

HMS800C C Compiler Guide (Ver. 1.2)

ABOV 의 800/810 Core MCU 를 위한 컴파일러 사용시 유의사항.



Version 1.2
Published by FAE Team
©2009 ABOV Semiconductor Co., Ltd. All rights reserved.

Additional information of this manual may be served by ABOV Semiconductor offices in Korea or Distributors.

ABOV Semiconductor reserves the right to make changes to any information here in at any time without notice.

The information, diagrams and other data in this manual are correct and reliable; however, ABOV Semiconductor is in no way responsible for any violations of patents or other rights of the third party generated by the use of this manual.

목차

목차	2
1. 컴파일러 제약 사항	3
1.1 변수형	3
1.2 전역변수 초기화.....	3
1.3 가변인자.....	3
1.4 Standard Library	3
2. 사용 시 유의사항	4
2.1 char vs int.....	4
2.2 Volatile 형 변수	4
2.3 Bit flag	4
2.4 지역 변수.....	6
2.5 전역변수 PAGE 할당	6
2.6 함수 인자.....	7
2.7 함수 프로토 타입 선언	7
2.8 파일 확장 자.....	7
3. 커멘드 라인 도구 사용방법	8
3.1 오브젝트 덤프	8
3.2 코드 사이즈.....	8
문서 수정 내역	9

1. 컴파일러 제약 사항

1.1 변수형

변수는 1 바이트 변수와 2 바이트 정수형 변수만 제공합니다.

1 바이트 변수: char
, unsigned char.

2 바이트 변수: int, short, long
, unsigned int, unsigned short, unsigned long
(즉, int, short, long 형 모두 2 바이트 변수로 처리 함)

부동 소수형 변수들은 제공하지 않습니다.

1.2 전역변수 초기화

전역변수 및 static 지역 변수의 초기화는 PAGE0 만 지원 합니다.

Example

```
unsigned char buf[3] = {0x00, 0x01, 0x02};

void func( void )
{
    static local_buf[3] = {7,8,9};
}
```

1.3 가변인자

HMS800CCC 는 가변 인자를 지원하지 않습니다. 가변 인자는 printf() 등에서 함수 인자를 가변적으로 사용하는 것을 말합니다.

1.4 Standard Library

Standard Library 는 제한 적으로 지원하며, 지원하는 라이브러리들은 매뉴얼에 명시되어 있습니다. 매뉴얼은 "C:\HMS800C\MANUAL" 폴더에 있습니다.

2. 사용 시 유의사항

2.1 char vs int

HMS800CC 에서 char 형은 1byte, int 형은 2byte 변수 입니다. 8 bit MCU 에선 1byte 변수를 사용하는 것이 가장 효율 적이기 때문에 가능한 변수는 1byte 변수인 char 형 혹은 unsigned char 형 변수를 사용하는 것이 효율적입니다.

2.2 Volatile 형 변수

Volatile 은 다른 함수에서 값이 변경될 수 있는 변수 임을 명시하는 키워드 입니다.

Example

```
volatile unsigned char a;

void function( void ) {
    a = 2;
    if( a != 2 ) {
        do_something();
    }
}
```

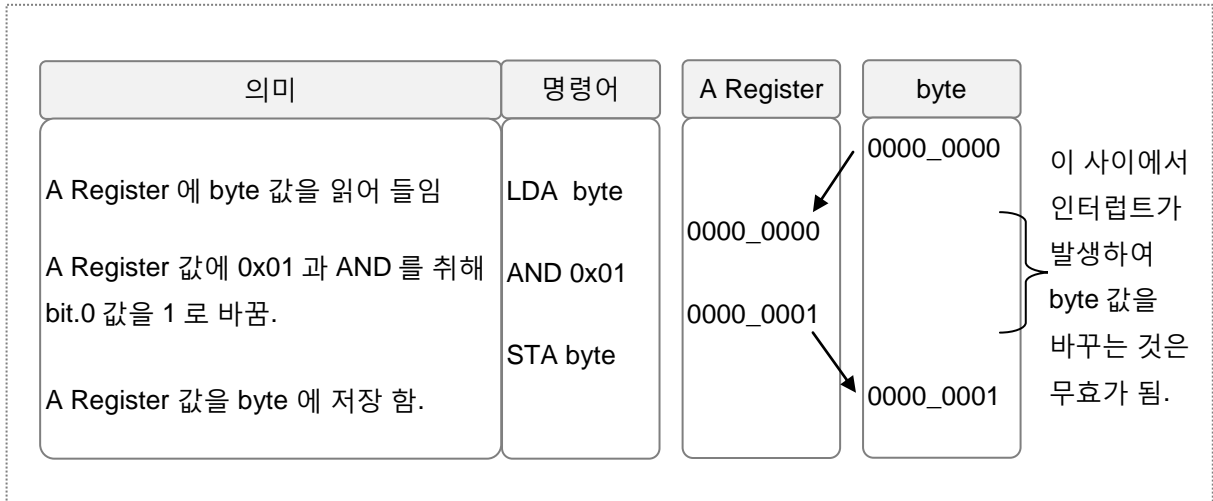
위 예제 코드에서 만일 변수 'a' 가 volatile 형으로 선언 되어 있지 않다면, 'if' 문은 언제나 '거짓' 이 되기 때문에 최적화 과정에서 'if'문장과 블록은 컴파일 된 코드가 생성되지 않습니다.

따라서, 인터럽트 함수 등에 의해서 변수 'a'가 변경 될 수 있는 경우 위의 예와 같이 변수 'a'를 반드시 volatile 형으로 선언해야 합니다. 그러면, "a=2;" 로 대입한 이후에 다른 곳에서 'a'가 변경 되어 'if'문장이 참이 될 수도 있기 때문에 'if'문장과 블록 부분의 코드도 생성이 됩니다.

2.3 Bit flag

Bit 연산이 "atomic operation"이 되도록 보장하려면 asm("set1 ..."); 과 asm("clr1 ...")을 사용해야 합니다.

"atomic operation"은 쪼개지지 않는 어떠한 동작(명령)을 말합니다. Bit 연산지가 쪼개질 경우 다음과 같은 문제가 발생할 수 있습니다.



위의 예에서, A Register 를 사용하여 byte 변수의 0 번째 비트를 변경하는 3 개의 명령어가 수행 되는 도중에 인터럽트가 발생하여 byte 값을 변경 할 경우 마지막 “STA byte” 명령에 의해 인터럽트에서 변경한 사항은 무효가 되어버립니다.

따라서 위와 같이 하나의 byte 를 인터럽트에서도 수정하고, main 루틴에서도 수정할 경우 수정하는 루틴은 수행 중에 쪼개지면(중간에 인터럽트가 끼어들면) 안 됩니다.(“atomic operation”이 되어야 함).

기본적으로 C 언어에는 하나의 bit 만 SET/CLEAR 하는 연산자가 없기 때문에 AND/OR 을 이용 구현 하며, “atomic operation”을 보장하지 않습니다.

HMS800CCC 의 경우 AND/OR 연산이 비트 하나만 변경할 경우 800 Core 의 set1/clr1 명령을 이용하여 컴파일 하도록 최적화 하기는 하지만 100% 보장하지는 않습니다. 따라서 “atomic operation”을 보장해야 하는 경우 asm(“”)을 이용하여 직접 set1/clr1 명령을 사용하는 것이 가장 안전합니다.

혹은 비트 연산 전에 인터럽트를 Disable 하고 연산 후에 Enable 할 수도 있지만 코드가 길어 지므로 비 효율적입니다.

사용 예로 다음과 같이 define 하여 사용할 수 있습니다. 단 이 경우 bit flag 는 PAGE0 에만 선언 할 수 있습니다.

Example

```
#include <hms800.h>
BYTE somebyte;
#define f_SomeFlag(a) if(a) asm("set1 _somebyte.0"); \
                    else asm("clr1 _somebyte.0");
#define f_SomeFlagT    ( somebyte.bit.bit0 ) //읽어 올 때 사용.
                                                    //읽어오는 방법은 무관함.
```

2.4 지역 변수

하나의 함수에서 사용된 지역 변수가 3byte 이상이 될 경우 일부를 전역변수로 바꾸어 지역 변수 사용을 3byte 미만으로 유지 하는 것이 효율적입니다.

지역 변수는 STACK 영역을 사용하기 때문에 현재 사용중인 총 지역 변수 크기는 STACK 크기를 넘을 수 없습니다.

현재 사용중인 총 지역 변수 크기는 현재 호출 중인 함수들에 따라 달라지기 때문에 실행 상황에 따라 가변적입니다. 또한 STACK 이 다 차서 Overflow 가 발생할 경우 이에 대한 보호장치가 없기 때문에 오동작을 유발 할 수 있습니다. 따라서 함수가 지나치게 중첩되어서 호출되지 않도록 코드를 작성하는 편이 안전합니다.

2.5 전역변수 PAGE 할당

전역변수가 할당되는 RAM PAGE 는 프로그래머가 명시해야 하며, 명시되지 않은 전역 변수는 PAGE0 에 할당 됩니다.

또한, 하나의 배열이 하나의 PAGE(256 byte)를 넘어서 선언 될 수는 없습니다.

PAGE1 이나 PAGE2 같은 키워드를 사용하면, 확장 영역에 변수를 선언할 수 있습니다. 하지만 이 키워드가 어디에 위치하는가에 따라 의미가 달라집니다.

,

- PAGE2 키워드가 변수 이름 앞에 있으면 이후에 정의된 변수들은 모두 PAGE2 에 할당 됩니다.

따라서 다음의 예는 모든 'a', 'b', 'c' 변수가 Page2 에 할당 됩니다.

Example

```
int PAGE2 a,b,c;
```

다음 의 예에서는 'a'는 Page0 에 할당되고 'b','c' 는 Page2 에 할당 됩니다.

Example

```
int a,PAGE2 b, c;
```

- 'PAGE2'키워드가 변수 이름 뒤에 오면, 해당 변수만 Page2 에 할당 됩니다.따라서 다음의 예에서 'a' 와 'b' 는 Page0 에 할당되고, 'c' 는 Page2 에 할당 됩니다.

Example

```
int a,b,c PAGE2;
```

다음의 예에서 'a','c' 는 Page0 에 할당되고 'b' 는 Page2 에 할당 됩니다.

Example

```
int a,b PAGE2, c;
```

가능하면 stack 영역이 있는 PAGE 에는 전역변수를 할당하지 않는 편이 좋다.(대부분 PAGE1 에 위치함) 전역 변수 사용으로 stack 영역이 줄어들면 그 만큼 stack overflow 가 발생할 가능성이 증가하기 때문입니다.

2.6 함수 인자

함수의 인자로 사용되는 변수가 늘어남에 따라 코드의 복잡도는 기하 급수 적으로 늘어 난다. 따라서, 함수의 인자는 2byte 이하로 관리하여 주는 것이 효율적입니다.

2.7 함수 프로토 타입 선언

함수의 프로토타입을 선언하지 않고 호출할 경우, 함수인자를 실제 데이터 타입과 상관없이 컴파일러 기본 형인 int(2byte) 변수로 간주하여 함수를 호출하게 됩니다.

따라서 반드시 함수는 프로토타입 혹은 함수의 정의를 선언한 뒤에 호출해야 하며, 특히 다른 파일에 선언된 함수를 호출할 경우 extern 키워드로 프로토타입을 명시한 뒤에 사용해야 합니다.

2.8 파일 확장 자

C 소스 파일의 확장 자는 반드시 소문자 “c”이어야 합니다. Hms800CCC 의 모태가 된 gcc 에서 대문자 “C”는 C++소스로 인식하기 때문에 C 소스 파일의 확장 자는 반드시 소문자 “c”로 만들어야 합니다.

마찬가지로 헤더 파일의 확장 자도 반드시 소문자 ‘h’이어야 합니다.

3. 커멘드 라인 도구 사용방법

HMS800CCC 는 다음 과 같은 커멘드 라인도구를 제공하며 이들을 통해, 디스어셈블 된 코드를 보거나 최종 컴파일 된 코드 사이즈 등을 알 수 있습니다.

3.1 오브젝트 덤프

hms800-elf-objdump 도구를 사용하여 생성된 코드에서 심볼 정보, 디스어셈블 코드 등을 볼 수 있습니다.

Example

hms800-elf-objdump -t target.out	->	심볼 정보 출력
hms800-elf-objdump -D target.out	->	디스어셈블 코드 출력.
hms800-elf-objdump -D -S target.out	->	C 소스와 함께 디스어셈블 코드 출력

그냥 hms800-elf-objdump 만 실행하면 도움말이 나온다.

꼭 *.out 파일을 사용해야 합니다. (HEX 나 OTP 파일에는 심볼 및 디버깅 정보가 없음)

3.2 코드 사이즈

hms800-elf-size 도구를 사용 생성된 최종 파일 (HEX,OTP) 에서 코드 사이즈 출력합니다.

Example

hms800-elf-size target.otp	->	코드 사이즈 출력.
----------------------------	----	------------

문서 수정 내역

VERSION 1.2 (December 23, 2009)

오타 수정

“2.8 파일 확장 자” 보완.

VERSION 1.1 (August 31, 2009)

“2.8 파일 확장 자” 추가

VERSION 1.0 (July 3, 2009)

초안.