WWDG (Window Watchdog)을 탑재하고 있다[10]. 본 논문에서 제안하는 두 WDG의 하이브리드 방식인 긴 꼬리 와치독 타이머 기법을 기술하기 위해 먼저 두 WDG의 작동 원리와 장점과 단점을 설명하고 짧은 시간과 높은 복원력을 모두 만족하기 위한 결합 방법을 제시한다.

3.1 IWDG(Independent Watchdog)

IWDG는 최대 180Mhz로 동작하는 프로세서의 코어 클럭과 독립적인 상대적으로 매우 느린 32kHz 클럭을 사용한다. IWDG는 프로세서 코어와 독립적으로 동작하므로 코어에 문제가 생기더라도 정상적인 수행을 할 수 있다[16]. 일반적인 WDG과 같이 지정된 주기에 대해 시스템 리셋을 수행되며 응용 프로그램과 완전히 독립적으로 수행하는 경우에 적합하지만, 클럭의 해상도가 매우 낮기 때문에 빠른 시간 내에 결함을 감지하고 시스템을 리셋 하는데 적합하지 않다.

Fig. 1의 IWDG 초기화 코드에서 IWDG_RLR로 설정된 다운 카운터 값이 0으로 떨어지기 전에 카운터가 재설정되지 않으면 IWDG는 프로세서 코어를 리셋하게 된다. 예를 들어 IWDG_RLR이 10000으로 설정되었을 경우 프로그램 코드가 매 10000/(32kHz) ≒ 312ms 마다 IWDG를 재설정하지 않으면 시스템이 리셋 되게 된다.

```
void IWDG_Init(void) {
  IWDG->KR = 0x5555;        // Enable write to PR, RLR
  IWDG->PR = IWDG_PR;       // Init prescalar
  IWDG->RLR = IWDG_RLR;     // Init RLR
  IWDG->KR = 0xAAAA;        // Reload the Watchdog
  IWDG->KR = 0xCCCC;        // Start the Watchdog
}
```

Fig. 1. Initialization Process of Independent Watchdog (IWDG).

3.2 WWDG(Window Watchdog)

WWDG는 카운터 값이 0이 되면 프로세서 코어를 리셋하는 기능은 IWDG와 동일하지만 다른 점은 카운터 값이 재설정될 수 있는 시점이 Fig. 2에서 설정된 윈도우 값인 WWDG->CFR의 값 보다 작고 0x3f 보다 클 때에만 가능하다는 점이다[10,16]. 이러한 윈도우 기반의 WWDG는 시스템 외부의 간섭이나 예측하지 못한 상황에서 코드가 실행되는 소프트웨어의 결함을 감지하고 복구하는데 사용된다. 지정한 윈도우의 값에 따라 카운터가 재설정 되어야 하는 시간이 정해지므로 높은 시간 정밀도가 요구된다.

특히 WWDG는 소프트웨어가 카운터의 초기화가 가능한 구간을 지난 이후인 0~0x3f 사이에 2단계의 결함 복구가 가능하도록 한다. Fig 3과 같이 WWDG는 카운터 값이 0×40이 되는 순간 EWI(Early wake-up interrupt)를 발생시켜 수행되던 응용 프로그램을 정지하고 인터럽트에 호출된 WWDG_Interrupt 인터럽트 핸들러에서 해당 카운터 구간에서 결함을 복구하고 카운터 초기화하여 시스템이 리셋 되는 것을 방지할 수 있다.

3.3 IWDG와 WWDG이 결합된 긴 꼬리 와치독 타이머

일반적인 소형 임베디드 시스템은 Fig. 4(a)와 같이 응용 프로그램이 하나의 무한루프 내에서 필요한 기능을 수행하고 단일 WDG가 최악의 경우 프로그램의 수행 시간으로 주기가 정해지는 것이 일반적이

```
void WWDG_Init(void) {
  RCC->APB1ENR |= (1<<11); // Enable WWDG clock
  WWDG->CFR |= (3<<7);     // Configure the WWDG prescalar
  WWDG->CFR |= 127;        // Configure the WWDG refresh window
  WWDG->CR |= (1<<7);      // Set the WWDG counter value and start
  WWDG->CFR |= (1<<9);     // Enable the Early wakeup interrupt
  WWDG->CR |= 126;         // Refresh the WWDG counter
}
```

Fig. 2. Initialization Process of Window Watchdog (WWDG).

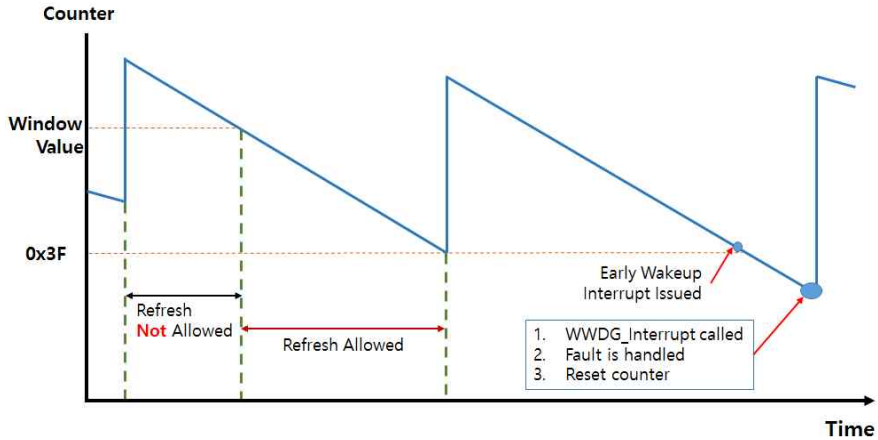
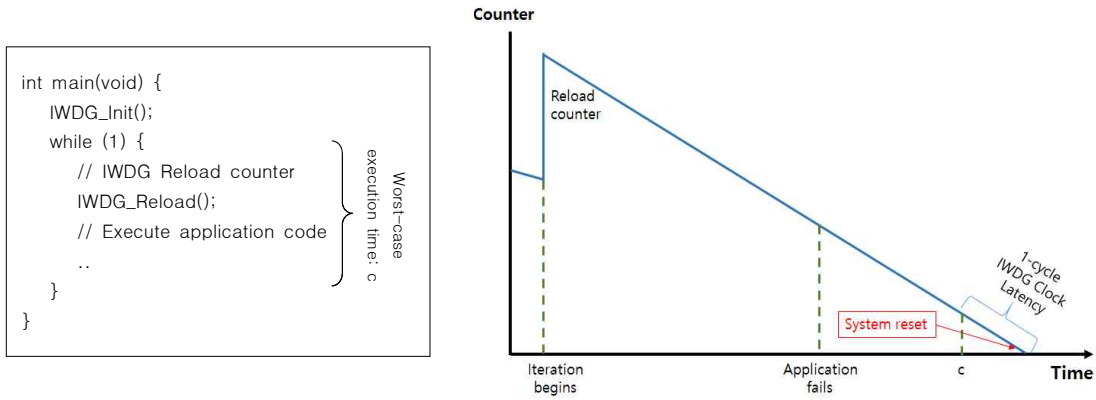


Fig. 3. Window Watchdog timing diagram.



(a) Pseudo code (b) Changes of counter values over time and event

Fig. 4. Typical IWDG operations in small-scale embedded systems.

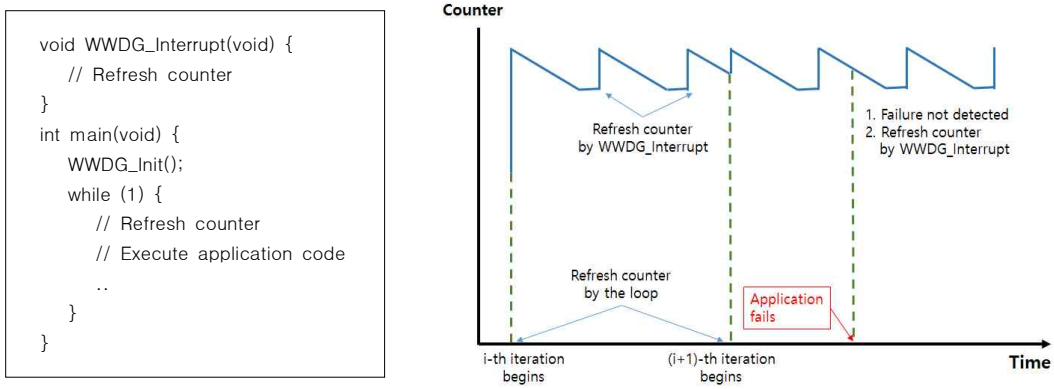
다. 응용 프로그램이 수행될 수 있는 최대 주기가 끝나면 WDG은 이를 결함으로 감지하여 시스템 초기화되어 복구된다.

IWDG를 사용하는 경우 Fig. 4(b)에서 볼 수 있듯이 응용 프로그램의 수행 중에 발생한 결함으로 최악의 경우에 허용되는 수행 시간 c를 지나고 한 클럭 이후 IWDG가 결함을 감지하고 시스템을 리셋한다. 이때 32kHz의 클럭을 갖는 STM32F4의 경우 1/(32kHz)≒32μs의 지연시간을 갖는다. 반면 WWDG의 경우 80Mhz~180Mhz로 동작하는 클럭을 사용하기 때문에 IWDG에 비해 최소 약 20000배의 시간 정밀도를 갖는 결함 감지와 복구가 가능하다.

WWDG의 입력 클럭은 프로세서 코어와 공유된다. 특히 입력 클럭은 주파수와 동작 여부를 소프트웨어로 초기화하도록 되어 있기 때문에 초기화가 잘못되면, 클럭의 작동이 중단되거나 엉뚱한 주기로 동작할 수 있다. 즉, Fig. 5(a)의 응용 프로그램이 정상적으로 수행되지 못하는 상황에서 WWDG_Interrupt 인터럽트 핸들러에 의해 결함을 감지하지 못하기 때문에, Fig. 5(b)와 같이 시스템을 복구하는데 실패하게 된다.

일반적으로 소형 임베디드 시스템은 단일 WDG를 사용하게 되는데, STM32F4 기반의 임베디드 시스템을 개발할 때 IWDG 만을 사용하는 경우 프로세서 코어의 일시적인 결함에도 복구가 가능하지만 이에 따른 지연시간이 상대적으로 길게 된다. 반면 WWDG 만을 사용하는 경우 상대적으로 복구가 빨리되지만 응용 프로그램이 무한루프에 빠져있는 경우 결함을 감지하지 못해 복구가 어려운 단점이 있다.

따라서 본 논문에서는 두 가지 서로 다른 특징을



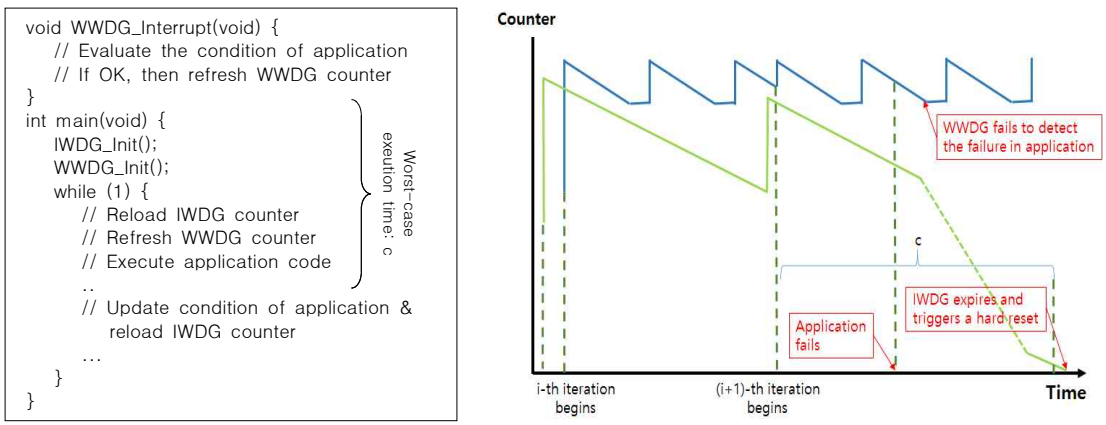
(a) Pseudo code (b) Changes of counter values over time and event

Fig. 5. WWDG fails to reset the hardware upon an application failure.

갖고 있는 WDG의 장점을 결합함으로써 일반적인 상태에서는 WWDG를 사용하여 결함의 감지와 복구가 빠르게 가능하게 하고, WWDG가 실패하는 경우에는 IWDG를 이용하여 응용 프로그램의 한 주기가 내에 결함을 감지하고 복구하는 방식인 긴 꼬리 와치독 타이머(Long-tail Watchdog Timer) 기법을 제안한다.

제안된 긴 꼬리 와치독 타이머 기법은 먼저 Fig. 6(a)과 같이 시스템이 초기화 될 때 서로 다른 주기를 갖도록 초기화 된다. WWDG가 IWDG에 비해 훨씬 빠른 클럭을 사용할 수 있기 때문에 IWDG의 주기는 응용 프로그램의 최악의 경우 수행시간인 c로 설정되고, 응용 프로그램의 한 주기 내에서 프로그램 수행 상태를 업데이트할 수 있는 시간 중에서 가장 긴 시간을 WWDG의 주기로 설정한다.

시스템이 시작되면 IWDG와 WWDG가 위와 같이 정해진 주기를 갖도록 초기화되고 카운트다운을 시작한다. 응용 프로그램은 하나의 루프로 수행되고 루프의 주기 내에 수시로 프로그램 수행 상태를 업데이트한다. 올바른 수행 상태인지 검사하기 위한 룰 기반의 assert() 기법 등을 사용하여 응용 프로그램이 결함을 발견하면 상태를 갱신한다. WWDG의 카운터 값이 0×40에 도달하게 되면 WWDG_Interrupt 인터럽트 핸들러가 호출되고 응용 프로그램의 수행이 중단된다. 만약 인터럽트 핸들러가 응용 프로그램이 결함 상태일 경우에는 카운터 값을 초기화하지 않아 WWDG에 의해 시스템이 리셋되도록 한다. 반대로 응용 프로그램의 상태가 정상적인 상태일 경우 카운



(a) Pseudo code (b) Changes of counter values over time and event

Fig. 6. Proposed Long-tail Watchdog Timer with an example to handle WWDG failure.

터 값을 초기화하고 인터럽트 핸들러의 수행을 끝내서 응용 프로그램으로 복귀할 수 있도록 한다.

WWDG는 응용 프로그램이 스스로 갱신하는 상태 정보에만 의존하기 때문에, 응용 프로그램이 상태 정보가 정상적인 수행 상태로 갱신된 상태에서 프로그램의 결함으로 무한루프에 빠지게 되면 프로그램은 더 이상 수행되지 못하지만 향후 호출되는 WWDG_Interrupt 인터럽트 핸들러는 마지막으로 보고된 수행 상태만을 검사하고 결함을 감지하지 못하게 된다. 즉, 응용 프로그램은 계속 무한루프를 수행하고 WWDG_Interrupt 인터럽트 핸들러는 결함을 감지하지 못하고 계속 카운터를 초기화를 하는 상태가 지속되게 된다.

이러한 문제점을 극복하기 위해서 제안된 긴 꼬리 와치독 타이머 기법에서는 무한루프에 빠진 시스템을 복구하도록 IWDG를 사용한다. Fig. 6(b)에서 응용 프로그램이 (i+1)번째 주기를 수행하는 중에 무한루프에 빠진 경우에도 시스템 초기화 과정에서 설정된 최악의 경우 수행 시간인 c 이후IWDG가 동작하여 응용 프로그램의 한 주기가 완료되어야하는 시간이 지났음에도 수행이 완료되지 못한 결함을 감지하고 시스템을 리셋하여 복구하게 된다.

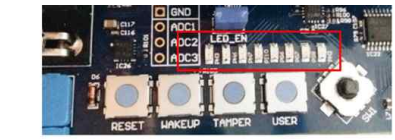
이와 같이 제안된 기법은 응용 프로그램 내에서 일시적인 계산 오류에 의해 발생하는 결함은 WWDG에 의해 신속하게 복구하도록 하고 응용 프로그램의 논리적인 오류로 더 이상 진행되지 못하는 결함은 IWDG에 의해 응용 프로그램의 다음 주기에는 복구되도록 하여 향상된 가동시간을 제공할 수 있다.

4. 성능 평가

본 논문에서 제안된 긴 꼬리 와치독 타이머가 다양한 경우에 발생할 수 있는 결함에 대해서 결함의 감지와 복구가 가능함을 보이기 위해서 가상의 결함을 주입하여 시스템의 복구 여부를 실험하였다. 또한, 복구 성능에 영향을 미치는 WDG의 설정 값인 윈도우 크기와 카운터 크기에 대해서 실험을 수행하여 성능을 평가 하였다. 실험에는 Keil사의 MCBSTM32F400 평가보드[17]를 사용하였으며 프로그램에서 Fig. 7과 같이 WDG에 의한 리셋 발생, 응용프로그램의 수행 여부, WWDG_Interrupt 인터럽트 핸들러의 수행 여부 등을 평가 보드의 LED를 이용하여 표시하도록 하여 주어진 환경에서 제안된 기법 동작 여부를 판별하였다.

4.1 WWDG_Interrupt 인터럽트 핸들러 단계의 실험

제안된 기법에서 가장 첫 단계로 결함이 감지되는 WWDG_Interrupt 인터럽트 핸들러에서 동작 여부를 실험하기 위해 Fig. 8과 같이 프로그램이 시작할 때 IWDG가 2초에 한 번씩 재설정이 되도록 설정하고 WWDG의 윈도우 값과 다운 카운터를 일반적인 경우인 127로 설정하여 실험을 수행하였다. 응용 프



WWDG reset	PH6	PH7	PI10	WWDG IRQ	Blinky	IWDG reset	Blinky
PH3				PG6	PG7	PG8	PH2

PH3	WWDG reset
PG6	WWDG_Interrupt
PG7	main()
PG8	IWDG reset
PH2	while ()

Fig. 7. Role of LEDs in the test scenarios.

1. Initialize IWDG per 2s
2. Initialize WWDG with window size 127 and counter 127

<A finite loop>

3. Reload IWDG
4. Refresh WWDG

<An infinite loop>

5. Reload IWDG only

6. WWDG counter reaches to 0x40
7. WWDG_Interrupt called
8. Virtual fault is handled
9. Reset counter

Fig. 8. Recovery process in the WWDG_Interrupt interrupt handler.

로그램의 수행 전반부에는 일정 시간 동안 IWDG와 WWDG 모두 제 시간에 재설정하도록 하여 두 WDG의 동작을 확인하였다. 응용 프로그램의 수행 후반부에는 가상의 결함을 주입하기 위해서 무한루프를 통해 IWDG의 카운터만 재설정하고 WWDG는 재설정하지 않아 WWDG_Interrupt 인터럽트 핸들러에 의해 카운터를 재설정하여 WWDG의 작동 여부를 판별하였다.

실험 결과 WWDG_Interrupt 인터럽트 핸들러가 작동하여 WWDG에 의한 리셋은 발생하지 않고 인터럽트 핸들러 안에서 가상의 결함이 감지되어 복구되는 것을 확인하였다.

4.2 WWDG 타임아웃 단계의 실험

WWDG의 카운터 값이 0으로 수렴하여 WWDG 타임아웃 단계에서 결함의 복구 여부를 위한 실험은 Fig. 9와 같다. 응용 프로그램의 결함으로 WWDG의 카운터가 0×40에 도달하여 WWDG_Interrupt 인터럽트 핸들러가 호출되지만 인터럽트 핸들러 자체가 문제가 있어 결함이 처리되지 않고 카운터가 초기화되지 못하는 경우에 제안된 기법은 시스템 초기화 과정에 설정하였던 WWDG 타이머에 의해 시스템 리셋이 수행되어 시스템 초기화 상태로 복구되는 것을 실험을 통해 확인하였다.

4.3 IWDG 단계의 실험

응용 프로그램이 수행되는 루프가 결함으로 더 이상 수행되지 못하는 경우일 때 WWDG_Interrupt 인터럽트 핸들러가 이 결함을 감지하지 못하면 WWDG 카운터가 0×40에 도달할 때마다 핸들러에 의해 카운터가 초기화되어 WWDG에 의한 시스템 리셋이 발생하지 않는다. 제안된 긴 꼬리 와치독 타이머가 이러한 경우에도 결함의 복구가 가능한지 확인하기 위해 Fig. 10의 실험을 수행하였다. 응용 프로그램이 무한루프에 빠지고 동시에 WWDG_Interrupt 인터럽트 핸들러가 결함을 감지하지 못하고 지속적으로 WWDG를 초기화하는 상황에도 긴 꼬리 와치독 타이머는 프로그램 초기에 설정하였던 IWDG에 의해 응용 프로그램이 수행되는 루프가 진행되지 못하고 있는 상태를 감지하고 IWDG 카운터가 0에 도달하게

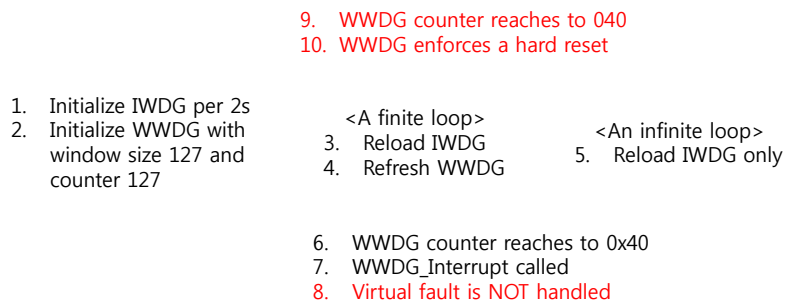


Fig. 9. Recovery process in the WWDG timeout.

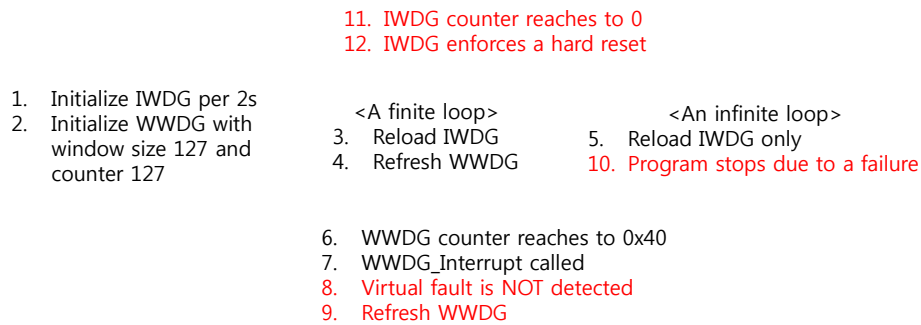


Fig 10. Test scenario 3.

되어 IWDG에 의해 시스템 리셋이 발생하고 결함을 복구하는 것을 확인하였다.

결함 발생으로 인해 응용 프로그램이 무한루프에 빠져서 더 이상 진행되지 못하는 상태에서는 WWDG_Interrupt 인터럽트 핸들러는 응용 프로그램이 여전히 수행 중이기 때문에 결함이라고 감지하기 못하게 된다. 그러나 시스템 시작 시에 초기화된 IWDG는 응용 프로그램 및 WWDG와 독립적으로 수행되기 때문에 WWDG가 감지하지 못한 프로그램의 결함을 감지하고 시스템 리셋을 수행한다.

4.4 단일 WDG와 제안된 기법의 성능 비교

임베디드 시스템에서 단일 WDG를 사용하는 경우와 본 논문에서 제안된 기법의 성능 비교를 위해서 응용 프로그램의 수행 주기를 설정하고 프로그램의 수행 상태를 매 33ms로 갱신하는 시스템을 구성하고 1~100ms 사이의 임의의 시간에 가상의 결함을 주입하여 (1) 33ms로 주기가 설정된 단일 WWDG를 사용하는 경우 (2) 100ms로 주기가 설정된 단일 IWDG를 사용하는 경우 (3) 본 논문의 긴 꼬리 와치독 타이머를 (1),(2)의 설정값을 사용하는 경우에 대해서 결함의 주입 시점부터 복구가 되는 시점까지의 시간을 비교하는 실험을 수행하였다.

Fig. 11의 실험 결과를 보면 (1)의 단일 WWDG를 사용하는 경우 약 60%의 결함이 10ms 이내에 복구되는 것을 볼 수 있고 30ms 이내에 91%의 결함이 복구되는 것을 알 수 있다. 단일 WWDG를 사용하는 경우 3.3절에서 기술한 바와 같이 응용 프로그램이 진행되지 않지만 WWDG가 이를 감지하지 못해 결함을 복구하는데 실패하는 경우가 발생하고 이 실험

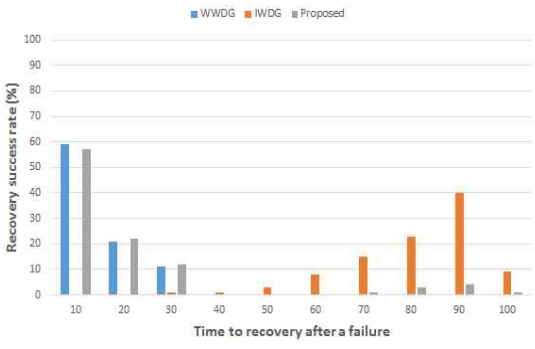


Fig. 11. Performance comparisons of WWDG, IWDG, and the proposed approach.

에 약 9%의 결함이 감지되지 못하는 것을 알 수 있다. (2)의 단일 IWDG를 사용하는 경우 (1)과는 달리 모든 결함이 복구가 되지만, 50%의 결함이 90ms 이후에 복구되어 WWDG를 사용하는 경우에 비해 결함의 처리가 지연되는 것을 알 수 있다.

본 논문에서 제안된 기법의 경우 WWDG가 결함 감지에 실패한 경우 IWDG가 결함을 복구하기 때문에 단일 IWDG를 사용하는 것과 같이 모든 결함을 복구하는데 성공한다. 또한, 91%의 결함이 WWDG에 의해 빠르게 복구되고 나머지 9%의 결함만이 IWDG에 의해 70ms 이후에 처리되는 것을 확인할 수 있다. 따라서 본 논문에서 제안한 기법이 빠르게 결함을 복구하면서도 보다 다양한 상황에서 복구가 가능함을 확인하였다.

4.5 WWDG 파라미터 설정에 따른 성능 평가

본 논문에서 제안한 긴 꼬리 와치독 타이머는 대부분의 경우에 WWDG에 기반 하여 동작을 하게 되고 WWDG가 정상적으로 결함을 감지하지 못하는 경우에만 IWDG에 기반 하여 동작하게 된다. IWDG의 경우 단순히 지정된 주기가 지나면 시스템을 리셋하는 방식이어서 결함의 검출 성능이 큰 의미가 없다. 반면 WWDG는 IWDG와 다르게 카운터를 재설정할 수 있는 범위가 따로 있고 인터럽트 핸들러에 의해서 2단계에 걸쳐서 결함의 감지와 복구가 일어나기 때문에 설정 파라미터인 윈도우 크기와 카운터 값에 따라 본 논문에서 제안된 기법이 얼마나 효과적으로 적용될 수 있는지에 대한 실험이 필요하다.

이러한 실험을 위해 uC/OS-II 실시간 운영체제 [18,19] 환경에서 임의의 시간 제약을 갖는 태스크에 대해서 가상의 결함을 갖도록 하고 UART를 통한 통신 활동이 발생하도록 하여 일반적인 소형 임베디드 시스템과 유사한 환경을 수립하였다. 실험에서는 WWDG에 의해 시스템 리셋이 발생할 때마다 UART를 이용해 전달된 문자를 PC의 터미널에 출력하도록 하고 1분 동안 WWDG가 수행된 횟수를 출력하게 하였다.

4.5.1 카운터 값에 대한 실험

WWDG 카운터 값에 대한 실험에서는 윈도우 범위 이외에서 리셋이 발생하지 않도록 윈도우 값을 127로 고정하였다. 실험에서 사용된 카운터 값은 127

보다 항상 작아야 하고 WWDG_Interrupt 인터럽트 핸들러를 발생시킬 수 있는 64(0 × 3f) 보다는 크다. WWDG 카운터 값에 대한 실험 결과는 Fig. 12와 같다.

Fig 11의 실험 결과에서 WWDG의 카운터 값이 증가함에 따라 WWDG_Interrupt 인터럽트 핸들러가 수행되는 횟수가 줄어들게 된다. 이를 좀 더 명확히 표현하기 위해 WWDG 실행 횟수를 카운터 값에 대한 환산 값(Normalized value)으로 변환한 결과 그 비율이 더 급격하게 차이 나는 것을 볼 수 있다.

4.5.2 윈도우 값에 대한 실험

WWDG의 카운터가 재설정 될 수 있는 범위를 결정하는 윈도우 값에 대한 실험에서는 마찬가지로 윈도우 범위 이외의 값에서 카운터가 재설정되는 경우를 배제하기 위해 카운터 값을 127로 고정하고 실험하였다. 실험에서는 윈도우 값을 65에서 125까지 변화시키면서 4.1절과 동일한 실험을 수행하였으며 실험 결과는 Fig. 13과 같다. 실험 결과 윈도우의 범위를 결정하는 윈도우 값의 변화에 대해 WWDG의 실행 횟수가 일정하게 변화하는 경향을 확인할 수 없었다. 즉, 윈도우 값보다 카운터 값이 WWDG의 실행 횟수에 더 많은 영향을 미치는 것을 알 수 있다.

5. 결 론

본 논문에서는 시스템이 일시적인 결함으로 인해 비정상적인 상태에 처한 경우에 시스템 리셋을 통해 다시 정상적인 상태로 복구 시킬 수 있는 와치독 타이머에 대해서 와치독 타이머 자체에 결함이 발생할 수 있다는 점에서 착안하여 STM32F4 임베디드 프로세서에서 제공하는 두 가지 서로 특성이 다른 IWDG와 WWDG를 결합하여 사용하는 긴 꼬리 와치독 타이머를 제안하였으며 결함이 발생할 수 있는 여러 테스트 시나리오를 통해 제안된 기법이 향상된 가동시간을 제공하여 줄 수 있음을 확인하였다.

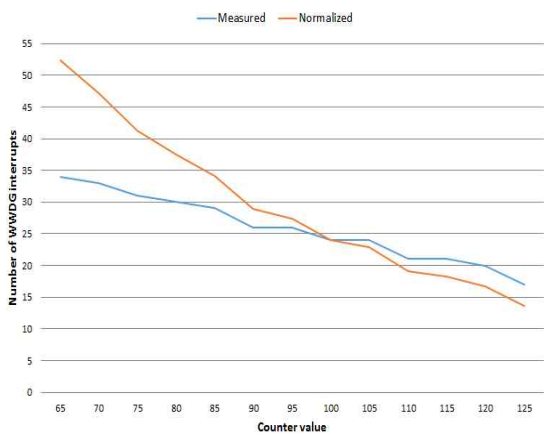


Fig. 12. Experimental results for WWDG counter value.

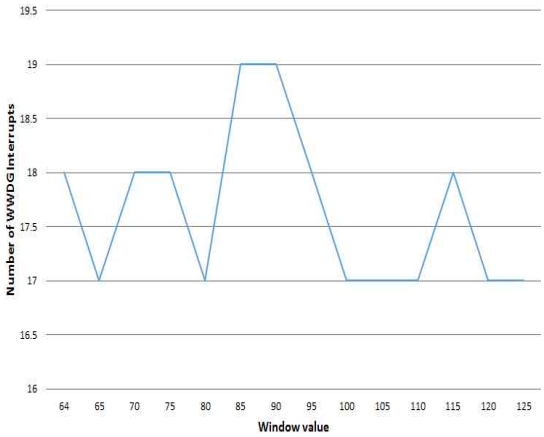


Fig. 13. Experimental results for WWDG window value.

또한, 제안된 기법이 일반적인 경우에 주로 사용하는 WWDG가 시간 제약성을 만족해야하는 실시간 태스크에 대해 WWDG의 설정 파라미터인 카운터 값과 카운터가 재설정 될 수 있는 윈도우 범위를 결정하는 윈도우 값에 대한 WWDG 실행 횟수에 대한 추가적인 실험을 실시함으로써 시간적 결함이 있는 태스크에서 카운터 값을 조정함으로 인해서 가동시간을 향상 시킬 수 있음을 확인하였다.

본 논문에서는 제안된 기법의 검증과 성능 실험을 위해 가상의 결함을 주입하였으나 향후 연구에서는 실제 소형 임베디드 시스템을 위한 프로그램에 탑재하여 분석함으로써 실제 사용에 있어서 발생할 수 있는 문제점들을 파악하고 다양한 경우의 결함 상태에 대한 연구를 진행할 계획이다.

REFERENCE

[1] Embedded System, http://en.wikipedia. org/wiki/Embedded_system (accessed Feb, 11, 2015).

[2] J. Woo and G. Sha, *Embedded Linux: Basics and Applications*, Hanbit Media, Seoul, Korea,

2007.

[3] T. Jeon and C. Kim, "A Real-Time Embedded Task Scheduler considering Fault-Tolerant," *Journal of Korea Multimedia Society*, Vol. 14, No. 7, pp. 940-948, 2011.

[4] D. Shin, J. Jung, and S. Kang, "Trends and Forecasting in IoT," *Korean Society for Internet Information*, Vol. 14, No. 2, pp. 32-46, 2013.

[5] J. Lee, S.H Kim, S.B Lee, H.J Choi, and J.J Jung, "A Study on the Necessity and Construction Plan of the Internet of Things Platform for Smart Agriculture," *Journal of Korea Multimedia Society*, Vol. 17, No. 11, pp. 1313-1324, 2014.

[6] D. Lewis, *Fundamentals of Embedded Software: Where C and Assembly Meet*, Prentice Hall, NJ, USA, 2001.

[7] Fault tolerance, http://en.wikipedia.org/wiki/Fault_tolerance (accessed Feb, 12, 2015).

[8] K. Jung, S. Tak, and C. Kim, "Mechanism for Managing Fault-Tolerant Embedded Real-Time Tasks," *Journal of Korea Multimedia Society*, Vol. 10, No. 7, pp. 882-892, 2007.

[9] Watchdog Timer, http://en.wikipedia.org/wiki/Watchdog_Timer (accessed Feb, 17, 2015).

[10] RM0090 Reference Manual, http://www.st.com/stonline/stappl/resourceSelector/app?page=fullesourceSelector&doctype=reference_manual &LineID=11 (accessed Feb, 17, 2015).

[11] Cortex-M4 Processor, http://www.arm.com/products/processors/cortex-m/cortex-m4-processor.php (accessed Feb, 12, 2015).

[12] R.M. Pathan, "Recovery of Fault-tolerant Real-time Scheduling Algorithm for Tolerating Multiple Transient Faults," *Proceedings of 10th International Conference on IEEE, Computer and Information Technology,* pp. 1-6, 2007.

[13] A.M. El-Attar and G. Fahmy, "An Improved Watchdog Timer to Enhance Imaging System Reliability in the Presence of Soft Errors," *Proceedings of the 2007 IEEE International Symposium, Signal Processing and Information Technology,* pp. 1100-1104, 2007.

[14] M. Duricek, M. Pohronska, and T. Krajcovic, "Functional Prototype of Multiple Watchdog System Implemented in FPGA," *Proceedings of International Conference on Applied Electronics,* pp.75-78, 2012.

[15] I. Exman and S. Reznitsky, "To Be and Not to Be at the Same Time: Hidden Watchdog Timers," *Proceedings of IEEE 26th Convention of Electrical and Electronics Engineers in Israel,* pp. 897-900, 2010.

[16] S. Park and Y. Oh, *Complete System Programming in ARM Cortex-M3 II*, D&W Wave, 2011.

[17] MCBSTM32F400 Evaluation Board, http://www.keil.com/mcbstm32f400/default.asp (accessed Feb, 17, 2015).

[18] uC/OS the real-time kernel, http://www.micrium.com/ (accessed Feb, 17, 2015).

[19] J.J. Labrosse, *MicroC OS II: The Real Time Kernel*, CMP Books, USA, 2002.

최 하 연

2013년 이화여자대학교 컴퓨터공학과 (학사)
2013년~이화여자대학교 컴퓨터공학과 석박통합과정 재학 중
관심분야 : 임베디드 소프트웨어, 실시간 시스템, PAN 등

윤 지 완

2016년 이화여자대학교 컴퓨터공학과 졸업 예정(학사)

박 서 연

2016년 이화여자대학교 컴퓨터공학과 졸업 예정(학사)

김 예 솔

2016년 이화여자대학교 컴퓨터공학과 졸업 예정(학사)

박 상 수

1998년 한국과학기술원 전산학과 졸업(학사)
2000년 서울대학교 컴퓨터공학과 졸업(공학 석사)
2006년 서울대학교 컴퓨터공학과 졸업(공학 박사)

2006~2008년 Univ. of Michigan Research Fellow
2008~2008년 삼성전자 종합기술원 전문연구원
2009~현재 이화여자대학교 컴퓨터공학과 조교수
관심분야 : 임베디드 소프트웨어, 실시간 시스템, 시스템 시뮬레이션 등