논문 2008-5-20

PICmicro controller 교육을 위한 RTOS 적용 연구

Implementation of a Real Time OS for the Education of PICmicro Controller

이영대[*], 문찬우[**]

Young-Dae Lee, Chanwoo Moon

요 약 자동차, 로봇 등의 분야에서 실시간 운영체제의 사용이 증가하고 있다. PICmicro의 마이크로컨트롤러는 산업현장에서 많이 사용되고 있는 마이크로컨트롤러로서, 이 제품군에 실시간 운영체제(RTOS)를 탑재하려는 연구가 폭넓게 이루어지고 있으나 일부 제품군에서는 리소스의 제한으로 RTOS의 탑재가 어려운 경우가 많다. 이 연구에서는 교육적 목적으로 리소스가 제한된 PIC16F87x 시리즈에 대해 RTOS의 기능을 구현한다.

Abstract The purpose of the project was intended to show an application method of RTOS to PICmicro with limited resources with several tasks controlling the peripheral devices. The application runs on the designed PIC16F87x evaluation board with a bootloader burned so the application program can be easily downloaded using the serial communication without using the ROM writer. Thus, it would also be a good example to use for instructional or tutorial purposes for PICmicro education. The demo shows a useful examples who wish to use the real time operation system in their own projects.

Key Words : PICmicro, operating system, bootloader

Ⅰ. Introduction

Micro-controllers are highly integrated computer systems on a single low cost commodity chip containing a processor, program memory, RAM, discrete I/O, A/D converters, serial ports, and other function supports. They offer great potential for reducing the cost and increasing the performance of car control, robot control and etc. Real time system and embedded controller are closely related even for the simplest micro controller systems.

The preemptive style RTOS is commonly used with large embedded systems projects, and there are many commercial vendors of such operating systems[1]. They switch tasks in response to interrupts that make a higher priority task ready to run (or, rather, ready to resume). These general style OS switches the context for each task, and the stack space of each task must be large enough to accommodate the context switching, savings of return addresses and local variables, plus a little bit for interrupt service routines.

However, we cannot apply the above approach to simple microcomputers such as PICmicro with limited resources. PICmicro's micro controllers are widely used in industry, and PICmicro has several product families, PIC10, PIC12, PIC14, PIC16, and PIC18.[2] PIC18 family is relatively high performance micro controller with sufficient resources, and some researches that implement an open source code RTOS

[*]종신회원, 세명대학교 정보통신학과
[**]정회원, 국민대학교 전기전자공학부(교신저자)
접수일자 2008.8.23, 수정완료 2008.9.11

have been reported[3][4]. [3] and [4] are implemented based on the OSEK/VDX standard[5,6]. On the other hand, lower families have less resource, and it is hard to port a RTOS.

The purpose of this paper is to present a small sized RTOS to a PICmicro controller that has limited resources with only a few hundred bytes of RAM and a few kilo bytes of FLASH ROM such as PIC16F87x which are commonly used for the microprocessor education in many universities and companies. A bootloader helps to download the program into a micro controller from a PC through a serial port if it is pre-burned on the ROM space. The PIC16F87x have the on-board FLASH memory which can implement a bootloader. Using a bootloader, no burner is needed to reprogram the PICmicro, and it is not needed to remove it from the circuit.

This work reports a way to design of a demo board to implement of a RTOS based application programming for PIC16F87x micro controller the aid of a bootloader for the easy and efficient PICmicro education purpose.

II. A RTOS for PICmiro

1. Structure of a typical RTOS, OSEK case[3]

OSEK/VDX is an industry standard of RTOS for vehicles electronics, and has following features.

Task management : Two different types of task are provided, basic task and extended task. A basic task is a sequential code which can not be blocked by service call. On the other hand, an extended task has waiting state. Figure 1 shows the possible task status.

Scheduling policy : All tasks has a fixed priority, and the scheduling policy can be one of full non preemptive, full preemptive or mixed preemptive.

Conformance classes : The conformance classes define versions of the real-time executive, and they are defined by the type of task, the possibility of recording multiple activation request for a basic task, and the number of tasks per priority level.

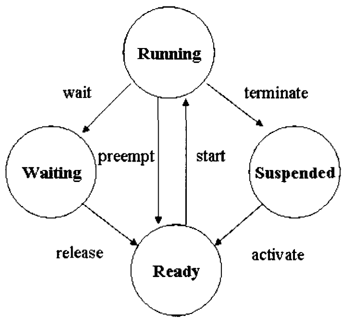


Fig. 1. Possible task states

Task synchronization : Only the owner task can explicitly wait for the occurrence of one or more of its events. The setting of occurrences can be made by tasks or ISRs

Resources management : The concurrent assess to shared resources is managed by OSEK-PCP, and the resource sharing is allowed between tasks and ISRs or between ISRs.

Alarms and counters : A counter counts ticks of a source, and an alarm links a counter and a task. OS allows the management of periodic tasks and watchdog timers for the monitoring of various situations.

Communication : Communication is based on message objects. Two types of message objects are provided, unqueued message and queued message.

2. Structure of the newly implemented RTOS

In our RTOS for the PICmicro, the hard real time task of interrupt level has the priority, and the other soft real time tasks are performed in a cooperative way without priorities. The cooperative and soft real time tasks is executed using round robin scheduler which is implemented by the 1ms timer interrupt. After the OS is initialized, the tasks are created, and they start or stop. Except the idle state that there is no task, the real time tasks(hard and soft) processes events.

The waiting OS can suspend the calling task for the specified time. The current task that will be set ready to run by interrupt or another task, and it can be made not ready state to running state, and its state can be determinated.

The message is generated in a task and it can be delivered to other task, and can be get from another task. A task acquires a semaphore or suspends until it is available. Also it releases a semaphore, if another task is waiting for the semaphore it becomes ready to run. The critical section and its identification processing routine is also implemented which are needed for such as EEPROM read/writing tasks.

Since the hard real time task requires an accuracy, the time amount should be minimized so it does not affect the other soft real time tasks. If a hard real time task needs a long time, then an event based method or the conversion to soft real time tasks are desirable. The priority between the soft real time tasks does not exist and the task switching occurs when a task requests an operating system service. Therefore, it is needed to insure that the task requests are made often enough to insure that the system realtime response requirements are met. Figure 2 shows flow of a typical user program. After initialization of special function registers and global variables, tasks and events are created. After that, the scheduler switches the tasks with round robin rules. Figure 3 is a typical timing chart of tasks execution time. Supposed that tasks A,B,C,D and E was created. Each task is executed sequentially with round robin scheduling. When task C is being executed, an interrupt happens, which has a higher priority, and task C is suspended.

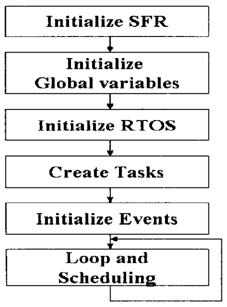


Fig. 2. Typical user program flow

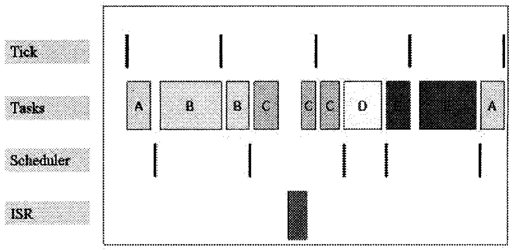


Fig. 3. A timing chart of tasks scheduling

After interrupt service routine is completed, task C is resumed. Task D is a short task which is less than one tick, and after task D is completed, next task E becomes running state by scheduler. In this case, Mean Latency Time of a task is calculated as equation 1[7].

$$MLT = \frac{1}{2n}\sum_{i=1}^{n} T_i \qquad (1)$$

Where n is number of tasks, and T_i is the execution time.

III. Implementation

1. Bootloader

All PICmicro programmers work the same way (except the bootloader), they generate a serial data stream using two signal lines, clock and data. Other pin controls the programming voltage (at MCLR), and two other pins supply power and ground. Another program running on the PC (the programming software) takes the hex file generated from the compiler translating it into a serial data stream. This is routed to a programmer through the correct interface (Serial, Parallel or USB). Figure 4 shows a commercial PICmicro programming products.

A bootloader is a program which stays in the micro controller and communicates with the PC usually through the serial UART interface. The bootloader receives a user program from the PC and writes it into the flash memory, then launches this program in execution.

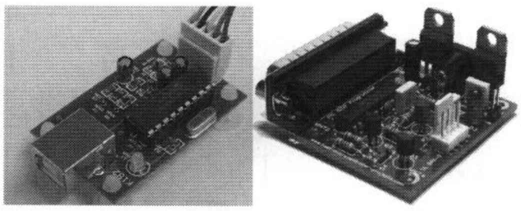


Fig. 4. Commercial PICmicro programming product examples using USB(left) and parallel port(right)

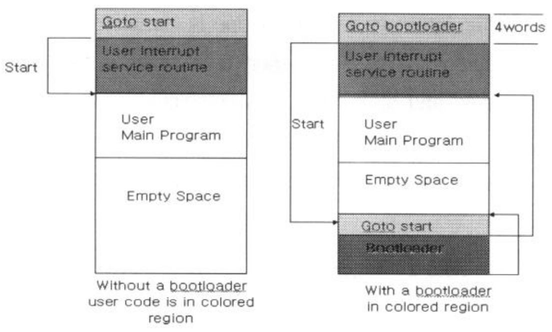

Goto start
User Interrupt service routine
Start
User Main Program
Empty Space
Without a bootloader user code is in colored region
Goto bootloader
4words
User Interrupt service routine
Start
User Main Program
Empty Space
Goto start
Bootloader
With a bootloader in colored region

Fig. 5. Operation of a bootloader for launching the program

Using a bootloader on the PIC, and a bootloader utility on the PC, the PICmicro can be reprogrammed in seconds over a serial link. Figure 5 shows the operations of a bootloader for launching a program.

2. Implementation of OS

For the main board design, we used PIC16F876A(it is compatible with PIC16F876 processor) with 20Mhz clock. Since RTOS can run in a micro controller with 128bytes data memory ram, PIC16F87x series have sufficient ram to implement it. From the table.1 which lists PIC16F87x series, PIC16F876/77 micro controllers are recommendable since they have 368bytes data ram and double flash memory which is larger comparing with PIC16F873/874 to store the program(application program + RTOS + bootloader).

Figure 6 shows the circuitries of the main board, power and RS232, respectively. We burned a recent bootloader into the chip which have small 255 byte instructions. The PIC16F87x serial bootloader has many variants and have been used successfully by thousands of users of worldwide.

Table 1. Comparison of PIC16F87x series

Features	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operation Speed (MAX)	20MHz	20MHz	20MHz	20MHz
FLASH Program Memory (14-bit word)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Port A,B,C	Port A,B,C, D,E	Port A,B,C	Port A,B,C ,D,E
Timer	3	3	3	3
Capture/Compare /PWM	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	-	PSP	-	PSP
10bit A/D module	5	8	5	8

- Burn the bootloader code in flash memory
- Check the application and OS size not to overwrite the bootloader area of flash memory in advance.
- Connect the PIC circuit to the PC via a serial link. and run the bootloader code from the PC and download your code to the circuit in seconds.

Figure 7 shows the parts of bootloader source program, and figure 8 is a downloader examples in PC. For the demo program, we tested the following tasks.

- Blink two LEDs alternatively at 1Hz rate
- Check a button on/off and display on another LED
- Check another button on/off and display on 7SEG LED
- Display time on the specified position of terminal
- Measure voltage in a variable resister and display the results on terminal

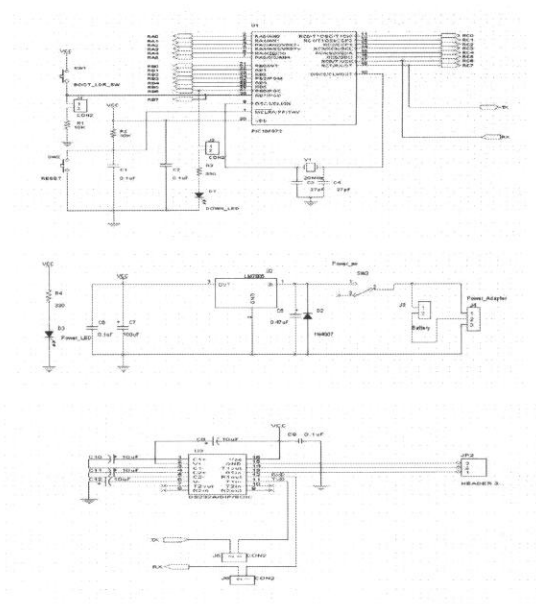
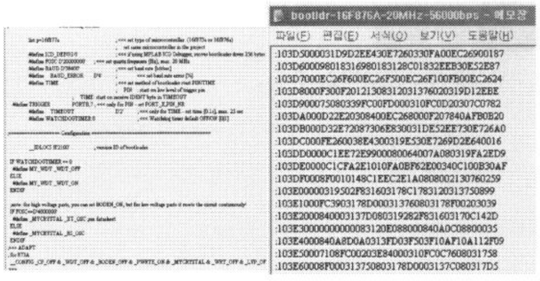


Fig. 6. The designed circuitry of main board part, power part and RS232 part, respectively



```
list p=16f877a                  ; <<< set type of microcontroller (16f873a or 16f876a)
                                ;     set same microcontroller in the project
#define ICD_DEBUG 0             ; <<< if using MPLAB ICD Debugger, move bootloader down 256 bytes
#define FOSC D'20000000'  ; <<< set quartz frequency [Hz], max. 20 MHz
#define BAUD D'56000'           ; <<< set baud rate [bit/sec]
#define  BAUD_ERROR    D'4'     ;          <<< set baud rate error [%]
#define TIME                    ; <<< set method of bootloader start PIN/TIME
                                ;   PIN  : start on low level of trigger pin
                       ;   TIME: start on receive IDENT byte in TIMEOUT
#define TRIGGER          PORTB,7 ; <<< only for PIN - set PORT,X,PIN_NR
#define   TIMEOUT                D'2'      ; <<< only for TIME - set time [0.1s], max. 25 sec
#define WATCHDOGTIMER 0                    ; <<< Watchdog timer default OFF/ON [0/1]

;------------------ Configuration ------------------------------------------

    __IDLOCS H'2100'          ; version ID of bootloader

IF WATCHDOGTIMER == 0
 #define MY_WDT _WDT_OFF
ELSE
 #define MY_WDT _WDT_ON
ENDIF

;note: for high voltage parts, you can set BODEN_ON, but for low voltage parts it resets the circuit continuously!
IF FOSC==D'4000000'
 #define _MYCRYSTAL _XT_OSC ;see datasheet
ELSE
 #define _MYCRYSTAL _HS_OSC
ENDIF
;<<< ADAPT
;for 873A
 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _MYCRYSTAL & _WRT_OFF & _LVP_OF
```

bootldr-16F876A-20MHz-56000bps - 메모장

```
:103D5000031D9D2EE430E7260330FA00EC26900187
:103D6000980183169801831 28C01832EEB30E52E87
:103D7000EC26F600EC26F500EC26F100FB00EC2624
:103D8000F300F2012130831203137602031 9D12EBE
:103D900075080339FC00FD000310FC0D20307C0782
:103DA000D22E20308400EC268000F207840AFB0B20
:103DB000D32E72087306E830031DE52EE730E726A0
:103DC000FE260038E4300319E530E7269D2E640016
:103DD0000C1EE72E9900080064007A080319FA2ED9
:103DE0000C1CFA2E1010FA0BF62E00340C100B30AF
:103DF0008F0010148C1EEC2E1A080800213076 0259
:103E00000319502F8316031 78C1783120313750899
:103E1000FC39031 78D0003137608031 78F00203039
:103E200084000313 7D080319282F8316031 70C142D
:103E300000000000083120E088000840A0C08800035
:103E4000840A8D0A0313FD03F503F10AF10A112F09
:103E50007108FC00203E84000310FC0C7608031758
:103E60008F0003137508031 78D0003137C08031 7D5
```

Fig. 7. Parts of source program source of a bootloader and its hex file, respectively.

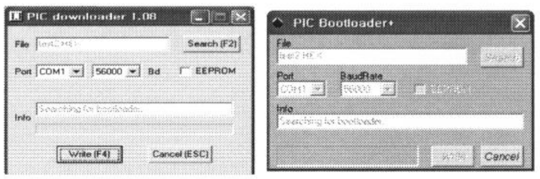


Fig.8. Downloader examples in PC

The demonstration code was written to test the designed PIC16F876A board as shown in Figure 9. In order to clearly present the hardware actually used in the demo, the below schematic of figure 10 shows the part of the designed PIC16F876A demo board that are needed for the demo to run.

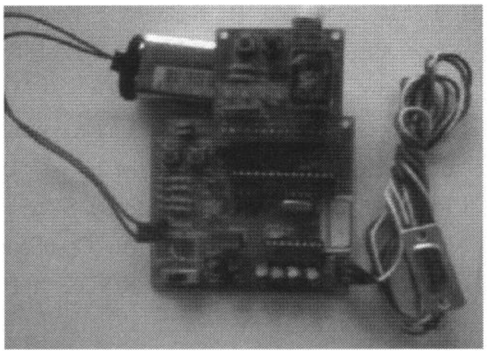


Fig. 9. The designed PICmicro demo board

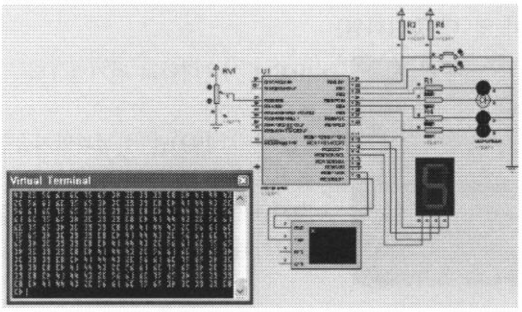


Fig. 10. Demo execution using the simulator

IV. Conclusion

A cooperative real-time operating system for micro controllers can run in a small size micro controller with a few hundreds of bytes RAM and kilo bytes of flash ROM.

The project of this paper is to implement and evaluate the performance of a cooperative RTOS in PICmicro controllers which have limited resources.

A demo board was designed to implement a minimal size bootloader which can download the executable program of RTOS, and its application program through a PC serial port using the burned bootloader. The demo code was written in C language using CCS-C compiler with IDE(Integrated Development Environment) and was successfully tested on the designed PIC16F876A board.

Reference

[1] J. Stankovic and R. Rajkumar. "Real-Time Operating Systems". The Journal of Real-Time Systems, 28(2/3): pp. 237-253, 2004.

[2] Y.D. Lee et al., "The recent PIC microcontroller design", C&C INst. Co. LTD., the 1st ed., 2008.

[3] Jean-Luc B´echennec, Mika¨el Briday, S´ebastien Faucou and Yvon Trinquet, "An OpenSource Implementation of the OSEK/VDX RTOS Specification"

[4] Pragmatec, "Real time kernel for PIC18, PICos18", 2006

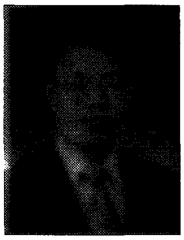
[5] Website, www.osek-vdx.org

[6] K. Zuberi et al. "EMERALDS-OSEK: A Small Real-Time Operating System for Automotive Control and Monitoring", In Society of Automotive Engineers Congress and Exposition, 1999.

[7] Francesco "Conversi, SISTEMA OPERATIVO PicOs", 2005

저자 소개

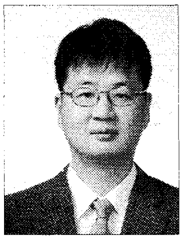
이 영 대(종신회원)



• 1998년 서울대학교 전기컴퓨터 공학부 박사
• 1998년 ~ 1999년 한국과학기술원 휴먼로봇센터
• 1999년 ~ 현재 세명대학교 정보통신공학과

<주관심분야 : 임베디드 시스템, 로보틱스, 이동통신 전력 제어>

문 찬 우(정회원)



• 2001년 서울대학교 전기컴퓨터 공학부 박사
• 2002년 ~ 2006년 전자부품연구원 선임,책임 연구원
• 2006년 ~ 현재 국민대학교 전자공학부 교수

<주관심분야 : 로보틱스 응용>