【 기술 노트 13 】

# 뱅크 스위칭 기법을 이용한 메모리의 확장

마이크로프로세서에서 선형적으로 액세스할 수 있는 메모리 용량은 어드레스 버스의 신호선 수에 의하여 결정된다. 예를 들어서 8051이나 80C196KC처럼 어드레스 버스가 16비트인 마이크로콘트롤러는 64 KB의 메모리 용량을 가지며, 80286처럼 어드레스 버스가 24비트인 마이크로프로세서는 16 MB의 메모리 용량을 가진다.

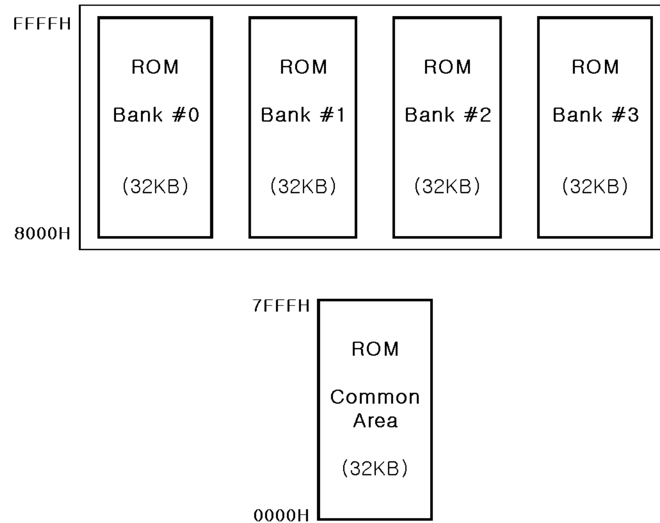
이렇게 선형 어드레스 영역이 부족하여 메모리 용량을 늘리고자 하는 경우에는 흔히 뱅크 스위칭(bank switching) 기법이 사용된다. 8051 마이크로콘트롤러에서처럼 메모리 영역을 프로그램 메모리와 데이터 메모리로 구분하는 경우에 특히 문제가 되는 것은 프로그램 메모리의 확장인데, 이러한 경우에 뱅크 스위칭 기법을 특별히 코드 뱅킹(code banking)이라고 부르기도 한다.

뱅크 스위칭 기법을 사용하려면 특별한 하드웨어를 필요로 하며, 여기에 적절한 소프트웨어의 지원이 따라야 한다. 뱅크 스위칭 기법을 사용하는 방법은 어느 마이크로프로세서에서나 매우 유사하다. 그러나, 여기서는 이 기능을 특별히 적극적으로 지원하는 Keil사의 소프트웨어 개발툴을 기본으로 하여 8051 시스템의 경우를 예로 들어 설명하기로 한다. 따라서, 8051을 사용하는 사용자는 이 방법을 그대로 적용하면 되겠지만, 다른 마이크로프로세서의 경우에도 이를 기본으로 하여 유사하게 설계하여 메모리를 확장하면 될 것이다.

## 1. 뱅크 스위칭의 개요

8051은 선형적으로 64 KB의 프로그램 메모리를 지원한다. 그러나, 뱅크 스위칭 기법을 사용하면 이 용량을 더 크게 확장할 수 있다. 뱅크 스위칭이란 쉽게 말하면 추가적인 하드웨어를 이용하여 하나의 메모리 영역에 여러개의 뱅크를 중첩되도록 위치시키는 방법이다. 이를 위하여 어드레스 버스의 상위 자리에 추가적으로 어드레스 버스의 역할을 수행하는 신호를 지정하여 사용한다.

예를 들어서 8051 시스템에서 0000H～7FFFH 영역의 32 KB를 기본 ROM으로 사용하고(이를 공통 영역 즉 common area라고 한다.), 8000H～FFFFH 영역의 32 KB에 4개의 32 KB ROM(이를 뱅크 메모리 즉 bank ROM이라고 한다.)을 중첩적으로 설치할 수 있다. 이 경우에 4개의 뱅크 ROM 중에서 어느 하나를 선택하기 위해서 2비트의 추가적인 어드레스 신호선이 필요하게 되는데, 이 신호선들은 일반적으로 8051의 병렬 I/O 포트의 2비트를 사용하거나 또는 XDATA 영역의 어느 한 번지에 있는 2비트를 사용한다. 이 2비트의 값이 00, 01, 10, 11로 동작함에 따라 각각 뱅크 0～3이 선택되는 것이다. 이를 알기 쉽게 도시하면 <그림 1>과 같다.




<그림 1> 뱅크 스위칭 기법의 개념도

## 2. BL51에서 뱅크 스위칭을 지원하는 기능

Keil사의 Code Banking Linker/Locator인 BL51.EXE은 이러한 뱅크 스위칭 기법을 적극적으로 지원하고 있다. 뱅크 스위칭 기법을 사용하기 위하여 사용자가 소스 프로그램에서 특별한 처리를 수행하는 것은 없다. 다만, 소스 프로그램을 컴파일하고 나서 BL51이 이를 링크할 때 각 프로그램 모듈들을 뱅크 스위칭에 적합하도록 배치하는 것이다. 이를 위하여 BL51에서는 BANKAREA, BANKx, COMMON 등의 지시어를 사용할 수 있으며, 각 뱅크를 스위칭 또는 선택하는데 필요한 명령 코드는 \C51\LIB 서브 디렉토리의 L51_BANK.A51 파일에 미리 준비되어 있어서 BL51이 이를 사용자 프로그램에 링크시키며, 사용자는 만약 필요하다면 이 파일을 수정하여 사용할 수도 있다. 여기서는 BL51에서 지원하는 뱅크 스위칭 관련 기능을 알아 보기로 한다.

(1) Common Code Area

BL51은 프로그램 코드를 1개의 공통 코드 영역(common code area)과 32개까지의 코드 뱅크에 나누어 배치(locate)시킬 수 있다. 공통 코드 영역은 항상 모든 뱅크에 의하여 액세스될 수 있으며, 따라서 모든 뱅크에 의하여 항상 액세스될 수 있어야 하는 리셋 벡터, 인터럽트 벡터, 인터럽트 서비스 루틴, 문자열 상수, 뱅크 스위칭 루틴, C51에 의하여 호출되는 런타임 라이브러리 함수 등은 이 공통 코드 영역에 채워져야 한다.

메모리 시스템을 설계할 때 공통 코드 영역을 얼마의 크기로 잡아야 하는지에 관한 일반적인 규칙은 없으며, 각각의 응용 소프트웨어와 하드웨어적인 제한 사항에 따라 달라진다. 다만, 일반적으로는 이 공통 코드 영역은 각 뱅크들과는 별도의 ROM으로 처리하는 것이

일반적이지만, 만약 ROM의 용량이 전체 공통 코드 영역을 커버하기에 다소 작다면 BL51은 공통 코드 영역의 일부를 잘라내어 각 코드 뱅크의 선두 부분에 공통적으로 복사하여 두도록 지정하는 것이 가능하다. 또한, BL51에서는 공통 코드 영역 전체를 각 코드 뱅크에 포함시키도록 하여 공통 코드 영역을 위한 별도의 ROM을 사용하지 않아도 되도록 할 수 있다.

(2) Code Bank Area

8051은 프로그램의 코드 메모리를 위한 어드레스 버스를 16개 제공하므로 64 KB의 코드 영역을 액세스할 수 있다. 이 코드 뱅크에는 기본적인 16개의 어드레스 라인 이외에 5개까지의 추가적인 어드레스 라인을 사용할 수 있으며, 따라서 32개의 뱅크까지 지원되는 것이다. 추가적인 어드레스 신호로는 통상 8051의 I/O 포트가 사용되지만, XDATA 영역에 매핑된 래치 또는 PIO 들을 이용하여 별도의 하드웨어로 지정할 수도 있다. 이 추가적인 어드레스 신호를 제어함으로써 원하는 코드 뱅크를 선택할 수 있게 된다.

각 뱅크를 스위칭하는 프로그램 루틴은 \C51\LIB 서브 디렉토리의 L51_BANK.A51 파일에 미리 준비되어 있어서 BL51이 이를 사용자 프로그램에 링크시키며, 사용자는 자신의 하드웨어에 맞도록 이 파일을 수정하여 사용할 수 있다.

(3) 뱅크 스위칭을 위한 최적 프로그램 구조

BL51은 각 뱅크에 대해 그 뱅크내에 위치하여 공통 영역이나 다른 뱅크에서 호출될 수 있는 함수들을 위한 점프 테이블을 자동으로 생성한다. 그리고, BL51은 호출되는 프로그램 루틴이 다른 메모리 뱅크에 위치하거나 또는 이 루틴이 공통 영역으로부터 호출될 때만 뱅크 스위칭 동작을 수행한다. 이러한 사실은 프로그램의 성능을 향상시키며 쓸데없는 뱅크 스위칭 동작을 줄여서 시스템의 성능이 저하되지 않도록 한다.

특히, BL51은 이러한 뱅크 스위칭 기법을 사용하는데 필요한 메모리나 스택의 크기가 다른 소프트웨어에 비하여 상당히 작다는 장점을 갖는다. BL51은 각 뱅크 스위칭 동작에 약 50개의 머신 사이클을 필요로 하며, 2바이트의 스택을 추가로 사용한다. 그러나, 사용자 프로그램이 최대의 성능을 위해서는 가급적이면 뱅크 스위칭이 필요하지 않도록 프로그램 구조를 설계하여야 한다. 이는 자주 호출되는 함수나 여러 개의 뱅크에서 호출되는 함수들은 공통 코드 영역에 배치해야 한다는 것을 의미한다.

(4) 공통 코드 영역과 코드 뱅크의 지정

BL51 코드 뱅킹 링커/로케이터는 뱅크 스위칭 영역의 위치와 크기를 지정하고 프로그램 세그먼트들을 어떻게 공통 영역 또는 각 코드 뱅크 영역에 위치시킬 것인지를 지정하기 위하여 BANKAREA, BANKx, COMMON 등의 지시어를 제공한다.

■ BANKAREA

이 지시어는 코드 뱅크가 위치할 영역의 시작번지와 끝번지를 지정하는데 사용한다. 이 번지는 코드 뱅크 ROM이 물리적으로 맵핑될 실제의 번지여야 한다. 하나의 코드 뱅크에 할당된 모든 세그먼트들은 BANKx 지시어로 다르게 정의되지 않는한 반드시 이 번지 영역에 위치해야 한다.

```
BANKAREA(start, end)
```

여기서 *start* 와 *end* 는 각각 코드 뱅크 영역의 시작번지와 끝번지를 나타낸다.

예 BL51 ... BANKAREA(8000h,0FFFFh)

위의 예와 같이 지정하면 BL51은 코드 뱅크 영역은 32 KB를 사용하고 이를 8000H~FFFFH 번지에 위치시킨다.

■ BANKx

이 지시어는 사용자가 어느 프로그램 루틴을 원하는 코드 뱅크에 위치시킬 수 있도록 지시한다. 어느 코드 뱅크에 위치시킬지 명백하게 지정되지 않은 프로그램 코드는 기본적으로 공통 코드 영역에 할당된다. 여기서 [ ]안은 생략 가능한 부분임을 나타낸다.

```
BANKx{filename[(sfname)][, filename(sfname), ...]}
BANKx(saddr[, sfname(addr), sfname(addr), ...])
```

여기서 *x* 는 코드 뱅크의 번호로서 0~31이 사용될 수 있다. { }로 둘러싸인 내부에는 *filename* 으로 표시되는 오브젝트 파일 또는 라이브러리 파일이 지정될 수 있고, ( )로 둘러싸인 내부에는 *sfname* 으로 표시되는 세그먼트 또는 C언어 함수의 이름이 지정될 수 있다. *saddr* 는 이 세그먼트들이 저장되는 영역의 시작번지를 나타내고, *addr* 은 각 세그먼트의 시작번지를 나타낸다.

예 BL51 COMMON(ROOT.OBJ),BANK0(B0.OBJ),BANK1(B1.OBJ),BANK2(B2.OBJ),BANK3(B3.OBJ)
TO PROG.ABS BANKAREA(8000h,0FFFFh)

■ COMMON

이 지시어는 사용자가 원하는 프로그램 루틴을 공통 코드 영역에 위치시킬 수 있도록 지시한다. 이는 CODE 지시어와 기능이 유사하다.

```
COMMON{filename[(sfname)][, filename(sfname), ...]}
COMMON([saddr,] sfname[(addr), sfname(addr), ...])
```

여기서 { }로 둘러싸인 내부에는 *filename* 으로 표시되는 오브젝트 파일 또는 라이브러리 파일이 지정될 수 있고, ( )로 둘러싸인 내부에는 *sfname* 으로 표시되는 세그먼트 또는 C언어 함수의 이름이 지정될 수 있다. *saddr* 는 이 세그먼트들이 저장되는 영역의 시작번지를 나타내고, *addr* 은 각 세그먼트의 시작번지를 나타낸다.

(5) 프로그램 세그먼트를 코드 뱅크에 위치시키는 방법

BL51은 코드 뱅크내에 세그먼트를 위치시킬 때 확실한 원칙이 있다. 오브젝트 모듈이나 라이브러리 파일내에 있는 세그먼트들은 BANKAREA 지시어에서 지정된 번지부터 차례로 할당된다. 그러나, BANKx나 COMMON 지시어에서 ( )안에 지정된 세그먼트들은 *saddr*에서 시작되거나 또는 *saddr*가 지정되지 않으면 0000H 번지부터 할당된다.

예를 들어 전형적인 코드 뱅크 지정 방법은 아래와 같다.

예 BL51 COMMON(ROOT.OBJ),BANK0(B0.OBJ),BANK1(B1.OBJ),BANK2(B2.OBJ),BANK3(B3.OBJ)
TO PROG.ABS BANKAREA(8000h,0FFFFh)

또한, 어느 코드 뱅크에서 개별적인 코드 세그먼트를 포함하도록 하려면 아래와 같이 지정한다. 여기서 BANK2(8000h,?PR?FUNC2?B2)는 C언어에서의 함수 func2를 8000H 번지에서 시작되는 뱅크 2에 할당하겠다는 것을 나타낸다.

예 BL51 COMMON(ROOT.OBJ),BANK0(B0.OBJ),BANK1(B1.OBJ) TO PROG.ABS
BANKAREA(8000h,0FFFFh) BANK2(8000h,?PR?FUNC2?B2)

끝으로, 다음은 개별적인 코드 세그먼트를 특정한 뱅크의 특정한 번지에 할당하도록 지정하는 예이다. 여기서 세그먼트 ?PR?FUNC1?B1 은 뱅크 1에서 8000H 번지부터 할당되고, 세그먼트 ?PR?FUNC2?B2 는 뱅크 1에서 8200H 번지부터 할당된다.

예 BL51 COMMON(ROOT.OBJ),BANK0(B0.OBJ) TO PROG.ABS BANKAREA(8000h,0FFFFh)
BANK1(8000h,?PR?FUNC1?B1,?PR?FUNC2?B2(8200h))

(6) 뱅크를 자동으로 할당하는 방법

BL51은 커맨드 라인에서 { }로 둘러싸인 오브젝트 파일 또는 라이브러리 파일들에 대하여는 순서대로 뱅크 번호를 자동 할당한다. 예를 들어 이러한 자동 뱅크 할당 방법은 아래와 같다. 이렇게 하면 B0.OBJ는 뱅크 0, B1.OBJ는 뱅크 1, B2.OBJ는 뱅크 2, B3.OBJ는 뱅크 3에 각각 할당되고, ROOT.OBJ는 공통 코드 영역에 할당된다.

예 BL51 {B0.OBJ},{B1.OBJ},{B2.OBJ},{B3.OBJ},ROOT.OBJ TO PROG.ABS
BANKAREA(8000h,0FFFFh)

위의 예는 아래와 기능적으로 동일하다.

예 BL51 COMMON(ROOT.OBJ),BANK0(B0.OBJ),BANK1(B1.OBJ),BANK2(B2.OBJ),BANK3(B3.OBJ)
TO PROG.ABS BANKAREA(8000h,0FFFFh)

### 3. L51_BANK.A51 파일의 수정

Keil사의 Code Banking Linker/Locator인 BL51.EXE를 사용하여 뱅크 스위칭 시스템을 설계하려면 코드 뱅크를 몇개 만들 것이며 이것들을 어떻게 스위칭할 것인지를 지정하여야 한다. 이는 \C51\LIB 서브 디렉토리에 제공되는 L51_BANK.A51 파일에서 몇가지의 상수를 수정하고 이를 다시 어셈블하여 링크하면 간단히 해결된다.

(1) L51_BANK.A51 파일의 상수 수정

\C51\LIB 서브 디렉토리에 제공되는 L51_BANK.A51 파일의 선두 부분에는 아래와 같이 EQU 문으로 몇가지의 상수를 정의하고 있다. 여기에는 뱅크 스위칭을 위하여 추가적으로 사용할 어드레스 라인은 무엇인지 또는 코드 뱅크는 몇 개를 사용할지를 지정한다.

```
;------------------------------------------------------------------------------
;  This file is part of the BL51 Banked Linker/Locater package
;  Copyright (c) 1988-1997 Keil Elektronik GmbH and Keil Software, Inc.
;  Version 1.4c
;------------------------------------------------------------------------------
;************************* Configuration Section ******************************
?B_NBANKS       SET     16       ; Define max. Number of Banks                 *
;                                ; the max. value for ?B_BANKS is 32           *
; Note: Valid numbers for ?B_NBANKS are 2, 4, 8, 16, or 32.  If you have      *
;       a application with 3 banks you must define ?B_NBANKS to 4!            *
;                                                                              *
?B_MODE         EQU     1        ; 0 for Bank-Switching via 8051 Port          *
;                                ; 1 for Bank-Switching via XDATA Port         *
;                                                                              *
?B_RTX          EQU     0        ; 0 for applications without RTX-51 FULL      *
;                                ; 1 for applications using RTX-51 FULL        *
;                                                                              *
IF  ?B_MODE = 0;                                                               *
;-----------------------------------------------------------------------------*
; if ?BANK?MODE is 0 define the following values                               *
; For Bank-Switching via 8051 Port define Port Address / Bits                  *
?B_PORT         EQU     P1       ; default is P1                               *
?B_FIRSTBIT     EQU     3        ; default is Bit 3                            *
;-----------------------------------------------------------------------------*
ENDIF;                                                                         *
;                                                                              *
```

```
IF  ?B_MODE = 1;                                                            *
;---------------------------------------------------------------------------*
; if ?BANK?MODE is 1 define the following values                            *
; For Bank-Switching via XDATA Port define XDATA Port Address / Bits        *
?B_XDATAPORT    EQU     0FFFFH  ; default is XDATA Port Address 0FFFFH      *
?B_FIRSTBIT     EQU     0       ; default is Bit 0                          *
;---------------------------------------------------------------------------*
ENDIF;                                                                      *
;                                                                           *
;****************************************************************************
```

여기서, ?B_NBANKS는 지원할 코드 뱅크의 수를 나타내며 2~32로 지정할 수 있다. 뱅크가 2개라면 단지 1개의 추가적인 어드레스 선이 필요하고, 코드 뱅크가 2~4개라면 2개의 어드레스가 필요하며, 코드 뱅크가 5~8개라면 3개의 어드레스가 필요하다. 또한, 코드 뱅크가 9~16개라면 4개의 어드레스가 필요하며, 코드 뱅크가 17~32개라면 5개의 어드레스가 필요하다.

?B_MODE는 뱅크 스위칭을 위하여 추가적인 어드레스 선으로 8051의 병렬 I/O 포트 단자를 사용할지 또는 XDATA 영역에 지정되는 번지를 사용할지를 지정한다. 이 값을 0으로 지정하면 8051의 병렬 I/O 포트 단자가 사용되며, 1로 지정하면 XDATA 영역에 지정되는 번지가 사용된다.

?B_PORT는 ?B_MODE가 0으로 지정되었을 경우에만 해당되는 것으로 추가적인 어드레스 선으로 사용될 8051의 병렬 I/O 포트를 지정한다. 예를 들어서 P1을 사용한다면 이를 P1으로 지정해도 되고 이것의 번지값인 90H로 지정해도 된다.

?B_XDATAPORT는 ?B_MODE가 1로 지정되었을 경우에만 해당되는 것으로 추가적인 어드레스 선으로 사용될 XDATA 영역의 번지를 지정한다. 여기에는 0000H~FFFFH가 사용될 수 있으며, 디폴트로 FFFFH가 사용된다. 사용자 소스 프로그램이 C51 언어가 아니라 PL/M-51 또는 A51로 작성되는 경우에는 ?B_XDATAPORT로 지정된 번지 및 ?B_CURRENTBANK 번지는 프로그램의 시작 단계에서 0으로 초기화되어 있어야 한다.

?B_FIRSTBIT는 추가적인 어드레스 선으로 사용하도록 지정된 포트에서 첫번째의 비트 번호를 지정한다. 예를 들어 P1.3과 P1.4가 추가적인 어드레스 선으로 사용된다면 ?B_FIRSTBIT는 3으로 지정되어야 한다. 이때 P1의 나머지 비트들은 물론 다른 용도로 사용될 수 있다.

이렇게 하여 올바르게 수정된 L51_BANK.A51 파일은 A51 어셈블러로 다시 어셈블하여 L51_BANK.OBJ로 만들어야하며, BL51에서 DEFAULTLIBRARY로 설정되어 있는 경우에는 이 오브젝트 파일이 자동으로 링크되고 그렇지 않으면 이를 명시적으로 지정하여 링크시켜야 한다.

(2) L51_BANK.A51 파일에서 기타 Public Symbol의 수정

L51_BANK.A51 파일에는 이밖에도 사용자에게 편리하도록 2개의 PUBLIC 심볼을 정의해 두

고 있다. ?B_CURRENTNABK는 현재 선택된 메모리 뱅크를 포함하는 DATA 또는 SFR 영역내의 메모리 위치이다. 이 메모리 위치는 디버깅을 위하여 그 내용이 읽힐 수 있으며, 이를 수정하더라도 뱅크 스위칭 동작에는 영향을 주지 않는다.

_SWITCHBANK는 뱅크 어드레스가 사용자 프로그램에 의하여 선택되도록 허용하는 C51 호환의 함수이다. 이 함수는 뱅크 스위칭을 위하여 사용될 수 있으며, 그 사용 예는 다음과 같다.

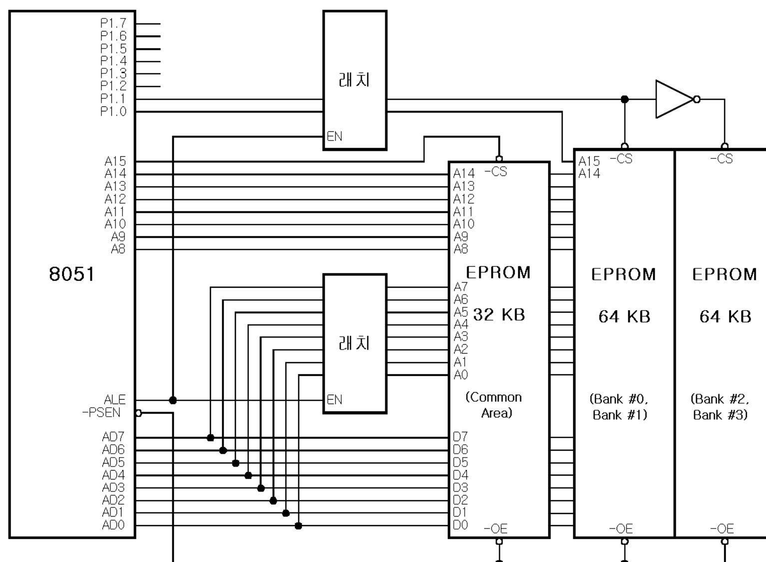
예
```
extern void switchbank(unsigned char bank_number)
  .
  .
swichbank(0);
```

**4. BL51에서 뱅크 스위칭을 사용한 메모리 시스템의 설계 예**

이제 여기서는 Keil사의 Code Banking Linker/Locator인 BL51.EXE를 기반으로 뱅크 스위칭 방법을 사용하여 메모리를 확장하는 시스템의 설계 예를 몇가지 살펴보기로 한다.

(1) 32KB의 공통 영역과 4개의 32KB 뱅크를 가지는 메모리 시스템

이 예는 앞의 <그림 1>에서 보았던 바와 같이 0000H~7FFFH 번지에 공통 코드 영역을 두고 8000H~FFFFH 번지에 4개의 32 KB 뱅크를 사용하는 시스템이다. 이를 구현하기 위하여 설계한 하드웨어의 구성은 <그림 2>와 같이 된다.




<그림 2> 32KB의 공통 영역과 4개의 32KB 뱅크를 가지는 시스템의 회로도

이렇게 뱅크 스위칭을 위하여 2개의 뱅크 선택 어드레스 선 P1.0 및 P1.1이 사용되었으므로 올바른 하드웨어 설정(configuration)을 위하여 L51_BANK.A51 파일을 아래와 같이 수정하여야 한다.

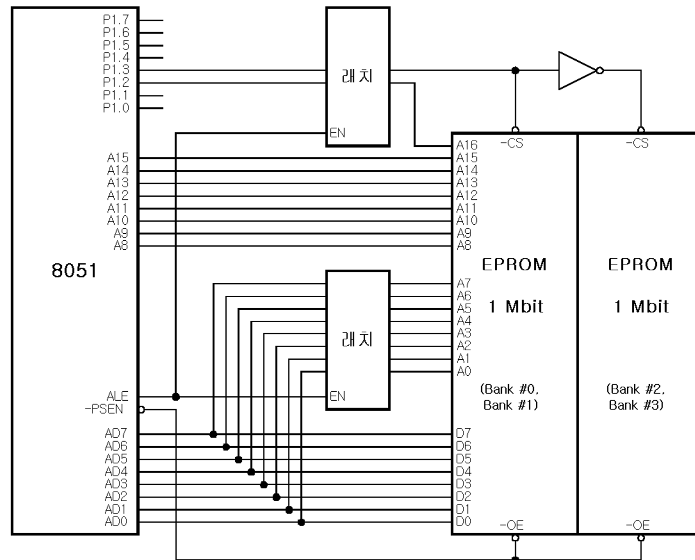
예

```
?B_NBANKS      SET    4      ; 4 banks are required
?B_MODE        EQU    0      ; 8051 I/O port is used.
?B_PORT        EQU    90H    ; P1 is used as address line.
?B_FIRSTBIT    EQU    0      ; P1.0 is the 1st address line.
```

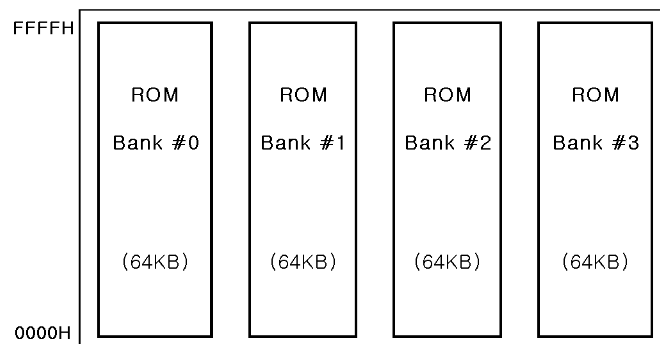
이때 BL51은 공통 코드 영역에 들어갈 프로그램 모듈과 각 뱅크에 들어갈 모듈을 아래와 같이 지정할 수 있다.

예

```
BL51 COMMON(ROOT.OBJ),BANK0(B0.OBJ),BANK1(B1.OBJ),BANK2(B2.OBJ),BANK3(B3.OBJ)
TO PROG.ABS BANKAREA(8000h,0FFFFh)
```




<그림 3> 4개의 64KB 뱅크를 가지는 시스템의 회로도




<그림 4> 4개의 64KB 뱅크를 가지는 시스템의 메모리 구성도

(2) 4개의 64KB 뱅크를 가지는 메모리 시스템

이 예는 2개의 1 Mbit(128 KB) EPROM을 사용하는 것으로 전제로 하여 <그림 3>과 같이 설계하기로 한다. 이렇게 하였을 때의 메모리 구성은 <그림 4>와 같게 된다.

이렇게 뱅크 스위칭을 위하여 2개의 뱅크 선택 어드레스 선 P1.2 및 P1.3이 사용되었으므로 올바른 하드웨어 설정(configuration)을 위하여 L51_BANK.A51 파일을 아래와 같이 수정하여야 한다.

예
```
?B_NBANKS       SET    4      ; 4 banks are required
?B_MODE         EQU    0      ; 8051 I/O port is used.
?B_PORT         EQU    90H    ; P1 is used as address line.
?B_FIRSTBIT     EQU    2      ; P1.2 is the 1st address line.
```

이때 BL51은 자동적으로 공통 코드 영역의 코드 및 데이터를 각 뱅크로 복사하여 각 뱅크 EPROM의 공통 영역이 서로 동일한 내용으로 되도록 한다. 이러한 경우에는 디폴트로 어드레스 영역 0000H~FFFFH가 뱅크 어드레스로 잡히므로 BL51에서 BANKAREA 지시어를 사용하지 말아야 한다.

따라서, 이때 BL51은 다음과 같이 사용될 수 있다.

예
```
BL51 COMMON(ROOT.OBJ),BANK0(B0.OBJ),BANK1(B1.OBJ),BANK2(B2.OBJ),BANK3(B3.OBJ)
TO PROG.ABS
```

### 5. HEX 파일의 생성 및 뱅크 EPROM의 프로그래밍

이제 뱅크 스위칭 사용 기법을 설명하는 마지막 단계로서 각 뱅크에 해당하는 코드를 어떻게 인텔 HEX 파일로 만들어 EPROM에 구울 수 있는지를 설명한다. 이를 위하여 아래와 같이 실제의 간단한 C 언어 소스 예제를 생각한다.

예
```
/***** ROOT.C : main program *****/

#include <stdio.h>
#include <reg51.h>

extern void func0(void);
extern void func1(void);
extern void func2(void);

void main(void)
{ SCON = 0x52;                /* initialize serial port */
  TMOD = 0x20;
  TCON = 0x40;
  PCON = 0x00;
  TH1 = 0xfd;

  printf("Mnain program calls a function in bank 0 ...\n");
  func0();
  printf("Mnain program calls a function in bank 1 ...\n");
  func1();
```

```
  printf("Mnain program calls a function in bank 2 ...\n");
  func2();

  while(1);
}
```

예

```
/***** B0.C : bank 0 program *****/

#include <stdio.h>

extern void func3(void);

void func0(void)
{ printf("Function in bank 0 calls a function in bank 3 ...\n");
  func3();
}
```

예

```
/***** B1.C : bank 1 program *****/

#include <stdio.h>

extern void func3(void);

void func1(void)
{ printf("Function in bank 1 calls a function in bank 3 ...\n");
  func3();
}
```

예

```
/***** B2.C : bank 2 program *****/

#include <stdio.h>

extern void func3(void);

void func2(void)
{ printf("Function in bank 2 calls a function in bank 3 ...\n");
  func3();
}
```

예

```
/***** B3.C : bank 3 program *****/

#include <stdio.h>

void func3(void)
{ printf("This is a function in bank 3 !\n");
}
```

이제 이러한 프로그램을 작성하였으면 다음과 같이 C51을 사용하여 이것들을 각각 컴파일한다.

예

```
C51 ROOT.C DEBUG EXTEND
C51 B0.C DEBUG EXTEND
C51 B1.C DEBUG EXTEND
C51 B2.C DEBUG EXTEND
C51 B3.C DEBUG EXTEND
```

다음에는 여기에서 생성된 오브젝트 파일들을 BL51을 사용하여 다음과 같이 링크한다. 여기서 TO PROG.ABS를 지정하지 않는다면 링크된 결과는 ROOT라는 이름의 파일로 저장된다. 이때 함께 생성되는 ROOT.M51 파일을 보면 메모리 할당에 관한 정보나 각 뱅크를 스위칭하기 위한 점프 테이블에 관한 정보를 볼 수 있다.

예
```
BL51 COMMON(ROOT.OBJ),BANK0(B0.OBJ),BANK1(B1.OBJ),BANK2(B2.OBJ),BANK3(B3.OBJ)
TO PROG.ABS BANKAREA(8000h,0FFFFh)
```

이렇게 하여 생성된 PROG.ABS 파일은 banked OMF format으로 되어 있기 때문에 이를 dScope에서 사용하거나 또는 나중에 EPROM으로 구우려면 다음과 같이 OC51 Banked Object File Converter를 이용하여 각 뱅크에 해당하는 standard OMF file들로 변환하여야 한다.

예
```
OC51 PROG.ABS
```

그러면 PROG.B00, PROG.B01, PROG.B02, PROG.B03과 같이 4개의 파일이 생성된다. 이들 각각의 파일에는 공통 코드 영역의 프로그램과 해당 뱅크에 해당하는 프로그램이 들어 있다. 이제 이것들을 각각 EPROM에 굽기 위하여 인텔 HEX 파일로 변환하려면 OH51을 사용하여 아래와 같이 처리하면 된다.

예
```
OH51 PROG.B00 HEXFILE(PROG00.HEX)
OH51 PROG.B01 HEXFILE(PROG01.HEX)
OH51 PROG.B02 HEXFILE(PROG02.HEX)
OH51 PROG.B03 HEXFILE(PROG03.HEX)
```

이상에서 Keil사의 8051 소프트웨어 개발툴을 사용하는 경우를 기준으로 하여 뱅크 스위칭 기법을 사용한 메모리 확장 기술에 대하여 설명하였는데, 이러한 사항에는 하드웨어와 소프트웨어의 밀접한 관련하에 처리되는 까다로운 내용이 많으므로, 이를 제대로 활용하기 위해서는 실제 시스템을 설계하고 만들어 가면서 좀더 자세히 연구하고 시험해 보는 것이 필요하다는 것을 마지막으로 강조한다.

이상에 관하여 추가적인 사항을 알고자 하면 아래 참고문헌을 참조하기 바란다.

[참고문헌] 8051 Utilities User's Guide, Keil Software, 1995
(1) BL51 Code Banking Linker/Locator
(2) LIB51 Library Manager
(3) OC51 Banked Object File Converter
(4) OH51 Object-Hex Converter