실시간 운영체제 UbiFOS™에서 Watchdog Timer를 이용한 멀티태스킹 시스템 오류제어

정근재°, 송예진, 김용희, 이철훈
충남대학교 컴퓨터공학과,
e-mail : {gjjeong°, songyj, yonghee, chlee}@cnu.ac.kr

Multi-Tasking System Error Control Using Watchdog Timer based on UbiFOS™ Real-Time OS

Gun-Jae Jeong°, Ye-Jin Song, Yong-Hee Kim*, Cheol-Hoon Lee
Dept. of Computer Engineering, Chung-Nam National University

요 약

내장형 시스템은 우리들의 생활에 커다란 변화를 가져왔으며, 많은 적용분야와 다양한 기능을 갖추고 있어서 일상 생활에 널리 사용되고 있다. 문제는 이러한 기기들이 마이크로 컨트롤러에 가해지는 전기적 잡음과 전자기 방해가 많은 환경에서 사용되어지고 있다는 점이다. 따라서 이러한 환경에서는 시스템의 안정적인 운영을 도울 수 있는 기술중의 하나인 Watchdog Timer(WDT)가 필요하다. 본 논문에서는 WDT 를 이용한 시스템 오류제어를 실시간 운영체제인 UbiFOS™에 적용하였다.

1. 서 론

임베디드 시스템은 전용 동작을 수행하거나 특정 임베디드 소프트웨어 응용프로그램과 함께 사용되도록 디자인된 특정 컴퓨터 시스템 또는 컴퓨팅 장치를 의미한다 [1]. 이러한 임베디드 시스템에 사용하는 운영체제가 실시간 운영체제(RTOS : Real-Time Operating System)이며, 이러한 실시간 운영체제는 앞으로 다양한 환경에서 제한된 목적을 안정적으로 수행하기 위한 기술이 필수적이다. WDT 는 카운터의 일종으로 정기적인 카운터 값의 입력이 정상 상태이며, 요구되는 카운터 값의 재 입력이 없는 비정상 상태에서 카운터가 증가 하면 시스템을 재 부팅 시키는 역할을 한다. 즉 이러한 기능은 시스템의 안정성을 위해 필요한 기능이다. 만약 응용 프로그램이 동일한 루틴을 계속 반복하는 무한 루프를 돌거나 폭주 또는 프로그램 어드레스 영역 밖의 공간을 침범하게 되면, 메모리에서 명령어를 가져와 실행할 수 없으므로 시스템이 정지(Halt) 상태가 된다 [2]. 이 경우 사용자는 리셋 버튼을 눌러 빠져 나온다. 이렇게 CPU 가 정지(Halt)상태가 되었을 때 리셋 버튼을 눌러 초기화하지 않아도 자동적으로 시스템을 리셋시키는 기능이 바로 WDT 이다 [4].

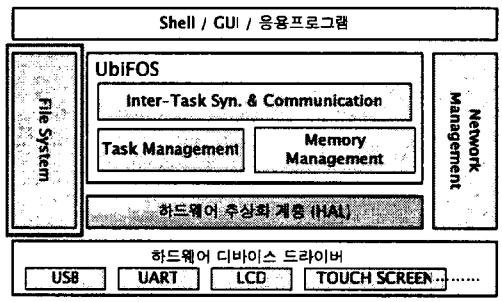
본 논문에서는 WDT 를 실시간 운영체제인 UbiFOS™에 적용하였다.

본 논문의 구성은 2 장에서 실시간 운영체제의 전체적인 구성을, 3 장에서는 WDT 를 이용한 시스템 오류제어의 설계 및 구현 내용을 다룬다. 4 장에서는 테스트 환경 및 결과를 기술하고, 마지막으로 5 장에서는 결론 및 향후 연구 과제에 대해서 기술한다.

2. 관련 연구

2.1 UbiFOS™

실시간 운영체제인 UbiFOS™은 우선순위 기반의 선점형 스케줄러를 제공하여 태스크의 우선순위를 0 부터 255 까지 256 단계로 구분하고 있다. 그리고 실행준비된 태스크들 중에서 정해진 시간 내에 가장 높은 우선순위의 태스크를 찾기 위해 별도의 테이블과 리스트를 관리하며, 같은 우선순위의 태스크 사이에서는 선입선출(FIFO) 방법과 Round-Robin 방법을 지원한다.

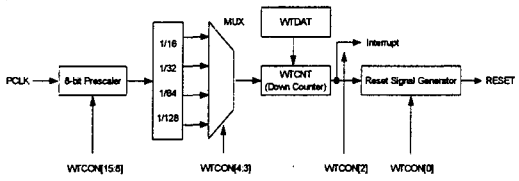


[그림 2-1] UbiFOS™ 전체 구성도

* 본 연구는 정보통신부의 선도기반기술개발사업의 지원으로 수행되었습니다.

2.2 WDT (Watchdog Timer)

타겟 보드인 SMDK2400 의 WDT 는, 콘트롤러의 동작이 노이즈나 시스템 에러 등의 오동작으로 방해를 받았을 때, 콘트롤러를 재가동하는 데 사용된다. WDT 의 리셋 신호는 128 PCLK 싸이클 동안 발생한다 [3].



Watchdog Timer Block Diagram

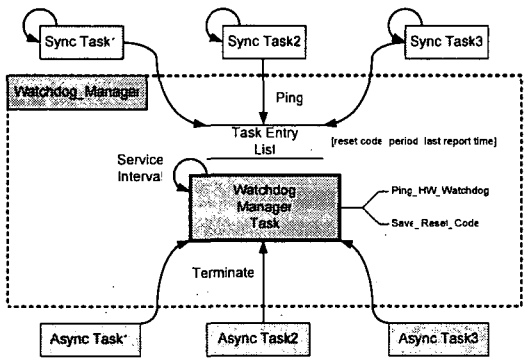
[그림 2-2] WDT Block Diagram

2.3 WDT(Watchdog Timer) 레지스터

- Watchdog Timer Control (WTCON) Register
  WDT 를 동작 가능/불가능으로 설정하고, WDT 카운터를 클리어 할 수 있다.
- Watchdog Timer Data (WTDAT) Register
  WDT 의 타임 아웃 주기 값을 가지고 있다.
- Watchdog Timer Counter (WTCNT) Register
  WDT 의 현재 카운트 값을 가지고 있다 [3].

2.4 WDT for Multitasking System

[그림 2-3]은 Watchdog_Manager 가 WDT 를 관리하고 태스크들이 스케줄링되어 실행될 때 WDT 관련 레지스터들을 모니터링 하고, 에러 상황 발생시 리셋 시그널을 발생시키는 모습을 보여준다. 멀티 태스킹 환경에서 스케줄러를 통해 실행 되어지는 태스크는 Watchdog_Manager 를 통해 WDT 레지스터의 싱크를 맞추어 WDT 의 레지스터 값을 유지하고 interval time 을 서비스 하게 된다 [4].



[그림 2-3] Watchdog component task relationships

3. Real-Time OS 인 UbiFOS™상에서의 WDT 구현

WDT 도 타이머의 일종이기 때문에 클럭이 발생할 때마다 인터럽트가 발생하게 된다. [그림 3-1]은 UbiFOS™에서 WDT 인터럽트를 처리하기 위한 과정을 나타낸다. 이 과정에서 인터럽트 레지스터에 WDT 의 인터럽트 비트를 마스크 시킨 후 IRQ 모드로 전환시키고, 분주요소와 프리스케일러 값을 변경시키면서 WDT 를 동작 시키게 된다.

```
MK_Void_t Test_WDT()
{
   unsigned int      PreScale;
   unsigned int      Data1;
   unsigned int      Frequency;

   MK_Printf("-----------------------₩n");
   MK_Printf("      [iRTOS WDT 테스트]   ₩n");
   MK_Printf("-----------------------₩n");

   rINTMOD = 0x00;
   rINTMSK &= ~BIT_WDT;
   rINTMOD = 0x00;                    //IRQ mode
   isWdtInt = 0;

   /* WDT INTMSK Enable */
   ClearPending(BIT_WDT);
   rSRCPND=(BIT_WDT);
   rINTMSK&=~(BIT_WDT);
   .
   .
   .
}
```

[그림 3-1] 인터럽트 레지스터 셋팅

[그림 3-2]의 Watchdog_Manager 는 스케줄 가능한 태스크들을 리스트로 관리하고, 태스크와 동기화를 하기 위해 WDT 관련 레지스터 값을 보존하고 있다.

```
typedef struct Watchdog_Manager {
   MK_U32_t          wm_Magic;
   // Watchdog_Manager 를 식별하는 Magic Number
   MK_S32_t          wm_WDTData;
   // WDT 의 interval time 를 저장하는 변수
   MK_U32_t          wm_WDTConter;
   // WDT 의 Counter 값을 저장하는 변수
   MK_U32_t          wm_WDTControl;
   // WDT 의 상태 레지스터 값을 저장하는 변수
   struct mk_task_struct *t_pTaskNext;
   // 태스크를 관리하기 위한 리스트의 이전 태스크 구조체
      를 가리키는 포인터
   // 태스크를 관리하기 위한 리스트의 다음 태스크 구조체
      를 가리키는 포인터
   } WDT_Manager;
```

[그림 3-2] Watchdog_Manager Structure

[그림 3-3]은 프리스케일러 값을 64(dec), 32(dec)로 셋팅 후 WDT 를 동작 시키는 과정을 나타낸다.

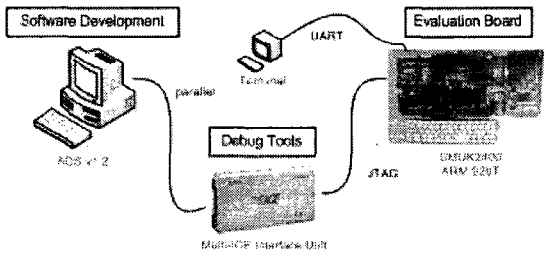
```
   //  Prescaler: 64
   isWdtInt = 0;
   rWTCON     = (64<<8) | (3<<3) | (1<<2);
   // Prescaler : 32, Interrupt enable
   rWTDAT     = 3077 ;
```

```
rWTCNT    = 3077 ;
rWTCON    = rWTCON | (1<<5);
while(isWdtInt != 10);

//  Prescaler: 32
isWdtInt = 0;
rWTCON    = (32<<8) | (3<<3) | (1<<2);
// Prescaler : 32, Interrupt enable
rWTDAT    = 3077 ;
rWTCNT    = 3077 ;
rWTCON    = rWTCON | (1<<5);

while(isWdtInt != 10);
```

[그림 3-3] 프리스케일러 값 변경

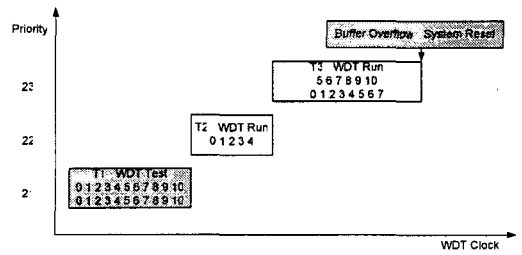
4. 테스트 환경 및 결과

본 논문에서 구현한 타이머는 실시간 운영체제인 UbiFOS™ 의 커널에 구현하였으며, WDT 를 ARM920T 32-bit RICS CPU 가 탑재되어 있는 SMDK2400 보드에서 ADS 1.2 통합 개발환경을 사용하여 테스트 하였다.

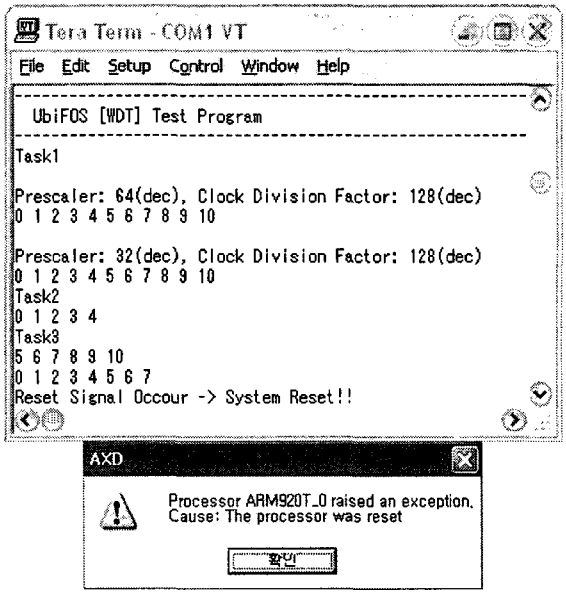


[그림 4-1] 테스트 환경

[그림 4-2]는 프로그램이 동작하는 과정을 그림으로 나타낸 것이다. 먼저 우선순위가 21, 22, 23 인 태스크를 생성하고 Watchdog_Manager 를 생성하여 WDT 를 관리한다. 다음으로 우선순위가 가장높은 Task1 이 실행되어 프리스케일러 값을 64, 32 로 변경하여 WDT 를 테스트 하고, 다음 우선순위의 Task2 는 4 Clock 동안 실행되고 나서 Task3 에게 CPU 점유권을 넘겨주게 된다. 이때 Task3 는 Task2 에서 동작되었던 WDT 레지스터 값을 이어 받아 그 이후부터 실행하게 된다. Task3 가 실행되는 도중 임의로 Buffer Overflow 상황을 발생시켰고, 이후 시스템이 리셋되는 것을 확인 할 수 있다.



[그림 4-2] 테스트 시나리오



[그림 4-3] 실험 결과

5. 결론 및 향후 연구 과제

본 논문에서 실시간 운영체제에서의 WDT 를 구현하였고, 이를 통해 시스템 오류 상황이 발생하였을 때 사용자의 리셋 입력을 받지 않아도 시스템 스스로 리셋을 발생시키는 것을 확인할 수 있었다.

차후에는 다양한 시스템 폴트 상황에 대한 테스트를 거쳐 시스템 안정화 연구가 진행되어야 하겠다. 그리고 실시간 운영체제에서의 효율적 오류 제어 및 흐름 제어에 대한 연구가 진행되어야 하겠다.

참고문헌

[1] Embedded System & RTOS, http://www.inestech.com.

[2] David Stepner, et. al., , " Embedded Application Design Using a Real-Time OS ", IEEE, 1999

[3] Samsung Electronics " S3C2400 RISC MICRO PROCESSOR User's Manual "

[4] "General Purpose Watchdog Timer Component for a Multitasking System", APRIL 1997 http://www.embedded.com.