논문 2008-45CI-6-12

# 소형 네트워크 임베디드 시스템에 TinyOS 이식 과정에서의 이슈 및 디버깅 기법

## ( Issues and Debugging Methodology for Porting TinyOS on a Small Network Embedded System )

김 대 남*, 김 교 선**

( Daenam Kim and Kyosun Kim )

요 약

ZigBee 통신기반의 네트워크 임베디드 시스템을 위한 많은 플랫폼들이 개발되어 왔으며 TinyOS와 같은 소형 운영체제가 탑재되어 다양한 주변장치를 통해 네트워킹, 정보수집, 명령 수행 등 다양한 기능들을 효율적으로 구현할 수 있도록 하고 있다. 새로운 플랫폼에 운영체제를 이식하는 과정에서는 계수기와 같이 중요한 특정 하드웨어 장치가 운영체계에서 요구하는 기능이 부족하다면 소프트웨어 및 다른 하드웨어 장치로 해당 기능을 구현해야 한다. 본 논문은 먼저 계수기에 비교기 인터럽트 기능이 없는 플랫폼에서 운영체계의 요구 기능을 만족하는 다중 시스템 타이머를 구현하는 기법을 제안한다. 또한, 이식과정에서 예측하기 어려운 오류가 주입될 수 있기 때문에 이에 따라 발생하는 수많은 오동작에 대처해야 할 것이다. 불행히도 TinyOS에는 하드웨어의 인터럽트에 의해 구동되는 수많은 비동기 처리가 필요한 반면 새로운 플랫폼에는 탑재된 하드웨어 각각에 대한 모델이 확립되지 않아 시뮬레이터가 미리 제공되지 못한다. 본 논문은 이러한 열악한 상황에서 사용할 수 있는 새로운 디버깅 기법을 제안한다. 이 방법은 레이디오펄스(주)의 MG2400과 MG2455에 TinyOS 2.0을 이식하는 과정에서 발생한 이슈들과 원인을 찾아내는데 사용되어 그 실용성을 입증하였다.

Abstract

Numerous platforms have been developed for ZigBee-based network embedded systems. Also, operating systems like TinyOS have been installed to facilitate efficient implementation of wireless sensor network applications which collect data, and/or execute commands. First of all, porting an operating system on a new platform may need invention of a substitute for a required but unsupported hardware component. This paper presents a multiplexed virtual system timer for a platform without a counter comparator which we have contrived to emulate by using an extra counter. Such porting also injects unexpected faults which cause a variety of painful failures. Unfortunately, TinyOS requires to handle a lot of asynchronous hardware interrupts which are hard to trace during debugging. Besides, simulators are not available for a new platform since the models of hardware on the platform are not usually developed, yet. We propose novel instrumentation techniques which can be used to effectively trace the bugs in such lack of debugging environment. These techniques are used to identify and fix a great deal of nasty issues in porting TinyOS 2.0 on MG2400 and MG2455 platforms made by RadioPulse Inc.

**Keywords**: Network Embedded System, TinyOS Porting, TinyOS Issue, Debugging, System Timer

## I. 서 론

유비쿼터스 컴퓨팅 시대 요구를 만족시키기 위해 네트워크 기능을 포함하는 임베디드 시스템 (Network Embedded System, NES)이 급속도로 파급되고 있다. 특히, 고성능 다기능화에 따라 PC에까지 탑재되었던 운영체제가 임베디드 시스템에도 탑재되고 있으며 무선 센서 네트워크 시스템에서도 TinyOS[1]와 같은 소형 운영체제가 탑재되고 있다. 운영체제는 응용 제품의 성능 및 기능을 향상시킬 뿐만 아니라 개발 생산성 및 제품 신뢰성을 증대시키지만 이를 새로운 플랫폼에 이식[2]하

* 학생회원, ** 정회원, 인천대학교 전자공학과
(Dept. of Electronics Engineering, University of Incheon)

※ 본 논문은 인천대학교 2008년도 자체연구비 지원에 의하여 연구된 것임.
접수일자: 2008년10월10일, 수정완료일: 2008년10월30일

는 단계에서는 그 자체의 잠정적 불완전성으로 인해 시스템이 정지하는 상황도 발생할 수 있으며 이에 대처해야한다. 예를 들어 하드웨어 인터럽트에 의해 비동기적으로 함수가 수행될 때 공유 영역에서 발생하는 경쟁상황과 같이 예상하기 어려운 문제가 발생하면 이를 추적하여 변수의 값을 직접 읽어 본다던지, 메모리 침범으로 인한 포인터의 오류는 없는지를 확인해야만 하기 때문이다.

본 논문에서는 TinyOS 2.0 이식에 필요한 타이머, UART, RF 등의 주변장치 드라이버와 MAC 프로토콜 및 네트워크 프로토콜 (CTP[3~5], Dissemination[6~8])을 이식하면서 발생했던 이슈들을 소개하고 제한된 환경에서 편리하고 효율적으로 사용할 수 있는 디버깅 방법들을 제시한다. TinyOS 2.0을 이식하기 위한 플랫폼으로는 8051기반의 ZigBee 단일칩 솔루션인 레이디오펄스(주)의 MG2400[9]과 새롭게 출시한 MG2455[10]를 사용하였다.

## II. 본 론

### 1. TinyOS의 동시성 모델

TinyOS의 동시성 모델은 그림 1과 같다. 타이머, UART, MAC, 외부 포트 등의 하드웨어에서 인터럽트가 발생하면 해당 이벤트 처리기 (Event Handler)가 수행되고 여기서 이벤트는 다시 운영 체계 계층 구조의 HIL (Hardware Implementation Layer) 및 HAL (Hardware Abstraction Layer)을 거쳐 응용 프로그램에 차례로 전달되어 처리된다. 이 때, 발생한 이벤트를 처리하는 함수들은 다른 장치의 명령을 호출하게 되며 이는 거꾸로 HAL 및 HIL을 거쳐 하드웨어에 전달된다. 데이터 처리 등 시간이 소요되는 작업은 작은 작업

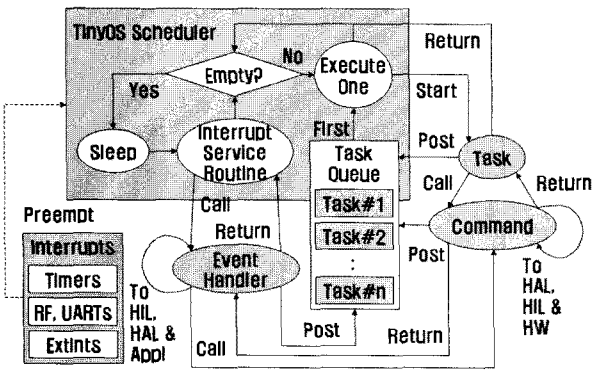



그림 1. TinyOS의 동시성 모델
Fig. 1. Concurrency model of TinyOS.

(Task)으로 나누어 작업 큐 (Queue)에 넣게 되며 스케줄러는 큐에 들어온 순서대로 작업을 하나씩 수행시킨다. 작업 수행 중에 다른 작업을 큐에 올릴 수도 있다.

이벤트의 처리는 비동기로써 다른 명령 (Command)이나 작업 또는 이벤트 처리를 점유 (Preempt)할 수 있으나 각 작업은 순서대로 수행되며 서로 점유할 수 없다. 이벤트 처리 시 호출되는 비동기 명령은 다른 함수를 점유하게 되나 동기 명령은 다른 함수를 점유하지 않는다. 동기 명령은 큐에서 하나씩 꺼내 실행되는 작업 또는 다른 동기 명령에서만 호출할 수 있기 때문이다. 여러 함수에서 값이 변경될 수 있는 변수들에 대한 경쟁 상황을 검사할 때 이와 같이 다중 계층 함수를 사용하면 일정 기간 인터럽트 불능을 유발하여 응답 속도를 저하시키는 원자 (Atomic)구간 사용을 줄일 수 있다.

명령은 분할 처리 방식 (Split-Phase)으로 수행된다. 먼저, 명령 함수의 호출은 봉쇄 (Block)되지 않으며 하드웨어에 수행 시작을 지시하거나 대기자 목록에 등록한 후 곧 되돌아간다. 실행 완료 후에야 이벤트를 발생시켜 해당 처리 함수에 명령의 결과를 전달한다. 만약 이 수행 완료 이벤트의 처리 함수에서 명령을 재차 호출한다면 상호 배제 검사가 필요한 비동기 함수 수행 기간 및 스택 메모리 영역 사용이 늘어날 수 있기 때문에 해당 작업을 큐에 올리는 것이 바람직하다.

### 2. 8051 기반 단일 칩 플랫폼에 TinyOS 2.0 이식

레이디오펄스(주)의 망고 플랫폼에 장착된 MG2400과 MG2455 칩들은 8051 컨트롤러, RF 송수신기, Modem, MAC, AES, RTC, WDT, ADC, 온도 센서, GPIO 등을 단일 칩에 집적하고 있다. 이와 같은 망고 플랫폼에 TinyOS 2.0을 이식하였으며 그 구조는 그림 2와 같다. 모듈 및 인터페이스를 포함한 HAL 상위 구조의 변경은 극소화하였으며 HIL 하위 구조는 RF 및 ADC가 단일 칩에 집적된 장점을 살려 최적화하였다.

#### 가. gcc와 KEIL 컴파일러의 차이

TinyOS는 NesC를 사용하며 NesC 컴파일러는 gcc로 컴파일할 수 있는 C 코드를 발생시킨다. 그러나 gcc는 8051 컨트롤러를 위한 교차 컴파일을 할 수 없다. 이러한 문제를 해결하기 위해서 8051 워킹 그룹[11]에서는 펄 스크립트 (Perl Script)를 사용하여 NesC 컴파일러가 gcc 용으로 발생시킨 코드를 Keil 컴파일러용으로 변환하도록 하고 있다. 이를 변환 스크립트 (Mangle Script)라 하며 당초 제공된 스크립트는 저장 계층 지정

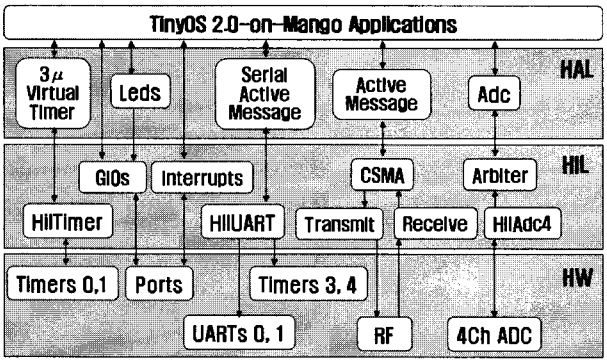



그림 2. Mango 플랫폼에 탑재된 TinyOS 2.0 계층 구조
Fig. 2. Structure of TinyOS 2.0 on the Mango platform.

```
#define XBYTE ((unsigned char volatile xdata*) 0)

#define MG2400R_SWRST (0x1A00 + 0xC7)
XBYTE[MG2400R_SWRST] &= 0xD7;
```

그림 3. KEIL컴파일러에서의 XBYTE표현
Fig. 3. Expression of XBYTE in Keil compiler.

```
typedef int XBYTE;
#define MG2400R_SWRST (0x1A00 + 0xC7)

*(XBYTE *) MG2400R_SWRST &= 0xD7;
```

그림 4. 수정된 XBYTE 표현
Fig. 4. Modified expression of the XBYTE construct.

등 일부 처리하지 못하는 부분이 남아 있었다. 8051은 변수의 저장 계층 (data/near, xdata/far, idata, pdata)을 선언하여 할당될 메모리 주소 영역을 결정하도록 되어 있으며 저장 계층이 지정되지 않으면 메모리 모델 (small, large)에 따라 data/near나 xdata/ far가 지정된다. 그러나 저장 계층 선언은 NesC가 이해 못하는 구문이다.

MG2400과 MG2455는 ADC 및 RF 등 집적된 장치의 각종 레지스터 및 버퍼 등을 확장 주소 영역에 두고 있는데 이에 접근하기 위해서 xdata 저장계층을 사용해야한다. 본 논문에서는 다음과 같은 저장계층 지정방법을 제안한다.

이 때, NesC 컴파일러는 xdata 라는 용어를 이해하지 못하기 때문에 오류로 처리한다. 이것을 우회하기 위해 NesC 환경에서는 다음과 같이 기술하며 변환 스크립트에서 "XBYTE"가 포함된 구문을 찾아낸 뒤 Keil 컴파일러가 이해할 수 있는 형태로 원상 복귀한다.

이밖에 복잡한 통신 계층 구조에서 오는 다단계 함수 호출 오버헤드를 줄이기 위해서는 gcc의 inline 함수를 사용하고 있는데 현재는 Keil 컴파일러가 이를 처리할 수 없기 때문에 inline 함수를 일반 함수로 변환하여 사용하고 있다.

나. 타이머

타이머는 프로세서와 독립적으로 동작하며 주기적 또는 비주기적으로 프로세서에 인터럽트 신호를 발생시켜 다중 프로세스, 네트워크의 패킷 시간 제어, 이벤트 발생 등을 처리할 수 있도록 하며 현재 시각 확인도 가능하도록 한다[12].

(1) 타이머의 구조

타이머 하드웨어는 그림 5와 같이 부호 없는 n 비트의 초기값 정수를 적재할 수 있는 계수기를 기본으로 하며 값이 증가하다가 최대수에서 0으로 바뀔 때 오버플로우 인터럽트가 발생한다. 이 때 초기값을 자동으로 재적재할 수 있는 레지스터를 추가하면 인터럽트 발생 시 처리 지연 없이 즉시 재적재할 수도 있다. 계수기 입력은 클록을 그대로 사용하거나 분주하여 사용하는데 이 때 프리스케일링 레지스터가 추가되며 이 레지스터 값의 지수 함수적 배율로 주파수가 분주된다. 계수기 값은 언제든지 읽어서 현재 시각을 확인할 수 있으며 비교기를 추가하면 특정값에 도달할 때 인터럽트가 발생하도록 할 수 있다. 이 때 계수기 값과 비교될 값은 비교 레지스터에 미리 저장된다.

운영체제 및 응용 프로그램에서는 일반적으로 많은 수의 32비트 주기 또는 비주기 타이머가 필요하지만 소형 컨트롤러에는 대개 2~3개의 8비트 혹은 16비트 타이머만이 제공된다. 더구나, 프리스케일러, 재적재 레지스터, 비교기 등이 포함되지 않을 수도 있다. 이러한 격차를 해소하기 위해서는 주어진 하드웨어 타이머 기능을 이용하되 운영체제 내에 소프트웨어로 이를 보완하여 가상 타이머를 구현한 후 시스템 호출 형태로 제공해야 한다.

먼저 계수기에 나타내는 시각의 단위를 결정해야 하는데 프리스케일러가 존재하지 않거나 분주 배율 레지스터 조정으로 원하는 단위를 만들 수 없을 경우 흔히

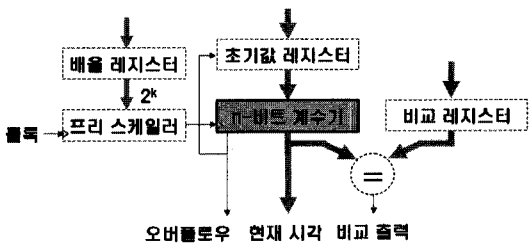



그림 5. 타이머 하드웨어
Fig. 5. Timer hardware.

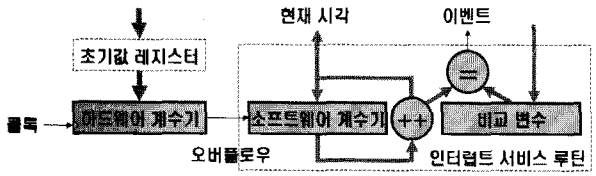



그림 6. 단위 시간 발생
Fig. 6. Jiffy generation.

그림 6과 같이 타이머 하드웨어에서 단위 시간마다 인터럽트를 발생시키도록 초기값을 설정해 놓고 인터럽트 서비스 루틴 (ISR)에서 소프트웨어 계수기를 증가시키는 방법을 사용한다. 증가된 계수가 예정 시각과 같으면 해당 이벤트를 발생시켜 처리되도록 한다. 이 방법은 단위 시간이 수백 밀리 초 이상일 경우 문제가 없으나 그 보다 간격이 좁을 때는 인터럽트 서비스 루틴의 잦은 수행으로 심각한 성능저하 현상이 나타난다. 또한, 초기값 재적재 레지스터 하드웨어가 제공되지 않을 경우 인터럽트 서비스 루틴에서 초기값을 적재하는 시간만큼 오차를 발생시키며 이 오차는 현재 시각에 누적된다. 결국 원하는 단위 시간을 만들지 못하면 단위 시간 자체를 변경할 수밖에 없다.

둘째는 8비트 혹은 16비트 하드웨어 계수기를 이용하여 32비트 계수기를 구현해야 하는데[13~14] 흔히 그림 7과 같이 오버플로우 인터럽트가 발생할 때마다 1씩 증가하는 소프트웨어 계수기 값을 상위 비트들에 매핑하고 하드웨어 계수기 값은 하위 비트들에 매핑하여 32비트 계수기를 구성한다. 이벤트가 발생할 예정 시각 역시 32비트 값으로 주어지는데 이 중 하위 비트들이 하드웨어 비교 레지스터에 전달되어 비교 출력 인터럽트를 기다린다. 그림 7에는 표시되지 않았지만 비교 출력 인터럽트는 현재 시각의 상위 계수와 예정 시각의 상위 계수가 같을 때만 허용된다. 그러나 이 방법은 하드웨어 비교기가 존재하여 인터럽트를 발생시킬 수 있어야만 구현이 가능하다.

셋째는 하나의 하드웨어의 계수기로 다수의 소프트웨어 가상 타이머를 구현해야 한다[13~14]. 타이머 시작 시간 (t0), 이벤트 발생 간격 (dt), 주기적 발생 여부, 상태 등으로 구성된 데이터 구조로 원하는 가상 타이머 수만큼 배열을 할당한 후 이들이 초기화되면 이벤트 발생 예정 시각 (t0 + dt)이 가장 작은 타이머의 값을 예정 시각으로 정한다. 이 값은 비교 레지스터까지 전달되며 계수가 진행되어 비교 출력 인터럽트가 발생하면 가상 타이머들 중 이벤트 발생 예정 시각 (t0 + dt)이 현재 시각 이거나 그 이전인 것들에 대해서 각각 이벤트를 발생시킨다. 만약 주기적 발생이면 t0는 t0 + dt 값으로 증가되고 비주기 발생이면 상태를 바꾸어 꺼 놓는다. 가상 타이머의 수가 많아지면 t0 + dt 값에 따라 자동 정렬되는 다양한 구조[13~15]를 사용할 수 있다.

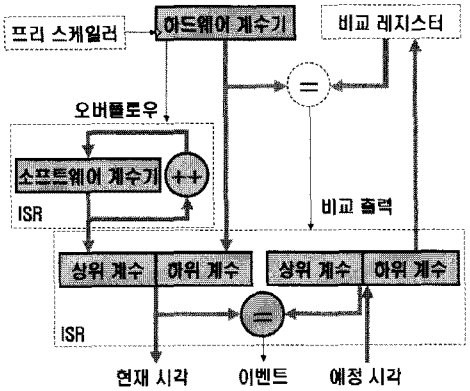



그림 7. 계수기 비트 수 확장
Fig. 7. Precision transformation of the counter.

(2) 8051 컨트롤러를 위한 타이머 설계

보통 8051 컨트롤러의 계수기에는 프리스케일러와 하드웨어 비교기가 탑재되어 있지 않다. 그러나 버전에 따라 16비트 계수기가 2개 이상 5개 있는 것도 있다. 일부를 UART와 연계하여 사용하면 2개 정도가 남는다. 비교기는 계수기의 값이 비교 레지스터의 값과 같을 때 인터럽트를 발생시킨다.

본 논문에서는 8051 컨트롤러와 같이 비교기가 없을 때 예정 시각에 인터럽트를 발생시키는 회로를 그림 8과 같이 제안한다. 타이머를 하나 더 사용하며 최대값 - (예정 시각 - 현재 시각)을 초기값으로 설정하면 예정 시각에 오버플로우 인터럽트가 발생한다. 이것이 비교기를 대체하게 된다. 현재 시각을 읽어 초기값을 계산해 내는 데 소모되는 지연 시간이 오차를 발생시킬 수 있으나 보정할 수 있고 가장 중요한 것은 현재 시각을 나타내는 계수기가 독립적으로 동작하여 다른 계수기의 초기값 계산중에도 멈추지 않기 때문에 이 오차가 누적되지 않는다는 것이다.

MG2400과 MG2455 내의 8051 컨트롤러는 입력 주파수 16MHz를 8MHz로 분주하여 동작하며 이를 12분주 (MG2455의 경우 6분주로 동작가능)하여 각 계수기가 동작된다. 따라서 1.5 usec마다 상향 계수된다. 또한, 16비트이기 때문에 98.304 msec마다 오버플로우가 발생된다. 응용 프로그램에서는 밀리 초 단위의 이벤트 발생이 필요하지만 ZigBee CSMA-CA 프로토콜의 타이밍을 위해서는 32 kHz (31.25 usec) 타이머가 필요하다.

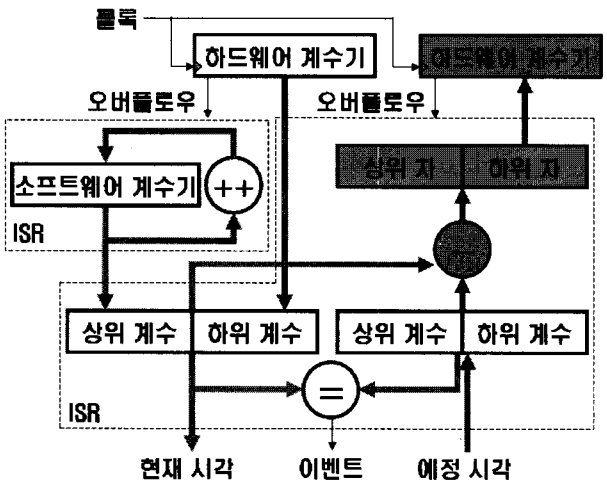



그림 8. 8051 컨트롤러를 위한 타이머 설계
Fig. 8. Timer design for the 8051 controller.

반면에 단위가 너무 작아지면 인터럽트 발생 횟수가 늘어나 시스템 부하를 가중시킨다. 부하가 가중되면 임계구간 (Critical Section) 수행 중에 인터럽트가 발생될 수 있으며 처리되지 못하고 기다리다가 다음 인터럽트가 발생되면 유실될 수 있다. 이러한 조건을 만족하는 기본 단위를 찾으려면 1.5 × 2n × k usec 가 31.25 usec에 가장 근사하며 n이 최대가 되는 n, k 값을 찾으면 된다. n = 1, k = 10 일 때가 최적이며 결국 1.5 x 21 = 3 usec 를 기본 단위로 해야 한다. 기본 단위가 하드웨어 계수기 주기의 2배이므로 하드웨어 계수기 16 비트 중 상위 15 비트만을 취하고 나머지 17비트는 소프트웨어 계수기 값을 사용한다. 32비트 가상 타이머는 3 × 232 usec = 12884.90189 sec (약 3시간 반)마다 오버플로우가 발생한다.

### 다. AD 변환기

MG2400에는 4채널 ADC가 탑재되어 있으나 MSP430에 비해 그 기능이 다소 부족하다. 특히 단일 채널 또는 다중 채널로부터 1회 또는 반복적으로 데이터를 읽어 들이는 등의 다양한 기능이 필요한데 하드웨어에서 이를 위한 모드 및 인터럽트를 제공하지 않는다. 따라서 소프트웨어로 이러한 기능들을 구현하였으며 이 때, 타이머를 이용한 캡처 기능이 이용되었다. 하나의 ADC 자원을 4개의 채널에 다중 연결하여 사용하기 때문에 라운드 로빈 (Round Robin) 방식의 자원 관리 기능이 포함된다.

### 라. UART

WSN 노드에서 호스트 컴퓨터나 장치에 유선으로 연결하여 데이터를 송수신하는 목적으로 UART가 사용된다. 액티브 메시지 (Active Message) 기반의 패킷 송수신 방식을 사용하며 호스트 컴퓨터에서 유선으로 연결된 WSN 노드에 데이터를 주고, 받을 수 있는 Java 프로그램도 제공된다. 전송속도를 115.2 kbps까지 올릴 수 있으며 수신 주기가 매우 빠를 경우(1 msec 미만)에는 8051 컨트롤러가 미처 수신된 바이트를 처리하지 못해 발생하는 오버런 (Overrun) 오류가 발생할 수 있기 때문에 WSN 노드로 송신을 할 때는 각 바이트 송신 중간에 지연 시간(약 1 msec)을 두는 것이 좋다.

### 마. RF

액티브 메시지를 기반으로 하고 IEEE 802.15.4 및 ZigBee 표준과 호환하는 CSMA-CA 패킷 전송방식을 구현한다. 기존에는 송신, 수신, 제어 등 여러 개의 모듈들이 하나밖에 없는 SPI 혹은 I2C의 사용권 획득을 경쟁하면서 (1) 명령 코드를 RF 트랜시버에 보내거나, (2) 송신 버퍼에 데이터를 보내고, (3) 수신 버퍼에서 데이터를 읽어 오는 등의 작업을 처리하였다. 따라서 FCFS (First Come First Served) 방식의 복잡한 자원 관리가 필요했었다. 그러나 단일 칩에 RF 트랜시버가 내장되어 있어 SPI 혹은 I2C를 사용하지 않고 레지스터나 버퍼를 데이터 메모리 영역에서 직접 접근할 수 있기 때문에 자원 관리가 필요 없다. 또한, CC2420과 같은 RF 칩과의 통신을 위해 필요했던 직병렬 데이터 변환 과정이나 스트로보 명령 발생, 상태 레지스터 값 수신 기능들은 모두 제거되었다. 복잡했던 인터럽트 및 타이머를 이용한 신호 캡처 기능 역시 대부분 제거되고 수신 완료 인터럽트 및 송신 완료 이벤트 캡처 기능만 사용된다.

### 바. 새로운 드라이버 이식

한백전자(주)의 Ubi-Mango[16]는 MG2400을 탑재한 ZigBee 모트 (Mote)로써 조도센서, 적외선센서, PCA9555, SHT11을 포함하고 있다. Ubi-Mango의 적은 GPIO의 불리한 점을 해결하기 위해서 모든 주변 장치를 I2C로만 제어하도록 설계되어 있으며 이 장치들을 TinyOS 2.0에서 사용할 수 있도록 장치 드라이버를 이식하는 작업을 하였다. 이식된 장치는 GPIO를 확장해 주는 PCA9555와 온·습도에 대한 아날로그 값을 디지털 값으로 변경해 주는 SHT11이 있다.

#### (1) PCA9555

PCA9555[17]는 I2C 통신 방식을 이용해서 GPIO의 수

를 최대 16개로 확장시켜준다. Ubi-Mango에 장착된 MG2400의 경우 확장하기 전의 가용 GPIO 수는 9개 이지만 PCA9555와 함께 이용하면 최대 23개로 늘어난다. 이는 많은 GPIO가 필요한 키패드나 LCD제어 시에 효율적이다. I2C 방식으로 장치의 주소, 쓰기 명령과 함께 자료를 전송하면 PCA9555를 통해서 신호를 내보낼 수 있으며 반대로 읽기 명령을 통해서 GPIO의 값을 가져올 수도 있다.

(2) SHT11

SHT11[18]은 온도와 습도를 동시에 측정하여 디지털 값으로 변환시켜 주고 이를 I2C를 통해서 읽어올 수 있는 칩이다. 이 칩은 TinyOS 2.0에서 사용할 수 있도록 이미 장치 드라이버가 만들어져 있다. SHT11은 변환이 완료되면 센서값을 읽을 수 있도록 DATA 핀이 LOW로 변한다. 이때 MCU는 이 신호를 외부인터럽트를 통해서 확인하게 된다. 하지만 Ubi-Mango 모트의 설계 구조상 외부 인터럽트 핀과 연결되어 있지 않아 사용할 수 없다. 그렇다고 루프를 사용하여 LOW로 변할 때까지 무한정 기다릴 수도 없다. 그래서 가상 타이머를 추가하여 DATA 핀이 LOW로 변했는지를 일정시간 간격으로 검사하게 된다. 핀이 LOW로 변했으면 가상 타이머를 정지하고 데이터를 읽어온다.

또 하나는 MG2400의 RF의 동작 상태가 송신중인지 수신중인지를 검사하는 핀과 SHT11의 CLOCK Pin이 서로 연결되어 있다. 그래서 SHT11을 가동할 때는 RF의 동작상태 검사기능을 해제하고 반대로 SHT11의 동작이 완료되고 RF를 사용할 때는 검사 기능을 활성화하도록 코드를 수정하였다.

사. 네트워크 프로토콜

TinyOS 네트워킹 그룹에서 공개한 것으로 현재 수집 트리 프로토콜 (CTP : Collection Tree Protocol), 파종 (Dissemination), Tymo, Deluge T2 등이 있으나 본 논문에서는 수집 트리 프로토콜과 파종의 이식을 수행하였다.

(1) 수집 트리 프로토콜

수집 트리 프로토콜은 네트워크상의 정보수집 경로가 트리구조를 형성하도록 하는 프로토콜이며 그림 9와 같이 배선엔진(Routing Engine), 전달엔진 (Forwarding Engine) 및 연결평가부 (Link Estimator)로 구성되어 있다.

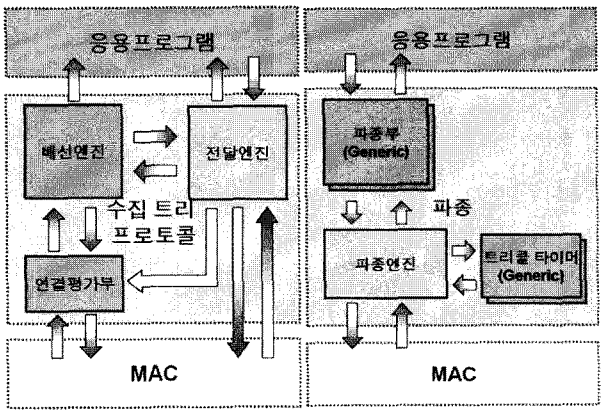



그림 9. 수집 트리 프로토콜
Fig. 9. CTP.

그림 10. 파종
Fig. 10. Dissemination.

연결평가부는 감지된 주변 노드들에 대한 신호 강도를 계산하여 우선순위를 결정하고 배선엔진에 추천한다.

배선엔진은 비콘 (Beacon) 송신을 수행하고 연결평가부로부터 전달받은 정보를 바탕으로 부모 (Parent) 노드를 결정한다.

전달엔진은 배선엔진이 결정한 부모 노드에 데이터를 전달하며 송신 완료 여부를 연결평가부에게 알려준다.

(2) 파종

파종은 네트워크상에서 변수값을 공유하기 위한 프로토콜이며 변수의 변화를 최대한 빠르게 전파하여 동기 시키되 전력소모가 크지 않도록 송신 횟수를 최소화하는 것이 목표이다. 구조는 그림 10과 같이 파종부 (Disseminator), 파종엔진 (Dissemination Engine) 및 트리클 타이머 (Trickle Timer)로 구성되어있다.

파종부는 네트워크상에서 공유하려는 변수에 대한 초기화, 업데이트 및 전파를 하기위한 컴포넌트이며 여러 개의 변수를 공유하기 위해서는 변수의 개수만큼 생성해야한다.

트리클 타이머는 Trickle[6]이라는 방식을 이용하며 한번 전파한 정보에 대해서는 더 이상 전파하지 않도록 하여 네트워크의 부하를 최소화 한다.

파종 엔진은 파종부와 트리클 타이머 사이에서 중재 역할을 한다.

## 3. 시스템 오동작의 원인

시스템이 오동작은 다양한 원인으로부터 발생한다. 기본적으로 회로에 공급되는 전원이 부족하거나 잡음에

의한 간섭이 발생할 수 있으며 정전기에 의한 회로 손상과 기판의 설계오류 및 칩 불량 등이 있다. 하지만 이러한 하드웨어 고장의 경우는 대부분 초기에 검출되며 충분한 검사 후에 시판되기 때문에 발생할 확률이 거의 없다. 그렇다면 가장 많은 시스템 오동작의 주요인은 무엇일까? 바로 소프트웨어이다. 아주 간단한 프로그램의 경우에도 간단한 초기화 실수 하나로 시스템이 정지하기도 한다. 특히, 새로운 플랫폼에 운영체제나 소프트웨어를 이식할 때에는 더 복잡한 문제에 대처할 수 있어야 한다.

## 4. 시스템동작의 오류 검출방법

### 가. 시뮬레이터를 이용한 오류 검출

시뮬레이터는 컴파일러에 포함되어있거나 칩 제작사가 무료로 배포하기 때문에 저렴하나 신뢰성은 보장하지 못할 수도 있다. 운영체제기반으로 동작하지 않는 시스템의 경우에는 소프트웨어 시뮬레이터를 이용해서 순차적인 소프트웨어 수행상의 문제를 해결할 수 있다. 정지위치를 지정하여 그 위치까지 실행한 후의 결과를 확인한다. 그밖에 시뮬레이션 수행도중 내부 데이터나 흐름을 변경할 수도 있다.

그러나 새롭게 개발되는 플랫폼의 경우 시뮬레이션은 주변장치를 포함하는 다양한 하드웨어의 모델이 개발되어야 하며 인터럽트의 처리와 같이 비동기 수행이 가능하도록 해야 한다. 대개의 경우 운영체계의 이식이 완료된 후에 이러한 시뮬레이터의 개발이 가능하게 된다.

### 나. LED를 이용한 오류 검출

운영체제를 새로운 플랫폼에 이식하는 과정에서 발생하는 오류를 적은 비용으로 검출할 수 있는 방법 중에 하나로 LED를 통한 단계별 오류검출 방법이 있다.

```
void __vector_8(void) interrupt 8
{
  P3_4 ^= 1;

  if(intStatus == 0x07) {
    P3_5 = 1;
    P3_6 = 0;
  }
  else {
    P3_5 = 0;
    P3_6 = 1;
  }
}
```

그림 11. LED출력을 통한 오류 검출코드
Fig. 11. Error detection code with displaying LED.

이 방법은 시스템 오류가 예상되는 지점의 통과 여부나 메모리의 값을 비교한 후에 그 결과를 LED를 통해서 확인하게 된다[19]. LED의 수가 3개 이상이면 특정 구간의 동작상태 (State)를 확인할 수도 있다. 하지만 확인이 된 지점부터 다음 지점을 확인하기 위해서는 코드를 수정하여 재 컴파일 후 실행하여만 확인할 수 있다.

그림 11은 인터럽트의 동작 상태와 ISR내에서 분기 방향을 알아내기 위한 코드이다. 인터럽트8이 주기적으로 발생한다면 P3_4의 LED는 계속 깜빡이게 되고 P3_5 및 P3_6의 LED중 어느 것이 켜지냐에 따라 인터럽트의 상태를 확인할 수 있다.

### 다. NesC 디버거 컴포넌트를 이용한 오류검출

Micaz[20] 등의 플랫폼 상에서 네트워크 프로토콜들을 개발할 때 NesC 디버거 컴포넌트를 개발하여 사용하였다. 이를 이용하여 디버깅을 수행할 경우 UART를 사용하기 때문에 LED를 이용한 방법보다 구체적인 확인이 가능하다. 하지만 정보를 패킷단위로 전송하기 때문에 많은 양의 RAM을 사용하게 되며 주 기능 컴포넌트와 디버거 컴포넌트를 연결해야 하는 불편함이 있다.

### 라. UART와 외부인터럽트를 이용한 오류 검출

본 논문은 NesC 디버거 컴포넌트를 이용한 방법보다 메모리를 훨씬 적게 사용하며 운영체제가 사용하지 않는 여분의 UART 채널을 이용하는 기법을 먼저 제안한다. 시스템의 동작 중에 UART를 통해서 실시간으로 출력을 하면 시스템에 큰 부하가 발생한다. 출력 시 부하를 최소화 하기위해서 시스템의 오류가 발생하거나 정지했을 때 외부 인터럽트를 인가하여 정보출력 기능을 가동시킨다. 하지만 인터럽트 신호로 제어할 수 있는 기능이 제한적이어서 필요한 정보만을 선택하여 확인하지 못하는 단점이 있다. 그림 12는 UART와 외부 인터럽트를 이용한 내부 변수 값을 확인하는 코드이다.

```
void __vector_0(void) interrupt 0
{
  sendByteuart(systemState);
  sendByteuart(checkCounter0);
  sendByteuart(intReg);
}
```

그림 12. UART를 이용한 오류 검출코드
Fig. 12. Error detection code with UART.

### 마. ART와 출력선택명령을 통한 오류 검출

NesC 디버거 컴포넌트를 이용하는 방법 및 앞서 제

안된 방법은 UART의 송신 기능만을 사용하고 있으나 UART 수신기능을 사용하면 UART 수신인터럽트를 이용하여 필요한 시점에만 변수 출력을 요청할 수 있을 뿐만 아니라 훨씬 다양한 디버깅 명령을 전달할 수 있다. 따라서 본 논문은 UART를 통해 모트에 출력 선택 명령을 전달하는 기법을 제한한다. 이는 UART를 통해서 (1) 출력할 변수를 선택하는 명령을 입력하여 원하는 정보만을 확인하는 대화형 디버깅을 함으로써 시스템 동작 중에 부하를 최대한 줄일 수 있다. 그리고 (2) 코드 전역에 걸쳐서 (3) 자료형에 상관없이 원하는 변수를 확인할 수 있으며 (4) 추가로 다른 변수를 확인하고 싶으면 설정파일 (Configuration File)에 해당 변수를 추가하여 스크립트를 실행하면 된다. 그리고 통신버퍼를 수십 바이트 정도만 사용하기 때문에 (5) 메모리를 절약할 수 있다. 오류가 발견됐을 때에는 스크립트로 변환된 코드를 바로 수정하여 컴파일 할 수 있다. 이는

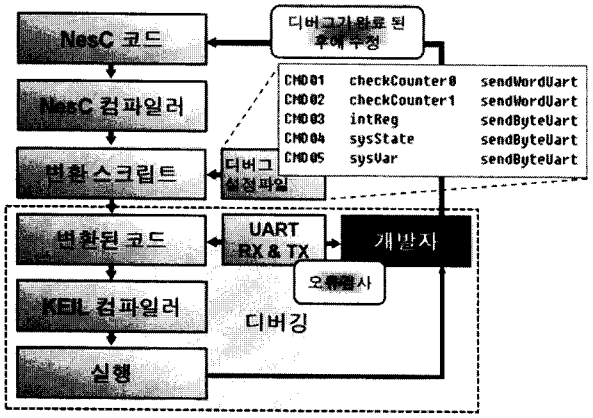



그림 13. 디버깅 진행과정
Fig. 13. Debugging process.

```
void runDebugcommand(void)
{
  if(command == CMD01) {
    sendWordUart(checkCounter0);
  }
  else if(command == CMD02) {
    sendWordUart(checkCounter1);
  }
  else if(command == CMD03) {
    sendByteUart(intReg);
  }
  else if(command == CMD04) {
    sendByteUart(sysState);
  }
  else if(command == CMD05) {
    sendByteUart(sysVar);
  }
  command = NOCMD;
}
```

그림 14. 오류검출을 위해 Configuration File로부터 자동 생성되어 입력된 명령 코드
Fig. 14. Automatically created command code for detecting error.

NesC 디버거 컴포넌트를 추가하는 것보다 업무가 수월하며 오류 유무를 정확하게 확인하기 전까지 NesC 코드를 수정하지 않아도 되기 때문에 다시 컴파일 할 경우 발생하는 시간을 최소화 할 수 있다. 이 방법은 TinyOS의 이식과정에서 발생했던 문제점을 해결하는데 결정적인 역할을 하였다.

진행 과정은 그림 13의 다이어그램을 통해서 알 수 듯이 변환 스크립트 실행 시에 디버그 설정파일을 첨부하면 그림 14와 같이 출력명령 처리코드를 자동으로 생성하여 삽입시킨다.

## Ⅲ. 실 험

본 장에서는 TinyOS 2.0을 MG2400 및 MG2455 플랫폼에 이식하면서 발생했던 이슈 및 디버깅 사례를 상술한다. 모든 이슈는 기존 코드를 수정하다가 주입된 것이 아니라 기존 코드를 새로운 플랫폼에 이식할 때 하드웨어 및 컴파일러가 서로 달라 발생한 것이며 새롭게 제안된 디버깅 방식을 통해 원인을 정확히 규명하고 신속하게 수정할 수 있었다.

표 1은 네트워크 프로토콜 응용예제에 대해 기존에 NesC 디버거 컴포넌트를 사용하였을 때와 본 논문에서 제시하는 디버깅 방법을 사용했을 때의 차이점을 나타

표 1. 수집 트리 프로토콜/파종에 대한 디버깅 방법 비교
Table 1. Comparison of each debugging methods on CTP/Dissemination.

| 방법 / 비교 | NesC | 본 논문 |
|---|---|---|
| RAM 사용 | 5.7 KB | 2.7 KB |
| ROM 사용 | 62.6 KB | 38.5 KB |
| 컴파일 과정 | NesC→스크립트→KEIL | KEIL |
| 컴파일 시간 | 22.6 초 | 2.7 초 |
| 오류 검출 사례 | ·CTP의 송·수신 오류 | ·가상 타이머 플래그의 초기화 문제<br>·ADC 동작상태 초기화 문제<br>·H/W MAC FIFO의 데이터 손상<br>·S/W MAC 버퍼의 포인터 손상<br>·RF 송신상태 오류<br>·RF 수신 오버런 발생<br>·CTP의 송·수신 오류<br>·파종 동작상태의 초기화 문제<br>·응용프로그램에서 UART로 출력 시 포인터 지정 오류 |

내고 있다.

본 논문에서 제시하는 디버깅 방법은 디버거 컴포넌트를 사용했을 때보다 데이터 메모리는 52.6%, 프로그램 메모리는 38.5% 적게 사용하였다. 그리고 컴파일 과정을 단축함으로써 시간을 약 8배 정도 빠르게 하였다. 이러한 높은 효율뿐만 아니라 버퍼나 포인터 관련 메모리 오류를 비롯한 다양하고 복잡한 오류를 검출하고 수정하는 효과가 기존 방법보다 탁월한 것으로 나타났다.

### 1. 타이머

MG2400과 MG2455의 구조에 맞게 설계된 타이머 컴포넌트를 이용하여 Blink 예제를 수행하여 보았다. 3개의 LED는 각각 1초, 2초, 3초로 깜빡이도록 프로그램되어 있다. 타이머 동작 시에 가상타이머의 플래그(Flag)가 초기화 되지 않아 발생했던 오류들을 앞에서 제시한 디버깅 방법을 이용하여 해결하였다. 그림 15는 제작된 보드에 MG2400을 장착하여 수행하는 그림이다.

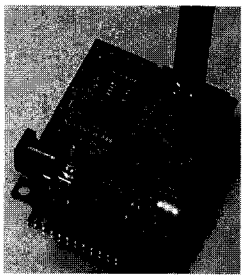


그림 15. Blink예제
Fig. 15. Blink Tutorial.

### 2. RF

MG2400과 MG2455의 구조에 맞게 설계된 MAC 컴포넌트를 이용하여 각 노드의 카운트값을 서로 교환하여 LED에 출력하는 BlinkToRadio 예제를 수행하여 보았다. 그림 16은 예제를 수행하는 모습을 나타낸다.

그리고 RF 동작 시 매우 혼잡한 상황 (Busy Traffic)에 대한 시스템의 신뢰성을 시험하였다. 시험 결과, 오류 증상은 20여개의 모트들 간에 통신이 이루어지고 있는 중에 짧게는 10분 이내에서 길게는 4시간 정도 지났을 경우 송·수신이 정지하는 증상이 나타났다. 때에 따라서는 송·수신 외의 기능까지 완전히 정지하였다. 시스템 오류의 주원인은 수신 버퍼의 포인터가 손상되는 것이었다. 포인터가 손상된 후에 시스템의 중요한 영역에 자료를 쓰면서 시스템이 정지하는 것이다. 포인터가 손상될 확률이 가장 높은 경우는 보통 메모리를

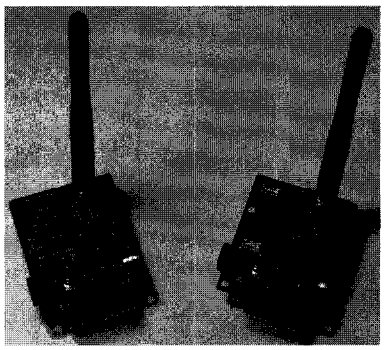


그림 16. BlinkToRadio예제
Fig. 16. BlinkToRadio Tutorial.

한 번에 일정한 크기로 복사하는 경우이다. 수신패킷의 시작부분에 패킷길이 (Packet Length)에 대한 정보가 존재하는데 이 부분이 손상되어 수신버퍼의 크기보다 많은 데이터를 복사하게 될 경우 오버런 문제를 발생시킬 수 있다. 먼저 LED를 통해서 RX FIFO의 오버런 여부를 확인하였다. 이 문제가 발생하였다는 것은 길이 (Length)를 포함해서 패킷 (Packet) 전체가 손상되거나 완전히 다른 값으로 교체될 수 있다는 증거이다.

이 문제를 해결하기 위해서 기존의 이식과정에 작성되어있던 패킷 오류 검사 코드 (자동 CRC, 자동 주소해석, 길이가 5보다 작을 경우, 길이가 128보다 클 경우, 잘못된 패킷) 외에 5단계 검사 코드를 추가 하였다. (1) 수신패킷 처리 작업이 종료될 때까지 패킷 수신을 금지하고 (2) 크기가 MAC 프레임의 최대 크기보다는 작지만 실제 할당된 패킷보다 클 경우에도 수신을 금지시켰다. 또한 (3) 크기 비교를 위해 복사해 놓은 패킷의 크기와 버퍼로 복사하기 바로 직전의 RX FIFO의 크기 값이 다를 경우와 (4) 잘못된 FCF (Frame Control Frame)의 검출을 하였다. 마지막으로 (5) 복사된 버퍼의 Pan ID와 목적지 주소를 비교하여 다를 경우도 수신을 금지 시켰다. 위의 디버깅 기법을 통해서 RF 수신

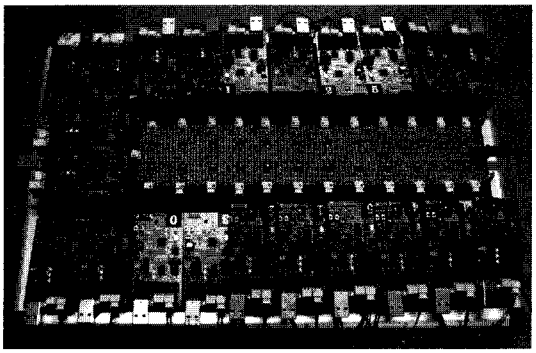


그림 17. TinyOS 2.0의 RF 동작에 대한 신뢰성 검사
Fig. 17. Reliability test of RF operation on TinyOS 2.0.

시 오버런문제로 길이정보 또는 패킷 전체가 손상되어 시스템에 피해를 주는 문제를 해결 할 수 있었다. 그림 17은 40 cm × 28 cm의 안에 22개의 모트가 모두 같은 PAN, 채널을 사용하되 2개씩 짝을 지어 송수신을 하도록 발신지 주소와 목적지 주소를 정하여 통신의 장애 여부를 시험하였다. 패킷의 크기는 17 Bytes(기본 11 Bytes + Payload 4 Bytes + 추가 Header 1 Byte + 추가 길이 1 Byte)이며 각 모트의 송신 주기는 약 10 msec 이다. 100시간 신뢰성 검사를 통과하였다.

### 3. 네트워크 프로토콜

앞에서 시험한 신뢰성 검사를 바탕으로 수집 트리 프로토콜과 파종 네트워크 프로토콜을 시험하였다.

먼저 수집 트리 프로토콜 구동 중에 전달엔진의 송신부에서 Ack수신 실패 시 무한 재송신을 하는 문제를 해결하였다. 전달엔진의 전달프레임과 배선엔진의 비콘 프레임 송신 시에 FCF의 플래그를 초기화 해주지 않아 불필요하게 Ack가 수신되기를 기다리거나 수신측에서 Ack를 송신하는 문제를 검출하여 해결하였다.

파종에서는 파종엔진의 구동 상태를 저장하는 변수를 초기화하지 않아 오동작한 문제와 파종할 자료(Data)의 포인터를 잘못 지정한 문제가 있어 이를 해결하였다.

그밖에도 Keil 컴파일러가 함수 내부에서 선언한 지역변수의 메모리를 스택영역이 아닌 정적영역에 할당하기 때문에 발생하는 문제가 나타났다. 일반적으로 함수의 내부에서 선언되는 지역변수는 스택영역에 할당되며 함수가 실행될 때마다 매번 스택영역에 새로 할당해야 한다. 그러나 Keil 컴파일러는 함수의 지역변수를 정적 메모리 영역에 할당하며 최적화 옵션에 따라 여러 함수가 그 영역을 공유하기도 한다. 비동기 인터럽트에 의한 함수 호출 시 다른 함수의 사용이 끝나지 않은 공유

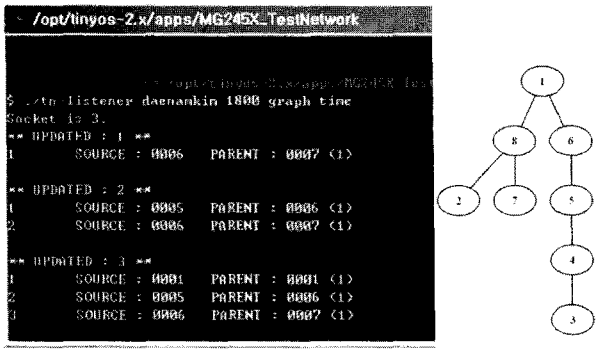



그림 18. 수집 트리 프로토콜
Fig. 18. Collection Tree Protocol.

영역을 침범하여 다른 함수의 지역변수 값들을 손상시키며 이것이 포인터인 경우 시스템을 정지시키는 심각한 오동작을 일으키기도 한다. 컴파일러의 최적화 레벨을 1로 낮추어 같은 영역을 공유하지 못하게 함으로써 문제를 해결하였다. 그림 18은 수집 트리 프로토콜을 통하여 트리형태의 경로를 통해 데이터가 수집되는 상황을 보이는 화면이며 SOURCE가 1인 노드가 수집 트르의 뿌리 (Root)이다. 생성된 트리 정보는 "dot" 프로그램을 이용하여 그래프로 보여줄 수 있다.

## IV. 결 론

소형 무선 센서네트워크 임베디드 시스템을 위한 새로운 플랫폼들이 개발되고 있으며 이들에 운영체계를 이식하는 과정에서 발생하는 시스템의 오류를 줄이기 위해서는 많은 노력이 필요하다. 본 논문은 국내업체에서 제작된 새로운 플랫폼에 TinyOS 2.0을 이식하는 과정에서 발생했던 여러 가지 이슈들을 도출하고 해결하는 방법을 제시하고 있다. 새로운 타이머의 설계와 스크립트 변환방법 추가를 통해서 플랫폼 및 툴체인 (Toolchain) 제약성을 극복하였으며 RF 통신 중의 오류 및 네트워크 프로토콜 오동작 등을 효과적으로 디버깅 할 수 있는 새로운 방법을 제시하여 성공적으로 해결하였다.

국내 무선센서네트워크 시장이 활성화 되고 많은 플랫폼들이 새로이 개발되었을 때 TinyOS나 그 외 운영체계를 이식하기 위한 많은 작업들이 필요할 것이며 본 연구진이 제시한 이슈 대처기법 및 디버깅 방법을 통해서 효과적으로 대응할 수 있을 것으로 기대된다.

또한 새로운 하드웨어에 이식된 TinyOS를 시뮬레이션 하기 위해서는 각 플랫폼에 맞는 TOSSIM (TinyOS Simulation)이 필요한데 이를 효율적으로 구축할 수 있는 방법을 연구할 계획이다.

## 참 고 문 헌

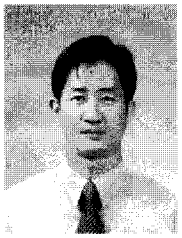
[1] TinyOS Working Group, "TinyOS 2.0 Documentation," http://www.tinyos.net/tinyos-2.x/doc/, Jul. 2007.

[2] C. Lynch, and F. O'Reilly, "PIC-based TinyOS implementation," Proceeedings of the Second European Workshop on Wireless Sensor Networks, pp. 378-385, Los Angeles, USA, Jan. 2005.

[3] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, and Philip Levis, "Collection," http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep119.html, Feb. 2007.
[4] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, Sukun Kim, Philip Levis, and Alec Woo, "The Collection Tree Protocol (CTP)," http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep123.html, Feb. 2007.
[5] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, and Philip Levis. "Four Bit Wireless Link Estimation," In Proceedings of the Seventh International Conference on Information Processing in Wireless Sensor Networks (IPSN), Cambridge, USA, Apr. 2007.
[6] Philip Levis and Gilman Tolle, "Dissemination of Small Values," http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep118.html, Feb. 2007.
[7] Philip Levis, Neil Patel, David Culler, and Scott Shenker. "Trickle: A Self-Regulating Algorithm for Code Maintenance and Propagation in Wireless Sensor Networks," In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation, San Francisco, USA, Mar. 2004.
[8] Kaisen Lin and Philip Levis. "Data Discovery and Dissemination with DIP," In Proceedings of the Proceedings of the Seventh International Conference on Information Processing in Wireless Sensor Networks (IPSN), St. Louis, USA, Apr. 2008.
[9] Radio Pulse Inc., "MG2400 Specification - Revision 1.13," RadioPulse Inc., 2006.
[10] Radio Pulse Inc., "MG2455-F48 Datasheet VER.1.0," RadioPulse Inc., 2007.
[11] Anders Egeskov Petersen, Sidsel Jensen, Martin Leopold, "Towards TinyOS for 8051," http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep121.html, Mar. 2006.
[12] Rainer Koster, "Design of a Real-Time Communication Service for Local Area Networks," Diploma Thesis, Dept. of Computer Science, University of Kiserslautern, Germany, Apr. 1998.
[13] Microsoft Corporation, "Guidelines For Providing Multimedia Timer Support," Microsoft Corporation, http://www.microsoft.com/whdc/system/CEC/mm-timer.mspx, Sept. 2002.
[14] Intel Corporation, "IA-PC HPET (High Precision Event Timers) Specification," Revision 1.0a, Intel Corporation, http://www.intel.com/hardwaredesign/hpetspec_1.pdf, Oct. 2004.
[15] G. Varghese, and A. Lauck, "Hashed and hierarchical timing wheels: efficient data structures for implementing a timer facility," IEEE/ACM Transactions on Networking, Vol. 5, Issue 6, pp. 824-834, Dec. 1997.
[16] 서창수, 이철희, "HBE-Ubi-Mango", (주)한백전자, 2007.
[17] PHILIPS, "PCA9555," PHILIPS, http://www.datasheetcatalog.org/datasheet/philips/PCA9555_3.pdf, May 2002.
[18] SENSIRION, "SHT11," SENSIRION, http://www.sensirion.com/images/getFile?id=25, May 2005.
[19] Jonathan W. Valvano, "Embedded Microcomputer Systems," Brooks/Cole, pp. 254-274, 2003.
[20] Crossbow Technology, "MICAz," Crossbow Technology, http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAZ_Datasheet.pdf.

## 저 자 소 개

김 대 남(학생회원)
2008년 인천대학교 전자공학과 학사 졸업.
2008년~현재 인천대학교 전자공학과 석사과정.
<주관심분야 : Sensor Networks, Embedded Systems>



김 교 선(정회원)
1986년 연세대학교 전자공학과 학사 졸업.
1988년 연세대학교 전자공학과 석사 졸업.
1998년 Ph.D. Department of Electrical & Computer Engineering, University of Massachusetts, Amherst, U.S.A.
1988년~2003년 삼성전자 CAE Center 주임, 선임, 책임, 수석연구원.
현재 인천대학교 공과대학 전자공학과 부교수
<주관심분야 : 상위수준합성, Reconfigurable Computation, Fault-Tolerance, Embedded Systems, Low-Power Design, Nanoelectronic Architectures>