범용 실시간 O.S.와 프로세스 보드의 인터페이스 설계

(Design of the Interface between General Purpose Realtime OS and CPU Board)

전형구*김진수*김남수**조병진***
(H.G.Jeon, J.S.Kim, N.S.Kim, B.J.Jho)

복잡하고 다양한 기능을 갖는 컨트롤 시스템은 멀티태스킹 구조를 요구한다. 또한 멀티태스킹 구조에서는 태스크 사이의 통신, 스케줄링 기능, 메모리 관리 기능등을 갖춘 실시간 O.S.를 필요로 한다. 타깃 컨트롤 보드에서 범용 실시간 O.S.를 수행시키려면 H/W 인터페이스용 BSP(Board Support Package) 프로그램이 요구된다. 본 고에서는 범용 실시간 O.S.와 프로세스 보드와의 인터페이스(BSP) 프로그램을 설계하였고 간단한 응용 프로그램으로 BSP 기능을 확인하였다.

I. 서 론

CDMA 방식 이동통신 기지국 내부에서 채널카드는 채널 엘리먼트와 채널 카드 컨트롤러의 두 개의 모듈로 구성된다. 채널 엘리먼트는 Intel i960 프로세서와 CDMA용 ASIC chip을 사용하여 구성되며 CDMA 변복조 및 code, decode를 담당하는 중요한 역할을 맡고 있다. 채널 카드 컨트롤러는 Intel 80C186 프로세서를 사용하여 2개의 채널 엘리먼트를 컨트롤하며 셀 컨트롤러와 HDLC 방식으로 제어 정보를 주고 받는다. 채널 카드 컨트롤러는 채널 엘리먼트를 감시제어하고 셀 컨트롤러와 제어정보를 교환하는 기능을 멀티태스킹으로 구현하고 있다.

멀티태스킹에서는 태스크 사이의 통신에 필요한 큐나 메일박스가 필요하며, resource 사이의 데드록(dead lock) 방지, 태스킹 사이의 프로세스 스케줄링을 해주는 실시간 O.S.가 요구된다.

채널 카드 컨트롤러는 멀티태스킹 프로그램 수행을 위해서 범용 O.S. 인 VRTX32를 사용하였다. VRTX32는 Ready System사에서 상용 O.S.로 개발한 S/W 컴포넌트이며 타깃 보드 메모리 영역의 어떤 위치에도 존재할 수 있게 되어 있다.

*무선기술연구실 연구원
**무선기술연구실 선임연구원
***무선기술연구실 실장

VRTX32는 Configuration 테이블을 통하여 미리 개발 환경에 대한 정보를 받기 때문에 외부 환경에 별 영향 받지 않도록 설계되어 있다. 또한 사용자는 이 configuration 테이블과 BSP를 작성하여 VRTX32와 H/W를 인터페이스 시켜주어야 한다. BSP 프로그램은 보드 초기화 및 인터럽트 서비스 루틴으로 구성된 어셈블리 프로그램이다.

본 고에서는 실시간 O.S.로 사용되는 VRTX32를 채널 카트 컨트롤러에서 수행시킬 수 있도록 하는 인터페이스(BSP) 설계 및 configuration 테이블 작성에 대해서 기술하였다. Ⅱ장에서는 VRTX32의 개요, 초기화 및 configuration 테이블의 파라미터 값 결정에 대해서 설명하고 Ⅲ장에서는 타깃 보드의 구조에 대해서 기술하고 그 프로세스 보드에 맞는 인터페이스(BSP) 프로그램 설계 절차를 제시하였다. Ⅳ장에서는 설계한 BSP 프로그램을 테스트하기 위한 응용 프로그램을 설계하고 타깃보드에서 실행시켜 그 결과를 확인하였으며 Ⅴ장에서 결론을 맺었다.

Ⅱ. 범용 실시간 O.S. VRTX32의 특성

본 장에서는 타깃 프로세스 보드에 이식할 실시간 O.S.인 VRTX32의 특징, 구조 및 초기화에 대해서 살펴보도록 한다.

1. VRTX32 개요

VRTX32는 86계열(80286, 80386은 리얼모드에서 지원) 및 68계열을 지원하는 멀티태스킹 용 실시간 O.S.이며, 하나의 독립된 프로그램 컴포넌트로 구성되어 사용자의 보드 특성이나 외부환경에 유연하게 적응할 수 있도록 설계되어 있다. VRTX32의 커널 사이즈는 약 8Kbyte 정도이며 상대적으로 빠른 속도를 가지고 있다.

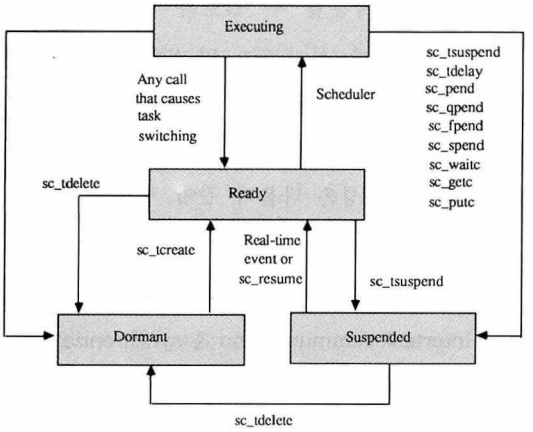
VRTX32는 실시간 O.S.가 요구하는 모든 특성을 제공하며 그 특성은 다음과 같이 정리할 수 있다.

- Multitasking support
- Event-driven, priority-based scheduling
- Intertask communication & synchronization
- Dynamic memory allocation
- Real time clock control, with optional time slicing
- Character I/O support
- Real-time reponsiveness

VRTX32는 멀티태스킹 관리, 메모리 할당, 통신 및 동기화, 캐릭터 I/O에 관련하여 다양한 시스팀 콜을 제공하며 사용자는 시스팀 콜을 액세스하여 응용 프로그램에서 필요한 기능을 구현할 수 있다.

태스크는 사용자의 태스크 생성 시스팀 콜에 의해서 생성되며 그 태스크는 255까지 번호 가운데 자기 ID 번호와 우선순위를 가질 수 있다. 멀티태스킹 환경에서 각 태스크는 4가지 상태(Executing, Ready, Dormant, Suspended) 중에서 반드시 어느 한 상태에 머물게 된다.

VRTX32 스케줄러는 태스크 수행중에 시스팀 콜이 발생하면 리스케줄링을 한다. 현재 실행중인 태스크가 suspended 되었다면 ready 상태에 있는 다음 우선순위 태스크로 태스크 천이(task switch)가 발생된다. 일단 Suspended된 태스크가 다시 ready 상태가 되면 현재 실행중인 태스크는 중지되고 preemption이 발생된다. (그림 1)은 태스크 상태와 태스크 천이를 나타낸다.



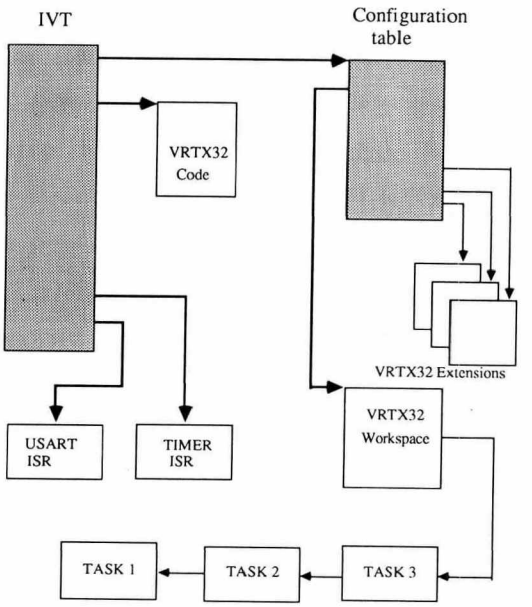
(그림 1) 태스크 상태와 태스크 천이

사용자는 VRTX32, VRTX 32 configuration 테이블 및 응용 프로그램을 연결하기 위해서 두개의 포인터를 인터럽트 벡터 테이블에 초기화해야 한다. 이 포인터는 configuration 테이블 포인터와 VRTX32 entry 포인터이다.

Configuration 테이블은 VRTX32가 요구하는 외부환경에 대한 모든 파라미터를 정의하고 있으며 초기화 과정에서 어드레스 200H(인터럽트 벡터 80H)에 이 테이블 포인터가 저장된다. VRTX 32 entry 포인터는 VRTX32가 시작하는 시작점이며 초기화 과정에서 어드레스 204H (인터럽트 벡터 81H)에 저장된다. 이와 같이 인터럽트 벡터 테이블을 사용하여 포인터를 지정 하므로 VRTX 32는 메모리의 어느 영역에도 위치할 수 있다.

(그림 2)는 인터럽트 벡터 테이블과 VRTX32 코드 및 configuration 테이블의 링크 구조를 나타낸다.

VRTX32를 타깃 보드에서 실행시키기 위해서는 BSP가 필요하다. BSP 프로그램은 VRTX32와 H/W를 연결시켜주는 H/W에 관련된 프로그램이



(그림 2) VRTX32 코드 및 configuration 테이블의 링크

다. (그림 3)은 시스템의 계층구조를 나타낸다.

BSP는 GPC86. INC, BOARD.ASM, TIMER.ASM, TTY.ASM의 4개 파일로 작성되며 VRTX32가 하드웨어를 직접 구동시킬 수 있게 한다. BSP, VRTX32 및 응용 프로그램을 ROM화하여 수행시키려면 startup 프로그램을 같이 링크시켜야 한다. 다음 장에서는 startup 및 BSP 프로그램에 대해서 자세히 다루기로 한다.

Application program
VRTX32/OS
BSP program
H/W

(그림 3) 시스템 계층 구조

2. 시스템 configuration

사용자는 Configuration 테이블을 통하여 VRTX32

에게 설치되는 환경조건 즉 최대 태스크, 사용되는 큐 개수 등에 대해서 미리 알려주어야 한다. VRTX 32는 초기화 과정에서 이 테이블을 참조하여 RAM 영역에서 작업영역 등을 확보한다. Configuration 테이블의 시작 주소는 인터럽트 벡터(80(H))테이블에 기록되며 정의되는 변수는 〈표 1〉과 같다.

〈표 1〉 VRTX32 configuration table

데이터	항 목	비 고
dw	seg VRTX_wspace	VRTX32 작업영역의 번지
dw	VRTX_wsize	VRTX32 작업영역의 크기
dw	0	reserved, 0으로 정함
dw	IStk_size	ISR stack의 크기
dw	Max_CB	control block의 크기
dw	IDtk_size	Idle task의 크기
dw	0	reserved, 0으로 정함
dw	TStk_size	Task stack의 크기
dw	0, 0	reserved, 0으로 정함
dw	Max_tasks	최대 task 수
dw	0	reserved, 0으로 정함
dd	v_tx_driver	Character I/O drive routine번지
dd	TCREATE	Tcreate hook 주소
dd	TDELETE	Tdelete hook 주소
dd	TSWITCH	Tswitch hook 주소
dd	dsp_CVT	component 벡터 테이블의 주소

VRTX32는 인터럽트가 발생되면 인터럽트 핸들러가 사용할 스택은 현재 인터럽트 당한 태스크의 스택을 사용하도록 하였다. 따라서 각 태스크는 인터럽트들이 사용할지도 모르는 충분한 스택용량을 가지고 있어야 한다. 이 방식은 태스크마다 많은 메모리가 필요하므로 메모리 사이즈가 제한된 시스템에서 문제를 야기할 수 있다. VRTX32는 이러한 문제점에 대한 대책으로 인터럽트 스택 스위칭 기능을 제공하고 있다. 인터럽트 스택 스위칭은 인터럽트 핸들러들만이 사용할 수 있는 하나의 인터럽트 전용 스택을 사용하게 하는 것이다.

인터럽트 스택 스위칭 기능이 사용되지 않으면 IStk_size 값을 0으로 세트하며 인터럽트 스택 스위칭 기능이 사용되면 IStk_size값을 각 인터럽트 핸들러가 요구하는 스택용량 값으로 세트해야 한다.

Max_CB는 프로그램에서 한번에 존재할 수 있는 최대 event_flag와 세마포의 수를 나타낸다. Event_flag와 세마포를 사용하지 않는다면 0으로 세트한다.

IDtk_size는 인터럽트 스택 스위칭 기능이 정의되어 있으면 0으로 세트하여 default 값(128)을 갖게 하고, 인터럽트 스택 스위칭 기능이 사용되지 않으면 최악의 경우(nested interrupt 경우)를 고려하여 최대로 필요한 인터럽트 스택 용량 값으로 세트해야 한다.

TStk_size는 다음 공식을 이용하여 구할 수 있다.

TStk_size=64+UX+USC+ISW+ISR+C+TSR

여기서 64는 VRTX32가 기본으로 요구하는 스택의 byte 수이다.

- UX는 사용자가 VRTX32 extension을 만들어 VRTX32에 추가할 경우 사용되는 스택의 양을 나타낸다.
- USC는 사용자가 시스템 콜을 만들어 VRTX32에 추가할 경우 사용되는 스택의 양을 나타낸다.
- ISW는 인터럽트 스택 스위칭이 발생할 때 필요한 스택의 양이며 인터럽트 스택 스위칭이 사용되면 16이고 사용되지 않으면 0으로 세

트한다.

- ISR은 인터럽트 스택 스위칭 기능이 사용될 때 0으로 세트하며 인터럽트 스택 스위칭 기능이 사용되지 않을 때는 인터럽트가 필요로 하는 총 스택 용량을 나타낸다.
- C는 다른 S/W 컴포넌트(디버거, 파일 관리 S/W 등) 콜에 의해 요구되는 총 스택의 용량을 나타낸다.
- TSR은 서브루틴 콜에 의해 요구되는 스택의 용량을 나타낸다.

Max_tasks는 최대로 존재하는 태스크수를 나타낸다. TCREATE, TDELETE, TSWITCH는 tcreate, tdelete, tswitch 시스템 콜 수행시 자동으로 기동되는 사용자가 정의한 확장 루틴의 어드레스를 나타낸다.

VRTX_wsize는 다음 공식으로 구할 수 있다.

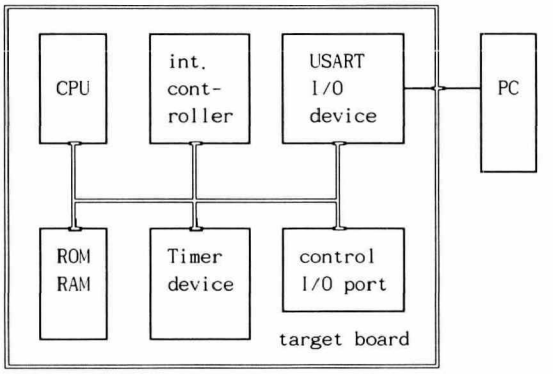
$$VRTX\_wsize = 2000 + (110 + US)T + 64P + 2B + 32E + 32Q + 4QE + 36Max\_CB + idle + istk$$

여기서 US는 TStk_size, T는 최대 태스크수, P는 최대 메모리 분할 수, B는 최대 메모리 블럭수, 또는 최대 메모리 분할 확장수, Q는 최대 큐의 수, QE는 최대 큐 엘리먼트 수, Max_CB는 event flag와 세마포의 최대 제어블럭수, idle은 아이들 태스크 스택 사이즈, istk는 인터럽트 스택의 크기를 나타낸다.

Ⅲ. 인터페이스(BSP)프로그램 설계

1. 타깃 보드의 구조

타깃 보드의 CPU는 80C186이며 구성도는 (그림 4)와 같다. 타깃 보드는 개발초기 디버깅을 위해서 PC와 통신할 필요가 있으므로 RS232C 시리얼 포트를 통해서 PC와 연결되어 있다. PC를 응용 프로그램 다운로드 용이나 단말기로 사용하며 타깃 보드에서 실행된 프로그램 결과를 모니터할 수 있도록 하였다.



(그림 4) 타깃 보드의 구조

2. BSP 프로그램 설계

가. STARTUP.ASM

80C186 CPU는 리셋 상태에서 프로그램 카운터가 'FFFF0'(H)을 가리키고 있다. 따라서 STARTUP.ASM 프로그램은 FFFF0 번지에서 다른 번지로 점프해서 시스템에 필요한 초기화 작업을 하는 bootstrap code를 포함하고 있으며, 응용 프로그램에서 사용하는 변수 초기값을 ROM 영역에서 RAM 영역으로 옮기는 작업을 하고 사용할 스택 영역을 확보하며, 메모리 모델(휴지, 스몰, 미디엄 모델 등)을 정의하고 마지막 단계에서 main() 프로그램을 콜한다. 본 고에서는 Microsoft C에서 제공한 샘플 start-up code를 약간 수정하였다. 샘플 startup code의 소스 코드는 참고문헌 [5]에서 구할 수 있다.

나. GPC86. INC

GPC86.INC는 시스템에서 필요한 레지스터 초기화 값 및 변수와 인터럽트 벡터에 필요한 값들을 모두 모아 정의한 파일이다. 여기서 포함되는 레지스터는 80C186 CPU 내부의 시스템 레지스터, 타이머 카운터 레지스터, DMA 컨트롤러 레지스터, 인터럽트 컨트롤러 레지스터와 그밖의 I/O 디바이스의 어드레스 값을 정의하고 있다.

다. BOARD.ASM

BOARD.ASM 프로그램은 프로세서 보드의 초기화 절차를 수행한다. 즉 인터럽트 컨트롤러나 주변 디바이스에 필요한 내부 레지스터 값을 적절한 값으로 세트한다.

프로그램은 초기화의 단일 모듈로 구성되며 'PUBPRO dsp_init_board'로 선언되어 응용 프로그램에서 이 모듈을 콜할 수 있도록 하였다.

```
Begin dsp_init_board
        save function code register
        initialize CPU internal controller
            - interrupt controller
            - other device
        restore function code register
        return
End  dsp_init_board
```

라. TIMER.ASM

TIMER.ASM 프로그램은 VRTX32에서 필요로 하는 클럭 카운트를 제공하는 프로그램이다. 이 프로그램은 dsp_timer_init, TIMER_isr의 2개 모듈로 구성된다.

dsp_timer_init는 타이머가 20ms마다 인터럽트를 발생할 수 있도록 내부 레지스터 값을 세트시키며 인터럽트 벡터 테이블에 TIMER_isr의 오프셋과 세트먼트 어드레스를 등록한다.

TIMER_isr은 인터럽트 서비스 루틴이며 인터럽트가 발생할 때마다 UI_TIMER 콜을 사용해서 VRTX32에게 클럭 tick을 알려주는 역할을 한다. 각 인터럽트 서비스 루틴은 시작과 끝에 UI-ENTER와 UI-EXIT 시스템 콜을 사용하여 VRTX32에게 현재 인터럽트 서비스 중이라는 사실을 알려주어 서비스 루틴 수행중에 이벤트가 발생하여도 태스크 스위칭이 일어나지 않도록 한다. TIMER.ASM은 다음과 같은 순서로 프로그램된다.

```
Begin dsp_timer_init
    save function code register
    initialize counter-timer
        - set timer mode
        - set maximum counter number
        - set current count value
    initialize vector in vector table
        - save TIMER_isr address in vector table
    restore function code register
    return
End  dsp_timer_init
Begin TIMER_isr
    save function code register
    call UI_ENTER
    save any other registers you use
    call UI_TIMER
    restore all register except function
    code register
    clear EOI(End of Interrupt) register
    call UI_EXIT
    return
End  TIMER_isr
```

마. TTY.ASM

TTY.ASM 프로그램은 VRTX32 character I/O를

지원하기 위한 프로그램으로서 dsp_init_tty, v_tx_driver, USART_ISR 등 3개의 모듈로 구성되어 있다.

dsp_init_tty 모듈은 character I/O 디바이스의 내부 레지스터를 필요한 값으로 세트시키며 인터럽트 벡터 테이블에 USART_ISR 루틴의 오프셋 및 세그먼트 어드레스를 등록시킨다. 캐릭터 I/O 디바이스는 9600 baud rate, 1 stop bit, no parity 모드로 세트한다.

v_tx_driver 루틴은 실제로 버퍼에 있는 캐릭터를 캐릭터 I/O 디바이스를 통해서 외부에 출력시킨다.

USART_ISR은 캐릭터를 캐릭터 I/O 디바이스에 송수신할 때 발생되는 인터럽트에 대한 인터럽트 서비스 루틴이다. USART_ISR은 캐릭터 수신 인터럽트가 발생하면 UI_RXCHR을 콜하여 VRTX32에 캐릭터가 수신되었음을 알린다. 캐릭터 송신시에는 UI_TXRDY를 콜하여 VRTX32로부터 송신할 캐릭터를 넘겨받고 v_tx_driver를 콜하여 그 캐릭터를 송신한다.

TTY.ASM은 다음과 같은 순서로 프로그램된다.

```
Begin dsp_init_tty
        save function code register
        initialize character I/O device
            - 1 stop bit
            - 9600 baud rate
            - no parity
        initialize vector in vector table
            - save USART_ISR address in vector table
        restore function code register
        return
End  dsp_init_tty

Begin v_tx_driver
        push the character in stack
        save function code register
        step 1 :
            check if USART is ready
            if not ready, then repeat step 1
        pop the character
        send the character
        restore function code register
        return
End  v_tx_driver

Begin USART_ISR
        save function code register
        call UI_ENTER
        save other registers used
    step 1 :
        check USART status
        receiver interrupt?
            yes, then: goto step 3
    step 2 :
        call UI_TXRDY
        if character present,
            then : call v_tx_driver
                   repeat step 2
        check USART status
        receiver interrupt?
            no, then : goto step 4
    step 3 :
        get character from USART
        call UI_RXCHR
        repeat step 1
    step 4 :
        clear EOI register
        restore registers
        restore function code register
        call UI_EXIT
        return
End  USART_ISR
```

Ⅳ. 프로그램 실행 test 및 결과

지금까지 설계한 BSP 프로그램을 토대로 간단한 응용 프로그램을 만들어 타깃 보드에서 수행

시켜 보도록 하자. 타깃 보드와 단말기는 RS-232C 시리얼 포트를 통하여 연결되어 있어서 프로그램 실행결과를 모니터할 수 있다. 테스트용 응용 프로그램은 VRTX32환경에서 2개 태스크로 구성하였다.

Main()은 처음에 타깃 보드와 VRTX32를 초기화하기 위하여 vrtx_init(), dsp_init_board(), dsp_init_tty(), dsp_timer_init() 함수를 차례로 콜한다. 그 다음 각 태스크에 대해서 고유번호를 지정하여 task1, task2를 생성하고 자신을 디스에이블 시킨다. 여기서 task1의 우선순위는 task2의 우선순위보다 높게 하였다.

각 태스크를 이루는 task1(), task2() function은 무한루프로 구성된다.

Task1()은 단말기로부터 'q' 캐릭터를 기다린다. 단말기를 통해서 'q'가 입력되면 global 변수 flag를 1로 세트해서 Task2에 알린다. Task2()는 global 변수 flag가 1이 되었는지 체크하고 1이 되면 단말기를 통해서 'q'가 입력되었다는 사실을 알린다. 시험 응용 프로그램은 다음과 같다.

```
#include "xcmpnent.h"
#include "vrtxil.h"

char q=0x71;
int flag;
void main()
{
  int err;

  dsp_init_board();
  dsp_init_tty();
  dsp_vrtx_init();
  dsp_timer_init();
  sc_tcreate (task1,1,1,1,&err); /* task create */
  sc_tcreate (task2,2,2,2,&err);
  /* start multi-tasking */
  sc_tdelete(0,0,&err); /* 자신을 delete 한다. */
}
void task1()
{
int err;
long m;
char g;
m=20;

for(;;)
    {
    flag=0;
    putline("task1 is running!!!\n\r");
    putline("task1 is waiting character 'q'!\n\r");
    g=sc_waitc(q,&err); /* wait a charater system call */
    sc_putc(g);
    flag=1;
    sc_delay(m); /* delay system call */
    }
}
void task2()
{
int err;
char *st;
for(;;)
    {
    if(flag==1){
      st="task2 is running!\n\r";
      putline(st);
      flag=0;
      }
    }
}
putline(disp)
char *disp;
{
  int i;
  for(i=0;disp[i]!=0;i++)
  sc_putc(disp[i]); /*one character output*/
}
```

(그림 5)는 프로그램 실행 결과를 보여준다.

```
task1 is running!!!
task1 is waiting character 'q'!
q
task2 is running!
task1 is running!!!
task1 is waiting character 'q'!
```

(그림 5) 프로그램 실행 결과

V. 결 론

프로그램 양이 많고 멀티태스킹으로 컨트롤 기능을 수행하는 프로세서 보드는 실시간 O.S.를 필요로 한다. 본 고에서는 실시간 O.S.로서 범용 O.S.인 VRTX32를 선택하였고 컨트롤 보드의 CPU는 80C186으로 구성하였다.

사용자는 범용 O.S.를 외부 H/W 환경과 일치시키기 위해서 configuration 테이블과 BSP 프로그램을 설계하여야 한다. VRTX32는 초기화 과정중에 configuration 테이블을 참조하여 작업영역을 확보한다. BSP 프로그램은 H/W와 VRTX32를 인터페이스 시켜주는 역할을 한다.

본 고에서는 configuration 테이블을 분석하여 파라미터를 결정하고 BSP 프로그램을 설계하였다. 설계된 BSP 프로그램 검증을 위하여 간단한 응용 프로그램을 수행시켜 보았으며 예측된 결과를 확인하였다.

본 고에서 제시한 BSP 프로그램 설계 절차는 86 계열을 지원하는 VRTX32에서 사용될 수 있을 것으로 생각된다.

참 고 문 헌

1. Ready System, VRTX32 Versatile Real-Time Executive for the IAPX86 Family User's Guide. California, May 1991, pp.1.1-4.14.
2. Ready System,VRTX32 C user's manual. California, Dec. 1988.
3. 조유근 외 1 공역, 운영체제론. 서울:홍릉과학 출판사, 1989, pp.53-68.
4. Ready System, How to Write a Board Support Package for VRTX. California, 1986, pp.1-15.
5. Wilson C. Chan, Writing ROMable Code in Microsoft C. System & Software Inc, Jan. 1990.
6. Intel, 16-/32-bit Embeded Processor. 1991, pp.1.59-1.127.
7. Intel, Embeded Application. 1991, pp.5.19-5.80.