JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 24 No. 11, pp. 21-27, November 2019
https://doi.org/10.9708/jksci.2019.24.11.021

EXCUTE REAL-TIME PROCESSING IN RTOS ON 8BIT MCU WITH TEMP AND HUMIDITY SENSOR

Ki-Su Kim*, Jong-Chan Lee*

*Student, School of Computer Information Engineering, Kunsan University, Kunsan, Korea
*Professor, Dept. of Computer Information Engineering, Kunsan University, Kunsan, Korea

[Abstract]

Recently, embedded systems have been introduced in various fields such as smart factories, industrial drones, and medical robots. Since sensor data collection and IoT functions for machine learning and big data processing are essential in embedded systems, it is essential to port the operating system that is suitable for the function requirements. However, in embedded systems, it is necessary to separate the hard real-time system, which must process within a fixed time according to service characteristics, and the flexible real-time system, which is more flexible in processing time. It is difficult to port the operating system to a low-performance embedded device such as 8BIT MCU to perform simultaneous real-time. When porting a real-time OS (RTOS) to a low-specification MCU and performing a number of tasks, the performance of the real-time and general processing greatly deteriorates, causing a problem of re-designing the hardware and software if a hard real-time system is required for an operating system ported to a low-performance MCU such as an 8BIT MCU. Research on the technology that can process real-time processing system requirements on RTOS (ported in low-performance MCU) is needed.

▸ **Key words**: Real Time, IoT RTOS, IoT System, Real Time Os, Real Time System, IoT Sensor

[요 약]

임베디드 시스템에서는 서비스 특성에 따라 정해진 시간 내에 처리해야하는 하드 실시간 시스템과 처리 시간이 더 유연한 유연한 실시간 시스템을 분리해야합니다. 실시간을 동시에 수행하기 위해 운영 체제를 8BIT MCU와 같은 저 성능 임베디드 장치로 이식하는 것은 어렵습니다. RTOS (실시간 OS)를 사양이 낮은 MCU에 포팅하고 여러 작업을 수행 할 때 실시간 및 일반 처리 성능이 크게 저하되어 8BIT MCU와 같은 저 성능 MCU로 포팅 된 운영체제에 하드 실시간 시스템이 필요한 경우 성능 저하로 인해 하드웨어 및 소프트웨어를 다시 설계하는 문제가 발생되고 있습니다. 저성능 MCU에 이식 된 RTOS (저 성능 MCU로 포팅)에서 실시간 처리 시스템 요구 사항을 처리에 대하여 연구하고 프로세스 스케줄링에 대하여 연구가 진행되었습니다.

▸ **주제어**: 실시간, IOT실시간 운영체제, IOT 시스템, 실시간 운영체제, 실시간 시스템, IOT센서

• First Author: Ki-Su Kim, Corresponding Author: Jong-Chan Lee
*Ki-Su Kim (vennomnight1@kunsan.ac.kr), Dept. of Computer Information Engineering, Kunsan University
*Jong-Chan Lee (chan2000@kunsan.ac.kr), Dept. of Computer Information Engineering, Kunsan University
• Received: 2019. 08. 30, Revised: 2019. 09. 19, Accepted: 2019. 09. 21.

Copyright © 2019 The Korea Society of Computer and Information
http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

4차 산업 스마트 팩토리 기술 중 사물인터넷(Internet of Things, 이하 IoT) 기술은 온도, 습도, 생산량과 같은 데이터 수집에 사용하는 기술로서, 최근에서는 인공지능과 결합되어 오토 튜닝 PID 최적 변숫값을 찾는 기술, 빅데이터 기반의 머신러닝이 활용된 부품 교환주기 예측 기술 등과 결합되고 있다.

인공지능과의 결합 시에 관리 및 대응에서 편리를 제공하지만, IoT 장비가 데이터 수집과 인공지능의 복잡한 작업을 요구하기 때문에, IoT 장비의 MCU(Micro Controller Unit)의 단일 처리로는 요구 성능을 충족시키기 어렵다. 특히 저 클럭 MCU에 운영체제(operating system; 이하 OS)가 이식되면 실시간 처리와 다수의 태스크 작업 시에 종료 시점을 놓치는 문제가 발생된다. 따라서 더욱 복잡한 작업의 효율적인 처리를 위하여 고성능 MCU를 필요로 한다. 고성능 MCU를 사용하면 사용자 요구와 실시간 처리 요구를 충족시킬 수 있다. 하지만 고성능 MCU일수록 클럭이 상승하므로 하드웨어 설계 비용이 증가하고 PCB 보드 설계 과정에서 방사 노이즈, 간섭(Cross Talk) 등이 발생한다. 클럭이 고속일수록 간섭에 의한 동작 불능 문제가 발생하므로 다층 기판, 배선 라우팅, 임피던스 매칭, 귀환 경로(return current path) 설계에 특히 주의해야 한다. 특히 산업현장의 IoT 장비는 외부 노이즈 영향에 민감하고 방사 노이즈로 인하여 다른 기기에 치명적인 문제를 발생시킨다. 따라서 저 클럭 MCU를 탑재한 IoT 장비를 개발하고, 운영체제를 이식하고, 다중 프로세스 처리로 사용자 요구의 충족과 실시간 처리 문제를 효과적으로 적용하면 방사 노이즈 저감 등의 문제를 해결할 수 있다[4-5].

WinCE나 리눅스에는 선점형 커널이 존재하지만 반드시 실시간 성 작업을 처리하기 위해서 사용되는 것은 아니다. 리눅스는 커널 자체에 실시간 기능을 지원하지 않지만, 실시간 기능을 부가한 NI Linux Real-Time도 상용화되어 사용 중에 있다. 본 논문에서는 FREE RTOS 선점형 커널을 저 클럭 8BIT MCU에 포팅하고, 이를 통하여 경성 실시간 처리를 구현하고자 한다.

여기서 FREE RTOS가 경성 실시간 처리에 적합한 특징을 기술하면 아래와 같다[1-3].

- 다양한 시스템에 이식이 가능하다. 대표적으로 ARM, AVR, X86, PIC, TI 등이 있다.
- 주 기억장치(RAM)에 전체 커널을 저장할 수 있으므로, WinCE나 Linux의 적재 시 요구되는 큰 메모리 없이 이식이 가능하다.
- 고성능의 하드웨어가 필요 없으므로 저전력 시스템에 적용이 가능하다.
- 선점형 커널을 지원하기 때문에 수행 중인 태스크를 강제 중단하고 우선권을 확보할 수 있다.
- 태스크의 생성, 메모리 관리, 멀티태스킹을 지원하고 기존의 인터럽트 타이머 방식의 펌웨어 소프트웨어 개발 시의 약점을 보완할 수 있다.

본 논문의 FREE RTOS는 오픈소스 실시간 운영체제이며 간편한 이식성, 간결함을 가진다. 코어 소스 코드가 4,000라인 이내이고, 불가피한 몇몇 코드를 제외하고는 대부분의 코드가 c언어로 작성되므로 이식성이 뛰어나다. 현재 8051, AVR, PIC18, H8, MSP430, HCS12, PIC24, ARM Cortex-M3, ARM7, ARM9, AVR32, ColdFire, x86 등의 다양한 8 bit, 16 bit, 32 bit 프로세서에 이식되어 사용되고 있다[6-7]. 본 연구는 8 BIT AVR128A MCU에 이를 적용하고, 경성 실시간이 필요한 습·온도 DHT11 센서가 멀티태스크를 수행하면서 실시간 처리 시의 문제점을 해결할 수 있는 방법을 제시한다.

II. Preliminaries

1. RTOS Porting To Avr128A

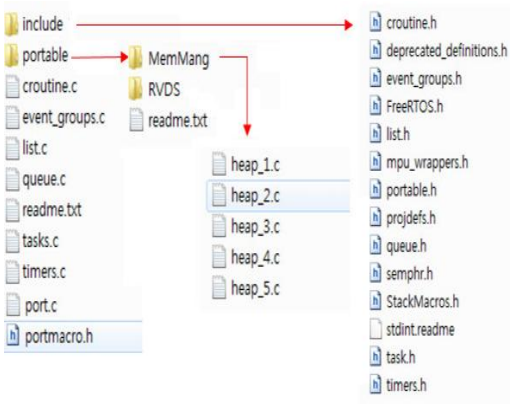

Fig. 1. Free RTOS Architecture

그림 1과 같이 FREE RTOS 포팅 구조를 구성하고 컴파일 하면 RTOS 포팅이 완료되고, FreeRTOSConfig.h 헤더 파일을 통하여 여러 가지 옵션을 사용할 수 있다. 본 논문에서 적용한 Atmega AVR128A은 8Mhz 외부 수정 진

동자를 사용하고 FreeRTO SConfig.h 헤더 파일에서 CPU 클럭을 8Mhz로 수정하고 포팅 하였다. CPU 클럭은 기준 시간으로, TICK_RATE는 1000Hz로, vTaskDelay 매크로 설정을 수행하였다. 이 외에 port.c 파일의 인터럽트 부분을 수정하였다. FREE RTOS는 디폴트 설정으로 AVR타이머1을 사용하고 있다[2].

그림 2와 같이 SIG_OUTPUT_COMPARE1A 함수에서 포팅타겟 MCU의 인터럽트 벡터와의 동일 여부를 확인한다. 벡터 명이 다르면, 불필요한 장소로 이동하므로 OS가 동작하지 않는다. 벡터명은 avr-toolchain을 통하여 확인하고 각 타겟 MCU별 io를 정의한 헤더 파일을 참조한다. 그림 2와 같은 벡터 함수는 iom128.h 헤더 파일의 벡터 테이블에 정의되어 있다.

```
void SIG_OUTPUT_COMPARE1A(void)
__attribute__((signal,naked));
void SIG_OUTPUT_COMPARE1A(void)
{
    vPortYieldFromTick();
    asm volatile ("reti");
}
```

Fig. 2. Free RTOS Timer Handler

본 논문에서 사용할 타겟 보드(Target Board)는 AVR128A이므로 해당 헤더 파일인 iom128a.h을 찾은 후 port.c에 있는 타이머1 벡터함수의 이름을 치환한다. 그림 3에 avr128a의 벡터 주소 명을 보인다.

```
#define TIMER1_COMPC_vect  _VECTOR(24)
#define TIMER1_COMPC_vect_num  24
```

Fig. 3. Timer1 Compare Match Vector Number

해당되는 벡터의 이름을 FREE RTOS의 port.c 파일에서 해당 명을 그림 4와 같이 치환한다.

```
#define SIG_OUTPUT_COMPARE1A
TIMER1_COMPA_vect
```

Fig. 4. Replace Macro Name to Vector 24 In port.c File

함수 치환으로 스케줄링 인터럽트인 타이머1의 동작이 정상적으로 동작하고, FREE RTOS를 구동하기 위한 포팅 작업은 완료된다. 또한 다수의 부가 기능을 사용하기 위해서는 freertos.org를 통하여 해당되는 레퍼런스를 확인 후, FreeRTOSConfig.h 헤더 파일에 정의되어 있는 매크로의 사용 여부를 체크하면 된다.

2. RTOS Porting Verification And Debugging Procedure

RTOS의 소스 코드의 컴파일 오류가 없을지라도 운영체제가 정상적으로 포팅 되었는지의 확인 절차가 필요하다. FreeRTOSConfig.h 헤더 파일의 타이머 틱(TICK_RATE)에 스케줄링이 일어나는 시점을 해당 주파수로 설정한다. 본 논문에서는 1000HZ로 설정된 부분을 500HZ로 변경하였고, 타이머 틱의 인터럽트를 비교 출력으로 설정하고 테스트를 진행하였다. 비정상적인 동작을 할 수 있으므로 실제 스케줄링에서 실행되는 타이머 틱은 하드웨어 테스트를 실시하여 확인하였다.

그림 5와 같이 TICK_RATE 500HZ의 파형을 볼 수 있다. AVR의 TIMER1출력을 하드웨어적으로 비교 매치 OC1A 출력으로 설정을 해줘야 하며, 본 논문에서는 토글 모드로 설정하여 진행하였다[8].

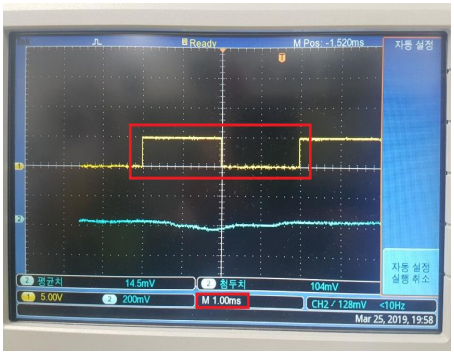

Fig. 5. Signal Of 500Hz Tick_Rate

그림 6과 같이 avr128의 데이터시트를 참조하면 16_bit 타이머 카운터 레지스터 설정을 확인할 수 있고, TCCR1A 레지스터의 값을 TCCR1A |= (1 << COM1A0); 으로 설정하였다.

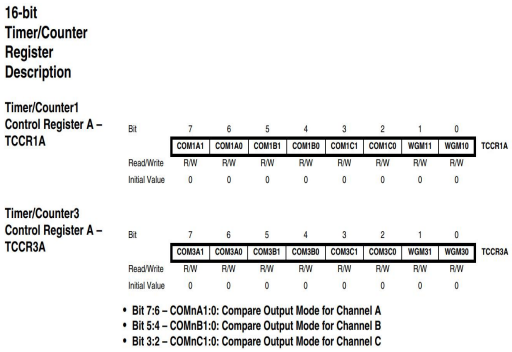

Fig. 6. 16-bit Timer/Count Register

현시점까지의 작업을 바탕으로 FREE RTOS 이식 및 타이머 인터럽트의 작동 여부와 스케줄링 동작 여부를 실험

하였다. FreeRTOSConfig.h 헤더 파일의 TICK_RATE의 주파수를 기존의 속도 실험에 필요한 원래의 값 1000HZ으로 다시 재설정하고 경성 실시간 시스템의 시험을 실시하였다.

```
#define configUSE_IDLE_HOOK 1
#define configUSE_TICK_HOOK 0
#define configCPU_CLOCK_HZ
((unsigned long) 8000000)
#define configTICK_RATE_HZ ((TickType_t) 1000)
#define configMAX_PRIORITIES (4)
#define configMINIMAL_STACK_SIZE
((unsigned short) 200)
#define configTOTAL_HEAP_SIZE
((size_t ) (1500))
```

Fig. 7. Config Tick_Rate In FreeRTOSConfig.h File

그림 7과 같이 Tick_Rate_Hz를 1000으로 설정하였다.

Fig. 8. Signal Of 1000Hz Tick_Rate

그림 8과 같이 실제 출력 파형을 오실로스코프로 하였다. 구형파의 주기는 2ms로 측정되며 1000Hz로 정상 동작되는 것을 확인하였다.

Fig. 9. Real Board Waveform Location

그림 9와 같이 오른쪽 위 박스의 GND 핀과 왼쪽 아래의 박스 OCA1/PB5의 핀 이 실제 출력파형의 위치이다.

III. Experiment to Real Time OS

센서를 DHT11로 선정하고 실시간 테스트를 진행한다. 실시간 테스트에 사용한 DHT11 센서는 온도 및 습도 측정 센서로서, HIGH, LOW신호를 HIGH 상태의 시간으로 측정하는 논리를 구별할 수 있다. 샘플링 시간 간격을 벗어나면 데이터가 훼손되는 문제가 발생되며, 비동기 프로그램 코드가 아니면, 무한루프의 현상이 발생하므로 정확한 실시간 샘플링이 요구된다. 그림 10은 DHT11의 데이터시트 상의 통신 스펙이다.

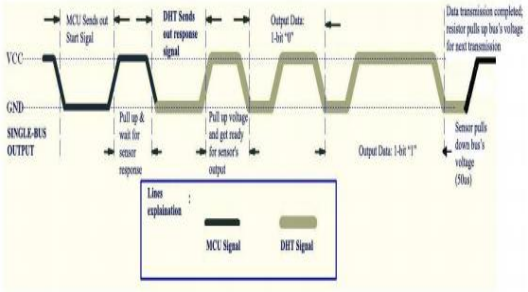

Fig. 10. Communication Specification Of DHT11

그림 10와 같이 데이터시트의 확인 과정 후에, 그림 11의 로직 레벨과 타이밍을 데이터시트와 같이 개발해야 한다.

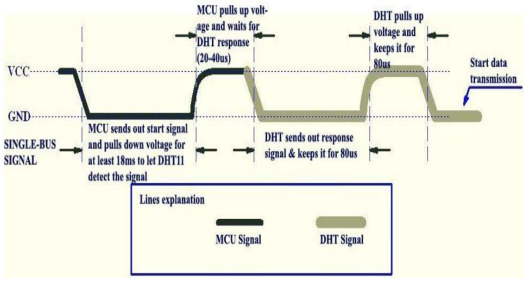

Fig. 11. Logic Timing Of DHT11

데이터시트 로직의 VCC 부분은 HIGH상태에서 LOW신호로 로직 레벨이 변화하면서 데이터 요청 시작을 알리는데, DHT11 신호의 감지까지 최대 18ms의 시간이 필요하다. 다음으로 약 20~40us 입력 신호를 풀업상태로 만들고 DHT11의 센서 데이터가 MCU 입력핀에 HIGH, LOW 신호로 응답한다. HIGH, LOW 신호는 단순히 VOLTAGE-LENGTH로 측정하고, LOW 데이터일 경우 HIGH의 신호가 약26~28us 동안 지속되고 HIGH 신호일 경우 HIGH의 신호가 약 70us 동안 지속된다. 즉 이 시간은 로직을 구분하는 신호이기 때문에 정확히 측정하지 못하면 데이터를 신뢰할 수 없는 문제가 발생한다. 그림 12는 DHT11의 정상 파형이다.

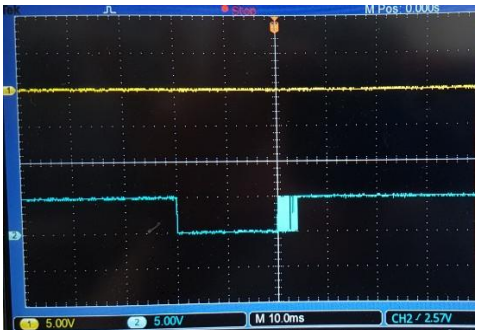

Fig. 12. Correct Waveform Of DHT11

FREE RTOS에서 두 개의 태스크를 생성 후 한 개의 태스크는 반목문으로 무한 루프의 상태로 만들고 두 번째 태스크에서 DHT11의 로직 파싱을 수행하였다. 스케줄링에 의해서 태스크 전환이 진행되고, 센서의 로직 파싱 중에 타이머 인터럽트에 의해서 인터럽트 루틴으로 분기되어 실시간성을 훼손하는 현상이 발견되었다. 초기에 설정한 주기마다 태스크 전환 상태이며, 센서 태스크가 원하는 신호를 측정하지 못하는 현상을 그림 13을 통하여 확인하였다.

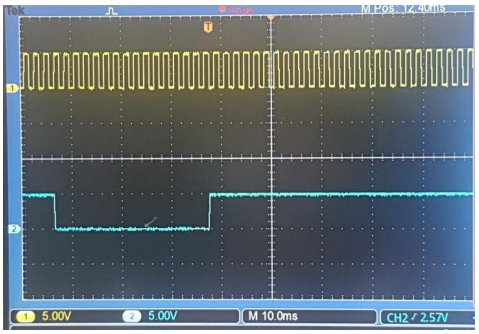

Fig. 13. Invalid Waveform Of DHT11

그림 13은 이 문제가 발생할 경우의 파형을 나타내며, 주기적 태스크 전환을 수행 중이고(노란색 파형), 센서 태스크의 문제로 인하여 더 이상 센서 파싱이 불가능하게 된다. 확인 불가한 특정 주소에서 무한 루프 상태인 것으로 판단되므로, 이 문제를 해결하기 위해서 AVR128 데이터시트의 인터럽트 벡터 테이블을 확인한다. 타이머 및 기타 페리페럴 장치의 인터럽트 우선순위를 확인 후 실시간 처리를 위한 함수 개발을 진행하였다. AVR의 경우 인터럽트가 발생하면 데이터시트 상의 인터럽트 벡터의 순서로 폴링을 한다. AVR은 인터럽트가 발생하면 순번이 낮은 벡터부터 수행한다. 또한 중첩인터럽트의 발생을 허용하기 위해서는 인터럽트내부벡터에서 인터럽트 허용 플래그를 사용함으로서 가능하다. 하지만 MCU가 중첩 벡터 지원이 가능할 지라도, 루틴 변경에 의한 부하가 동일하게 발생한다. 고성능 시스템의 경우와 달리, 8MHz 8bit 저성능 MUC에서는 멀티태스크 환경에서 실시간처리 문제가 상위와 같이 발생할 수 있다[6-7].

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	$0014	TIMER2 OVF	Timer/Counter2 Overflow
12	$0016	TIMER1 CAPT	Timer/Counter1 Capture Event
13	$0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	$001A	TIMER1 COMPB	Timer/Counter1 Compare Match B

Fig. 14. Reset And Interrupt Vectors Table

그림 14와 같이 프로세스 스케줄링 시에 필요한 타이머 인터럽트의 폴링 순서가 13번임을 확인하고 13번의 상위 순위에 있는 인터럽트를 사용 시에는 서비스 순서를 고려해야 한다. 인터럽트에 의해서 실시간성이 침해되므로 우선순위가 높은 서비스가 폴링 순서에 의해서 처리가 지연될 경우, 시간 오차가 발생할 수 있다[1]. TIMER2로 치환하여 TIMER1대신에 사용할 수 있지만, TIMER2는 8bit의 카운터 레지스터 때문에 주파수 설정이 어렵고 표현에 한계가 있다. 이를 해결하기 위하여 인터럽트 플래그를 통하여 폴링 순서에서 제외하는 방법이 있다. 또한 FREE RTOS 커널 튜닝을 수행하여 시간의 오차를 막는 방법이 있다. 정확한 시간에 처리하기 위해서는 다른 서비스를 지연시키는 방법이 있을 수 있는데, AVR은 중첩 인터럽트가 발생하지 않기 때문에 벡터 테이블을 확인 후 기아 혹은 인터럽트의 불일치 등을 고려하고 프로그램을 개발해야 한다. 이 방법은 경성 처리로 완료할 수 있는 작업은 정확한 시간에 종료할 수 있다.

그 결과를 실험을 통하여 도출하였고 먼저 커널에 삽입하는 코드부터 살펴본다. 사용자 정의 함수로서 경성 실시간 지원을 위한 셋 함수인 void set_real_time()를 FreeRTOS.h 헤더 파일에 선언하고, 경성 시스템을 다시 연성 시스템으로 변환하는 void release_real_time() 함수를 선언하였다. 본 함수는 port.c 파일에서 구현하였고 port.c의 소스에서 타이머 인터럽트 및 스케줄링 함수 루틴이 실행되기 때문에 실시간처리 함수도 port.c에 구현하였다. 함수 구현 부는 데이터 시트를 근간으로 개발하였으며, 주요 부분은 타이머에 관한 레지스터이다.

본 논문에서는 TIMER1부분을 사용하므로 AVR 데이터 시트의 TIMER/COUNTER INTTERRUT MASK REGISTER (TIMSK)를 확인한다[8-10].

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 15. Timsk Register

그림 15번은 타이머 제어를 위한 레지스터이다. TIMSK에서 타이머1과 관련 있는 비트는 OCIE1A, TOIE1 TICIE1이고, OCIE1A는 Output Compare Match Interrupt Enable 필드이고, TOIE1은 Overflow Interrupt Enable 필드이다. TICIE1는 Input Capture Interrupt Enable 필드이다. 실시간 경성 시스템을 구현하기 위하여 시간 정확성이 요구되는 태스크는 스케줄링의 발생을 억제하고, 스케줄링에 의한 컨텍스트 스위칭 또한 MCU 클럭을 소모하여 시간을 사용하기 때문에 실시간 처리에 방해요소가 된다. Set_real_time() 함수 구현부에는 타이머1과 관련된 인터럽트가 발생하지 않도록 구현해야 한다. Release_real_time() 함수에서는 다시 연성 시스템으로 변환 가능하게 하였다[11].

```
void Set_real_time()
{
    TIMSK = TIMSK & ~(1 << OCIE1A);
}
void Release_real_time()
{
    TIMSK |= (1 << OCIE1A);
}
```

Fig. 16. Set/release Real_time Function

Set_real_time 함수는 TIMSK의 기존에 설정된 비트를 변경하지 않고 TIMSK의 OCIE1A의 비트만 0으로 설정함으로써 타이머1의 인터럽트만 중지하고 다른 비트의 수정 없이 구현하였다. Release_real_time() 함수도 역시 TIMER1의 OCIE1A의 필드만 1로 설정하고 다시 스케줄링이 실행될 수 있도록 구현하였다.

```
while(1)
{
    Set_real_time();
    Request();
    Response();
    I_RH=Receive_data();
    D_RH=Receive_data();
    Release_real_time();
}
```

Fig. 17. Use Set/release Function In Main Thread

그림 17과 같이 태스크 함수 중 Set_real_time의 사용으로 타이머1 인터럽트의 스케줄링을 중지하고, Release_real_time함수를 이용하여 다시 연성 시스템으로 복귀하도록 코드를 적용하였다. 보드에 업로드 후의 파형의 결과는 그림 18와 같이 일정한 파형과 정확한 데이터가 발생하는 것을 확인할 수 있었다.

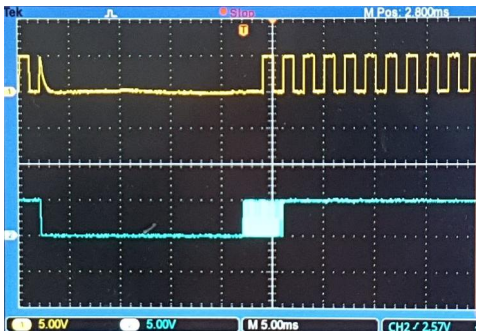

Fig. 18. Test Application Result

그림 18과 같이 노란색 파형은 스케줄링 발생 시점의 아웃풋 모드로 설정하였고 파란색은 실제 정확한 시간을 요하는 센서의 파형이다. 동작이 진행되고 완료와 동시에 다시 연성 시스템으로 복귀하는 것을 확인하였다[12].

IV. Conclusions

RTOS를 포팅 하더라도 경성 시스템화하지 않으며, 태스크 간 우선순위를 높인다고 하더라도 타이머 인터럽트의 스케줄링을 회피할 수 없다. 따라서 본 논문에서는 FREE RTOS 커널에 인터럽트 제어 함수를 구현함으로써 실시간 시스템에 필요한 태스크가 정확한 시간에 시작하여 종료하는 것을 그림 18를 통하여 확인하였다. 또한 저 클럭 MCU에 적용 가능한 경성 시스템 연구 및 한계점 해결을 위한 접근 방법을 제시하였다. 현재 대다수의 IoT 장비의 MCU가 고성능인 32bit로 개발되지만 여전히 비용 문제가 있으므로, 저 클럭 8bit MCU에서도 실시간시스템 처리 방법과 다양한 병렬 장치를 추가하여 실시간 태스크 처리 연구를 지속적으로 진행할 계획이다.

ACKNOWLEDGEMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT)

(No.2018-0-00389,R2R printed NFC active QR code-label for checking history of agro-fishery products to prevent forged record using a smart phone.)

REFERENCES

[1] *Atmel Avr128 DataSheet, https://www.microchip.com/wwwproducts/en/ATMEGA128*

[2] *FreeRtos Reference Manual, https://www.freertos.org/Documentation/RTOS_book.html.*

[3] *Mastering the FreeRTOS™ Real Time Kernel, https://www.freertos.org/Documentation/RTOS_book.html.*

[4] Seungjae Yu, Gu-in Kwon, “Smart Factory data collection using IoT device and Non-licensed frequency band wireless network”, KCI, Vol.25 No.2, 2017.

[5] Jae-jun Oh, Seong-ju Choi, Jin-Sa Kim, “Development of Multiple Wireless Communication Controller for Smart Factory Construction”, KIEEME, Vol.30, No.9, September 2017.

[6] Ki-Su Kim, “Atmega128A RTOS PORTING”, KSCI, Vol.25 No.2 ,pp. 266-269, 2017, KOREA, 2017.

[7] Bryant, Randal E./O'Hallaron, David R , “Computer Systems” Paperback, pp. 30-35, 2016.

[8] Su-Lim Tan ,Tran Nguyen Bao Anh, “Real-time operating system (RTOS) for small (16-bit) microcontroller” IEEE 13th International Symposium on Consumer Electronics, pp. 25-28, May 2009.

[9] Jean J Labrosse, “*uC/OS-II The Real-Time Kernel*”, CMP Books, pp. 80-120, 2009.

[10] K.Sakamura, H.Takada, “*μITRON for small scale embedded systems*”, IEEE Micro, vol. 15, pp. 46-54, December 1995.

[11] Kwai Hidemi, “*OS structure and principle*”,hanbit media, pp. 75-130, 2007.

[12] Silverschatz ,Galvin ,"*Operating System Conceps 7th Edition*", JOHN WILEY & SONS. INC, pp. 50-59, 2004.

Authors

Ki-Su Kim received the B.S., M.S. degrees in Computer Information Engineering from Kunsan University, Korea, in 2016, 2017, respectively. Ms. Kim is currently a Ph.d Course in the Department of Computer Information Engineering, Kunsan University. He is interested in Embedded Systems, PCB Design and Machine Learning, and Control System.

Jong-Chan Lee received the M.S. and Ph.D. degrees in computer science and engineering from Soongsil University, Korea, in 1996 and 2000 respectively. Dr. Lee was a senior member of engineering staff in Mobile Telecommunication Research Laboratory, Electronics and Telecommunications Research Institute (ETRI) from 2000 to 2005. Since 2005, he has worked in the Department of Computer Information Engineering, Kunsan National University as an professor. His current research interests are in the areas of dada analysis and deep learning.