Journal of The Korea Society of Computer and Information
Vol. 18, No. 7, July 2013

www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2013.18.7.047

임베디드 시스템 MCU 타이머 클록 펄스 동기화

이 형 봉*, 권 기 현**

Clock Pulse Synchronization of MCU Timers in Embedded Systems

Hyung-Bong Lee*, Ki-Hyeon Kwon**

요 약

임베디드 시스템에 구현되는 대부분의 어플리케이션들은 MCU가 제공하는 타이머를 사용한다. 타이머 사용의 목적은 실시간 운영체제의 소프트웨어 타이머 구현에서부터 센서의 워밍업이나 처리의 경과 시간 측정 등에 이르기까지 다양하다. 이들 어플리케이션들이의 시간 측정은 그 길이 뿐만 아니라 정밀도 측면에서 수 us~수 백 ms 정도로 그 범위가 다양하다. 이 논문에서는 타이머를 활용하는 과정에서 클록 펄스 비동기화로 인해 발생할 수 있는 오차 요인을 분석하고, 이러한 오차를 감소시키기 위한 타이머 클록 펄스 동기화 방안을 제시한다. 실험 결과, 32768Hz의 타이머를 8 분주한 4096Hz 타이머의 경우 약 230us까지의 편차가 발생하지만, 제안된 방법을 적용하면 타이머로 인한 편차를 10us 이내로 유지할 수 있다.

▸ Keywords : 임베디드 시스템, 타이머 측정 오차, 타이머 펄스 동기화, 분주

Abstract

Most of the applications implemented in embedded systems use timers equipped in MCU. The purposes of timer usage of the applications lie in a wide range of areas such as implementing software timers of real-time operating systems to measuring processing time of sensors. The elapsed times measured by the applications are various in length as well as in precision ranging from several us to several hundreds of ms. The paper analyzes the timing error factors caused by un- synchronizing timer clock pulse when timers are manipulated, and proposes a method of how to synchronize timer clock pulse to reduce the timing errors. As a result of an experiment, this

•제1저자 : 이형봉 •교신저자 : 권기현
•투고일 : 2013. 5. 21, 심사일 : 2013. 6. 12, 게재확정일 : 2013. 6. 27.
* 강릉원주대학교 컴퓨터공학과(Dept. of Computer Science & Engineering, Gangneung-Wonju National University)
** 강원대학교 전자정보통신공학부(Dept. of Electronics, Information & Communication Engineering, Kangwon National University)

paper shows that an error of 230us is reduced within 10us in case of appling the proposed method to a 4096Hz timer prescaled from 32768Hz by 8.

▸ Keywords : Emdedded Systems, Timer Measurement Error, Timer Clock Pulse Synchronization, Prescaling

I. 서 론

스마트 폰의 보편화로 대변되는 IT의 대변혁이 이루어지고 있는 오늘날 소프트웨어의 중요성이 그 어느 때보다 강조되고 있다. GPS(Global Positioning System) 센서를 비롯한 다양한 주변 장치와의 연동을 바탕으로 한 기발한 어플리케이션(어플 혹은 앱)들이 등장하고 있으며 그 잠재력 또한 무궁무진하다. 이러한 스마트 폰 어플리케이션들의 대부분은 단순한 응용 소프트웨어가 아니라 시스템 소프트웨어에 가깝다는 특징을 가지고 있다. 즉 센서, 터치 스크린, 블루투스 통신, UART(Universal Asynchronous Receiver and Transmitter) 통신 등 하드웨어를 제어하고, 대화적 콘솔이 존재하지 않는 환경에서 운용된다는 점에서 시스템 소프트웨어라고 말할 수 있는 것이다. 스마트 폰 어플리케이션 뿐만 아니라, 무선 통신 기반의 유비쿼터스 센서 네트워크(USN: Ubiquitous Sensor Network) 어플리케이션들은 운영체제나 개발도구가 제공하는 단순한 API(Application Programming Interface) 만으로는 구현될 수 없고, 센서 노드에 탑재된 MCU(Micro Controller Unit) 모듈에 직접 접근하는 경우가 많다는 점[1-3]에서 이 또한 시스템 소프트웨어에 속한다.

위와 같이 스마트 폰이나 USN 노드는 대화적 콘솔이 존재하지 않고, 어플리케이션들에게 다양한 하드웨어 직접 접근을 허용하여, 제한되고 특화된 기능을 발휘할 수 있는 환경을 제공한다는 점에서 전형적인 임베디드 시스템이라 말할 수 있다[4]. 임베디드 시스템의 핵심인 MCU에 SOC (System-on-a- chip) 형태로 장착된 하드웨어 모듈 중 활용 빈도가 가장 높은 것 중의 하나로 타이머(Timer)를 들 수 있는데, 타이머의 기본 기능은 어플리케이션들이 요구하는 경과 시간을 측정하여 인터럽트 신호로 알려주는 일이다. USN에서 무선 시분할 다중 접속 (Wireless Time Division Multiple Access) 형태의 프로토콜[2,5,6]과 같이 엄격한 시간 동기화가 요구되는 어플리케이션에서는 타이머의 정확도가 전체적인 시스템 성능에 큰 영향을 미친다. 이를테면 타이머 시간 측정 오차가 크다면 그 시간만큼 더 많은 활동 시간이 필요하기 때문에 에너지원이 극히 제한된 센서노드 구현에 어려움을 준다.

이 논문에서는 [2,5,6]의 무선 센서 네트워크 프로토콜 설계 및 구현 과정에서 나타난 타이머의 경과 시간 측정 오차 요인[7]을 분석하고, 이를 최소화시킬 수 있는 방안을 제시함으로써 타이머 활용 시 정확도 향상에 기여하고자 한다. 이를 위하여 II 장에서 Atmega2560[8] MCU와 WinAVR[9] C 컴파일러 및 AvrStudio4.19[10]를 기준으로 일반적인 타이머 활용 방법과 이 때 존재하는 오차를 규명하고, III 장에서 규명된 오차를 감소시키기 위한 방안을 제시하며, IV장에서는 제안된 방법의 성능을 검증한다. 그리고 마지막 V 장에서 결론으로 맺는다.

II. 타이머 설정 및 오차 분석

1. 일반적인 타이머 특성 및 사용 방법

1.1 타이머 클록

MCU에 장착된 타이머는 보통 CPU에 공급되는 클록을 공유한다. 본 연구에서 사용된 Atmega2560 MCU는 7.3728MHz XTAL을 사용하므로, 타이머 또한 이 클록을 공급받는다. 이 경우 클록 사이클 주기는 약 136ns (=1/7372800)이고 이는 매우 짧은 시간으로 분류될 수 있다.

1.2 타이머 카운터

모든 타이머에는 카운터 레지스터가 있어서 타이머에 공급되는 타이머의 클록 주기마다 1씩 증가한다. 타이머 카운터가 최대 값에 도달하면 다시 초기 값(=0)부터 다시 시작하게 되는데, 최대 값에서 0으로 변경되면서 타이머 종료 인터럽트를 발생시킨다. 8 비트 카운터의 경우 256 클록 주기 즉, 약 35us(=256/7372800)마다 인터럽트가 발생하고, 16 비트

카운터의 경우 65536 클록 주기(약 9ms=65536/9372800)마다 인터럽트가 발생한다. Atmega2560은 8 비트 타이머 2 개와 16비트 타이머 4 개를 제공한다.

1.3 타이머 동작과 정지

타이머는 기본적으로 클록이 공급되지 않아 유휴 상태(disabled)를 유지하고 있다가, 관련 제어 레지스터 설정에 의해 클록을 공급받는 순간 동작 상태(enabled)로 전환된다.

1.4 타이머 클록 분주

위에서 언급한 바와 같이 CPU 클록을 그대로 공급받는 경우 클록 사이클 주기가 필요 이상으로 짧아 타이머 운영에 부담이 될 수 있다. 이를테면 16 비트 타이머로 100ms를 측정하기 위해서는 총 12 번의 인터럽트를 처리해야 한다. 그러나 클록의 속도를 1/8로 감속시킬 수 있다면 단 두 번의 인터럽트로 처리를 마칠 수 있다. 이와 같이 타이머의 속도를 감속시키는 개념을 분주(prescaling)라 하는데, 그 원리는 원래의 클록 소스에서 몇 개의 주기를 그룹핑함으로써 그만큼 느린 클록으로 변형시키는 것이다. 이를 테면 원래 클록의 8 주기를 그룹핑 즉, 8 분주하여 하나의 주기로 변형하면 클록의 속도는 1/8로 감소한다. ATmega2560의 경우 타이머에 따라 8, 32, 64, 256, 1024 등의 분주를 허용한다. Atmega2560의 16 비트 타이머 3(16 비트 카운터)에 CPU 클록을 1024 분주한 7200Hz의 클록을 공급하여 기동시키는 프로시저와, 대응되는 인터럽트 서비스 루틴(ISR: Interrupt Service Routine)의 코드를 그림 1에 보였다.

```
void inline enable_timer3 ( )
{                                   // 7372800÷1024 = 7200
    TCCR3B = (1 << CS32) | (1 << CS30);  // 0x05
}                                   // prescaler is 1024
void inline set_timer3_count ( uint16_t  count )
{
    TCNT3 = -count;          // 65536 – count : remaining
    Timer_ovflag = 0;                          // to MAX
}
uint16_t inline get_timer3_count ( )
{
    return( TCNT3);          //  current value of Timer 3
}                                       //  counter register
ISR ( TIMER3_OVF_vect )
{
    Timer_ovflag = 1;          // time expiration flag
}
```

그림 1. Atmega2560의 타이머 기동 및 ISR 루틴
Fig. 1. Routines for Timer Operation and ISR of Atmega2560

2. 타이머 오차 시간 측정

2.1 오차 측정 실험 환경

그림 1의 7200Hz 타이머를 이용하여 고정된 카운트 수만큼의 경과시간을 오실로스코프로 측정하는 실험을 여러 번 실시하면 타이머 사용 시 발생하는 오차의 특성을 분석할 수 있다. 이 실험을 위하여 바쁜 대기(Busy Waiting) 형태로 3초 전후의 랜덤 처리 시간을 소모한 뒤, 이 타이머에 36 개의 클록(5ms=36/7200)을 설정하여 타임 종료 시간을 오실로스코프로 반복 측정하였고, 이 때, 오실로스코프의 트리거 입력원으로는 Atmega2560 GPIO(General Purpose IO) 핀 중 LED와 연결된 PORTL의 0번 핀에 대한 On/Off를 활용하였다. 그림 2에 타이머 오차 측정을 위한 실험환경을 보였고, 그림 3에는 발췌된 타이머 구동 코드를 보였다.

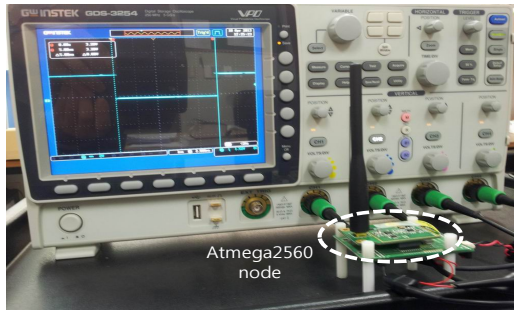

그림 2. 타이머 오차 측정 실험 환경
Fig. 2. The Experimental Environment for Measuring Error of a Timer

2.2 오차 측정 결과 및 분석

그림 2,3의 실험을 100회 이상 실시하여 타이머의 종료 시간을 관찰한 결과가 표 1과 같고, 그림 4에 최소 및 최대 값이 측정된 오실로스코프 모습을 보였다. 표 1에서 볼 수 있는 내용은 아래와 같다.

```
uint8_t  Timer_ovflag;
void main( )
{    :
     enable_timer3 ( );
     while ( 1 ) {
          r = rand ( ) % 255;       // delay for
          delay_ms ( 2875 + r );          // random time
          set_timer3_count ( 36 );
          Timer_ovflag = 0;
          LED_ON ( 0 );              // for triggering
          while ( !Timer_ovflag );  // wait flag set in
          LED_OFF ( 0 );                    // ISR of Fig. 1
     }
}
```

그림 3. 타이머 오차 측정을 위한 코드
Fig. 3. The Code for Measuring Error of a Timer

▣ 최대 오차

최대 측정오차는 0.12ms(=5.00-4.88) 즉, 120us이다. 이는 7200Hz의 한 콜록 사이클인 0.138ms(=1/7200) 즉, 138us에 가깝다.

▣ 측정값의 군집성

측정값들이 약 0.03ms(=30us) 단위로 형성된 단계적 군집을 이루고 있다.

표 1. 7200Hz 타이머에서 36 클록(5ms) 카운트 결과
Table 1. Result of Counting 36 Clocks(5ms) with a 7200Hz Timer

측정시간	4.88	4.91	4.94	4.97	5.00
측정횟수	16	26	19	22	17
평 균	4.93				

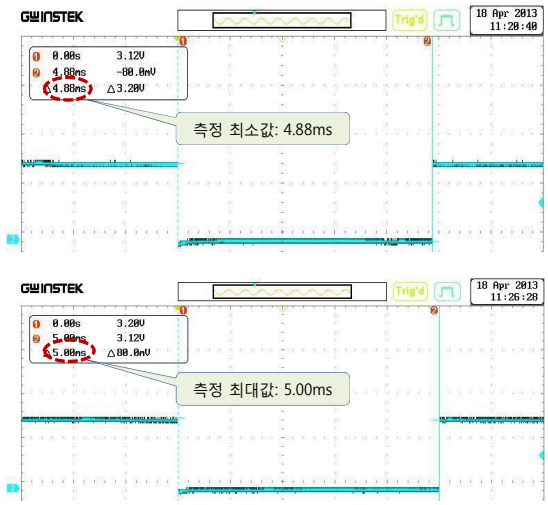

그림 4. 측정된 최대 시간과 최소 시간
Fig. 4. The Maximum and Minimum Time Measured

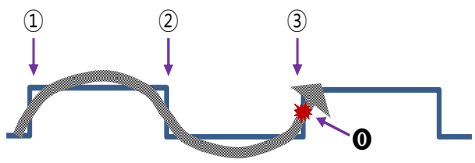

그림 5 타이머 측정 시간 오차 원인
Fig. 5. The Cause of Error of Timer Measurement

위의 특징들이 시사하는 바를 그림 5에 보였는데, 요약하면 현재 동작하고 있는 타이머의 클록 펄스 스케줄에 단순히 카운터만을 설정할 경우 최대 한 클록 주기만큼 짧은 시간으로 측정될 수 있다는 것이다. 그림 5는 ⓞ으로 표시된 상승에지(Rising Edge)에서 카운터 증가가 일어나는 타이머의 클록 펄스를 보인 것인데, 카운터를 설정하는 시점이 ①에 가까울수록 정확한 타임이 측정되고 ③에 가까울수록 측정 시간이 점점 더 짧아진다. 만약 시점 ②에서 카운터가 설정된다면 클록 펄스 주기의 약 1/2에 해당되는 시간만큼 측정 시간이 짧아진다. 또한 표 1의 측정값들이 단계적 군집을 이루는 이유는 측정에 사용된 오실로스코프의 측정 분해능과 관련이 있다. 즉, 본 실험 시 설정된 오실로스코프의 분해능은 10us이기 때문에 10us 단위의 단계적 군집이 이루어져야 하나, 커서(Cursor)를 수동으로 조작하는 과정에서 시각적 패턴이 뚜렷하도록 30us 단위의 단계적 군집으로 수집한 것이다.

III. 타이머 클록 펄스 동기화

그림 5에서 가장 바람직한 타이머 카운터 설정 시점은 ①인데, 이 시점은 타이머 카운터가 증가한 직후이다. 여기서는 이 성질을 이용하여 타이머 펄스에 동기를 맞추기 위한 방법을 제시한다.

1. 바쁜 대기 기반 타이머 클록 펄스 동기화

타이머의 카운터가 증가하는 시점을 인지하기 위한 하나의 방법으로 타이머 카운터를 쉬지 않고 읽으면서 그 값이 변경되었는지를 감시하는 기법을 들 수 있다. 즉, 바쁜 대기 경과 시간을 고려하지 않는 단계에서 일차적으로 타이머 클록 펄스와의 동기화를 이룬다. 다음으로 바쁜 대기 경과 시간을 고

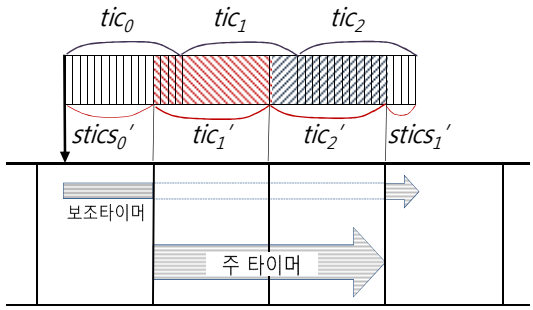

그림 6. 보조 타이머의 개념
Fig. 6. The Concept of Secondary Timer

려해야 하는데, 이 문제는 분해능이 더 높은 타이머를 사용해서 해결할 수 있다. 즉, 주 타이머 요청 카운터가 n이라면 주 타이머에서는 n-1 클록만을 운용하고, 나머지 한 클록은 보조 타이머로 소모하는 것이다. 보조 타이머의 카운터는 주 타이머의 클록 펄스와 동기가 이루어진 직 후 중지되어 있다가, 주 타이머의 종료와 함께 카운팅을 재개하여 나머지 시간을 측정한다. 그림 6에 3 개의 클록 카운터를 측정하기 위한 주 타이머보다 16배 빠른 보조 타이머의 개념을 보였다. 이 그림에서 주 타이머의 첫 클록 tic_0는 보조 타이머의 $stics_0$'와 $stics_1$'으로 재구성되어 보조 타이머에서 운영되고, 나머지 tic_1과 tic_2는 tic_1'와 tic_2'로 대응되어 주 타이머에서 운영된다. 보조 타이머의 $stics_0$'는 주 타이머의 클록 펄스와 동기를 맞출 때까지 운영되고, $stics_1$'는 주 타이머 종료 후 운영된다. 그림 7에 바쁜 대기 기반 타이머 클록 펄스 동기화를 위한 알고리즘을 보였다.

```
global  Timer_active, Timer_flag;
procedure_timer_busywaiting( n )
{
   mag = clock_speed_of_2nd_timer /
                        clock_speed_of_main_timer;
   counter_of_2nd_timer = mag;
   start_2nd_timer( );
   cnt =  current_counter_value_of_main_timer;
   while (cnt == current_counter_value_of_main_timer)
       ;        //  search the start point of a clock pulse
   Timer_active = 1;
   counter_of_man_timer = n - 1;
   stop_2nd_timer( );  // will be started again after the
}                                  // main timer expiration
ISR_of_main_timer( )
{
   if (Timer_active) {
       Timer_active = 0;
       start_2nd_timer( );  // spend the remaining time
   }                                    // of the 2nd timer
}
ISR_of_2nd_timer( )
{
   stop_2nd_timer( );
   Timer_flag = 1;        // expiration of the whole time
}
```

그림 7. 바쁜 대기 기반 타이머 펄스 동기화 알고리즘
Fig. 7. The Algorithm for Busy-waiting-based Timer Clock Pulse Synchronization

2. 인터럽트 기반 타이머 클록 펄스 동기화

주 타이머의 카운터 값이 변경되는 순간까지 기다리는 바쁜 대기 방법은 최대 한 클록 정도까지의 지연 처리를 감내해야 하므로 때로는 이 방법의 적용이 불가능할 수 있다. 이런 경우에는 주 타이머의 인터럽트를 활용하는 방안이 있다. 즉, 주 타이머의 카운터에 최대값을 설정하면 최소값 0으로 변경되면서 적어도 한 클록 주기 이내에 반드시 인터럽트가 발생하고, 이 시점이 바로 그림 5의 ①에 가장 가까운 시점인 것이다.

인터럽트 기반 타이머 클록 펄스 동기화에서도 첫 번째 클록 주기를 인터럽트가 발생할 때까지의 시간 즉, 주 타이머와의 동기가 이루어질 때까지의 시간과, 주 타이머가 종료된 후 나머지시간으로 재구성하여 보조 타이머로 운영하는 그림 6의 개념은 동일하게 적용된다. 그림 8에 인터럽트 기반 타이머 클록 펄스 동기화를 위한 알고리즘을 보였다.

```
global Timer_tics,    Timer_flag,
                      Timer_active1, Timer_active2;
procedure_timer_set_interrupt( n )
{
    mag = clock_speed_of_2nd_timer /
                        clock_speed_of_main_timer;
    counter_of_2nd_timer = mag;
    start_2nd_timer( );
    Timer_tics = n -1;
    counter_of_main_timer = MAX_VAL;
    Timer_active1 = 1;
}
ISR_of_main_timer( )
{
    if (Timer_active1) {      // will be started again after
        stop_2nd_timer( );     // the main timer expiration
        counter_of_main_timer = Timer_tics;
        Timer_active1 = 0;
        Timer_active2 = 1;
    }
    else if (Timer_active2)
        start_2nd_timer( );  // spend the remaining time
        Timer_active2 = 0;              // of the 2nd timer
    }
}
ISR_of_2nd_timer( )
{
     stop_2nd_timer( );
     Timer_flag = 1;        // expiration of the whole time
}
```

그림 8. 인터럽트 기반 타이머 펄스 동기화 알고리즘
Fig. 8. The Algorithm for Interrupt-based Timer Clock Pulse Synchronization

IV. 타이머 클록 펄스 동기화 검증 및 평가

1. 기본 기능 검증 및 평가

제안된 방법의 기본적인 기능을 검증하고 평가하기 위해 오차분석에 사용되었던 그림 1~4의 실험 과정을 제안된 방법을 적용하여 실시하고 결과를 분석한다. 이를 위하여 그림 7과 그림 8의 제안 알고리즘을 Atmega2560 환경에서 구현한 코드를 그림 9와 그림 10에 각각 보였고, 이들 코드를 구동하기 위해 그림 3을 변형한 주 모듈 코드를 그림 11에 보였다.

이들 코드를 이용하여 제안된 방법을 100회 이상 관찰한 결과 양쪽 모두 오차분석에서 보였던 120us의 변위 없이 5.03ms로 측정되었다. 이 결과에 대한 분석을 요약하면 다음과 같다.

```
extern Timer_active, Timer_ovflag;
void procedure_timer_set_busywaiting ( uint16_t  tics )
{
      uint16_t   cnt;
      set_timer5_count ( 128  ); // 921600 ÷ 7200 = 128
      enable_timer5 ( );
      cnt = get_timer3_count ( );
      while ( cnt == get_timer3_count ( ) );
      stop_timer5 ( );
      set_timer3_count  ( tics -1 );
      Timer_active = 1, Timer_ovflag = 0;
}
ISR ( TIMER3_OVF_vect )          // main timer
{
      if ( !Timer_active ) return;
      Timer_active = 0;
      enable_timer5 ( );
}
ISR ( TIMER5_OVF_vect )          // secondary timer
{
      stop_timer5 ( );
      Timer_ovflag = 1;             // time expiration flag
}
void inline enable_timer5 ( )     // 7372800÷8 = 921600
{     TCCR5B = (1 << CS51) ;  } // prescaler is 8
void inline stop_timer5 ( )
{     TCCR5B = 0;  }
void inline set_timer5_count ( uint16_t  count )
{     TCNT5 = -count;           }  // 65536 – count
```

그림 9. Atmega2560 바쁜 대기 기반 타이머 펄스 동기화 코드
Fig. 9. The Test Code for Busy-waiting-based Timer Clock Pulse Synchronization of Atmega2560

○ 어떤 경우에도 편차가 거의 없이 5.03ms가 일정하게 측정되어 안정된 타이머 운용이 가능하다.

○ 그림 3의 실험에서보다 길게 측정된 이유는 주 타이머의 첫 번째 카운터에서의 손실이 전혀 없고, 타이머를 설정하는 프로시저의 연산이 약간 늘어났기 때문이다.

```
extern uint8_t  Timer_active, Timer_ovflags, Timer_active2;
extern unt16_t Timer_tics;
void procedure_timer_set_interrupt ( uint16_t  tics )
{
      set_timer5_count ( 128  ); // 921600 ÷ 7200 = 128
      enable_timer5 ( );            // secondary timer
      Timer_tics = tics -1;
      Timer_active = 1;
      set_timer3_counter ( 0xffff  );
}
ISR ( TIMER3_OVF_vect )         // main timer
{
      if ( Timer_active ) {
            stop_timer5( );
            Timer_active = 0, Timer_active2 = 1;
            set_timer3_counter ( Timer_tics );
      } else if ( Timer_active2 ) {
            enable_timer5 ( );
            Timer_active2 = 0;
      }
}
ISR ( TIMER5_OVF_vect )          //secondary timer
{
      stop_timer5 ( );
      Timer_ovflag = 1;
}
```

그림 10. Atmega2560 인터럽트 기반 타이머 펄스 동기화 코드
Fig. 10. The Test Code for Interrupt-based Timer Clock Pulse Synchronization of Atmega2560

```
uint8_t     Timer_active, Timer_ovflag, Timer_active2;
uint16_t  Timer_tics;
void main( )
{     :
      enable_timer3 ( );
      while ( 1 ) {
            r = rand ( ) % 255;
            delay_ms ( 2875 + r ); // delay random time
            LED_ON ( 0 );
            procedure_timer_set_busywaiting ( 36 );
            // procedure_timer_set_interrupt ( 36 );
            while ( !Timer_ovflag );
            LED_OFF ( 0 );
      }
}
```

그림 11. 제안 방법을 구동하는 시험 코드
Fig. 11. The Test Code for Driving the Proposed Methods

○ 정확하게 5.00ms를 측정한다는 것보다는 편차 없이 언제나 일정한 시간을 측정해 내는 것이 중요하다. 즉, MCU에 사용된 XTAL의 정확도에 따라 타이머와 실 시간과의 오차는 피할 수 없다. 만약 정확하게 5.0ms를 원한다면 시행착오를 거쳐 보조 타이머의 카운터 값을 조금 가감하여 설정한다. 이 실험의 경우 보조 타이머 카운트를 28만큼 감한 결과 5.00ms를 얻을 수 있었다(28/921600≅30us).

○ 오실로스코프의 분해능을 더 높일 수 있다면, 보조 타이머의 클록 주기 범위 내에 존재하는 오차가 관찰될 수 있을 것이다.

2. 무선 TDMA 프로토콜에서의 적용

제안된 방법의 실질적인 효용성을 검증하기 위해 무선 TDMA 프로토콜의 하나인 LSNP(Linear-Wireless Sensor Network Protocol)[5]에서의 적용결과를 분석한다.

2.1 LSNP의 타임 슬롯 및 시간 동기화

LSNP는 그림 12와 같이 터미널 노드에서부터 싱크 노드까지 송신(TX)과 수신(RX) 구간이 겹쳐지도록 타임 슬롯을 차례로 할당하여 각각의 노드가 자신의 타임 슬롯에서 깨어나 이전노드로부터의 데이터를 수신 받아 다음 노드로 전달하는 단방향 가상 선로를 구성한다. 자신의 타임 슬롯이 지나면 해당 노드는 다음 주기까지 휴면에 들어간다. 이 때, 휴면 시간의 정확성이 따라 타임 슬롯의 크기를 좌우하게 되는데, 만약 정확하지 못하다면 타임 슬롯에 여유 시간을 많이 두어 크게 설정해야 하고 이는 곧 많은 에너지 소모로 귀결된다.

2.2 LSNP의 타이머 운용 및 실험

LSNP는 활성 구간(active period)에서는 빠른 MCU 클록으로 동작하는 921600Hz 타이머(Timer 5)를 사용하고, 휴면 상태에 진입하면 MCU 클록이 아닌 외부의 느린 클록으로 동작하는 4096Hz 타이머(Timer 2)를 사용한다. 그림 13에 LSNP의 휴면 타이머를 설정하는 알고리즘을 보였는데, 빗금 친 부분이 느린 타이머의 클록 펄스 동기를 기다리는 제안 부분이고, 기존의 루틴에서는 존재하지 않는다. 기존 루틴과 제안 부분이 추가된 두 가지 알고리즘을 사용했을 경우 각각에 대하여 인접한 두 노드의 RX 구간이 시작하는 시점의 시차를 측정하였다. 여기서 RX 구간은 921600HZ 타이머로 9450 클록이며 이는 약 10.25ms(=9450/921600)이다.

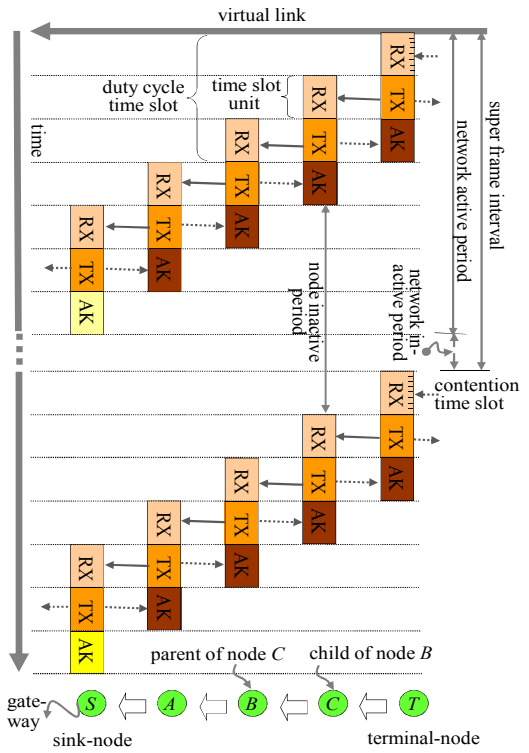

그림 12. LSNP의 시간 동기화
Fig. 12. The Time Synchronization of LSNP

```
global Timer_flag, Timer_active;
procedure_LSNP_sleep_timer_set( n )
{
    uint32_t   cnt;
    cnt = counter_of_timer2;
    while ( cnt == counter_of_timer2 );
    stop_timer5( );            // 921600 ÷ 4096 = 225
    counter_of_timer2 = counter_of_timer5 ÷ 225;
    counter_of_timer5 = counter_of_timer5 –
                        counter_of_timer2 × 225;
    Timer_active = 1, Timer_flag = 0;
    while(!Timer_flag) cpu_sleep( );
}
ISR_of_timer2( )        // ISR of the main timer
{
    if (Timer_active) {
        Timer_active = 0;
        enable_timer5( );
    }
}
ISR_of_timer5( )        // ISR of the 2nd timer
{
    Timer_flag = 1;   // expiration of the whole time
}
```

그림 13. LSNP의 시간 동기화 알고리즘
Fig. 13. The Time Synchronization Algorithm of LSNP

2.3 실험 결과 및 분석

그림 13에 보인 두 가지 방법을 LSNP에 적용하여 실시한 실험 결과를 표 2에 비교해 보였다. 이 표에서 보는 바와 같이 제안된 방법의 타이머 측정 시간 편차가 기존 방법의 230us보다 월등히 우수한 10us 이내 임을 알 수 있다.

표 2. LSNP에서의 제안 방법의 효과 테스트 결과
Table 2. The Result of the Efficiency Test of the Proposed Method in LSNP(단위: ms)

적용 방법	최 소	최 대	편 차
기존 방법	10.19	10.42	0.23
제안 방법	10.31	10.32	0.01

V. 결 론

임베디드 시스템에서 타이머는 어플리케이션의 질에 영향을 미치는 매우 중요한 디바이스 중의 하나이다. 일반적인 타이머 사용법에 의하면 타이머의 분해능만큼의 최대 편차를 감수해야 한다. 특히 에너지 소모를 줄이기 위해서는 가급적 분해능이 낮은 타이머를 사용해야 하는데, 이 경우 편차가 커서 심각한 소프트웨어 설계상의 제한 요인이 된다. 이 논문에서는 타이머의 클록 펄스에 동기를 맞춤으로써 타이머 측정 시간의 편차를 크게 줄이는 방안을 제안하고 그 효용성을 평가하였다. 평가에 사용된 무선 TDMA 프로토콜에서의 실험 결과 230us 편차를 약 1/20인 10us 이내로 감소시킬 수 있음을 보였다. 이와 같이 제안된 방법이 임베디드 시스템 타이머를 활용하는 어플리케이션의 질 향상에 기여할 것으로 기대된다.

참고문헌

〔1〕 SungHak Chung, "A Study on the Improvement Alternatives using USN Technology on Bicycle and Infrastructures", Journal of The Korea Society of Computer and Information, Vol. 15, No. 8, pp. 173-180, Aug. 2010.

〔2〕 Hyung Bong Lee, Lae-Jeong Park, Jung-Ho Moon, Tae-Yun Chung, "Design and Implementation of a TDMA-based Bidirectional Linear Wireless Sensor Network", Journal of KIISE : Computing Practices and Letters, Vol. 14, No. 4, pp. 341-351, Jun. 2008.

〔3〕 Hyung-Bong Lee, Jung-Ho Moon, and Tae-Yun Chung, "An Image-based Remote Snow Height Measurement System using a USN", Journal of IEMEK, Vol 2, No. 2, pp. 76-85, Nov. 2010.

〔4〕 C. Hallinan, "Embedded Linux Primer, Second Edition" PRENTICE HALL, pp. 9-35, 2011.

〔5〕 Hyung-Bong Lee, Ki-Hyeon Kwon, Lae-Jeong Park, Tae-Yun Chung, and Qishi Wuo, "A Lightweight Lap Time Measurement System for Alpine Ski Sport using a TDMA-based Linear-Wireless Sensor Network", International Journal of Distributed Sensor Network(IJDSN), Vol. 2012, pp. 1-15, Mar. 2012.

〔6〕 Hyung-Bong Lee, Lae-Jeong Park, Sung-Wook Park, Tae-Yun Chung, and Jung-Ho Moon, "Interactive Remote Control of Legacy Home Appliances through a Virtually Wired Sensor Network", IEEE Transactions on Consumer Electronics, Vol. 56, Issue 4, pp. 2241-2248, Dec. 2010.

〔7〕 Hyung-Bong Lee, "Synchronization of Timers in Embedded Systems", Proceedings of the 39th KIPS Spring Conference 2013, Vol. 20, No. 1, pp. 13-14, May 2013.

〔8〕 Atmel Corporation, "8-bit Microcontroller with 64K/128K/256K Bytes In-System Programmable Flash", Atmel Documents, 2012. (http://www.atmel.com/Images/doc2549.pdf, accessed May. 2013)

〔9〕 WinAvr Projects, http://winavr.sourceforge.net/, accessed May 2013.

〔10〕 Atmel Corporation, "AVR Studio 4.19", Atmel Tools, 2012. (http://www.atmel.com/tools/STUDIOARCHIVE.aspx, accessed May. 2013)

저 자 소 개

이 형 봉

1984: 서울대학교 계산통계학과 이학사.
1986: 서울대학교
계산통계학(전산과학)과
이학석사.
2002: 강원대학교 컴퓨터과학과
이학박사
1986~1993: LG전자 컴퓨터연구소
선임
1994~1998: 한국디지탈(DEC Korea)
책임
1999~2003: 호남대학교
정보통신공학부 조교수
2004~현재: 강릉원주대학교
컴퓨터공학과 교수
관심분야: 임베디드 시스템,
센서 네트워크,
데이터마이닝 알고리즘
Email : hblee@gwnu.ac.kr

권 기 현

1993: 강원대학교
전자계산학과 이학사
1995: 강원대학교
전자계산학과 이학석사
2000: 강원대학교
컴퓨터과학과 이학박사
1998~2002: 동원대학
인터넷정보과 교수
2002~현재: 강원대학교
전자정보통신공학부 교수
관심분야: 패턴인식, 미들웨어,
임베디드 소프트웨어
Email : kweon@kangwon.ac.kr