Order this document
by EB283/D

Motorola Semiconductor Engineering Bulletin

EB283

C Macro Definitions for the MC68HC11C0

By John Bodnar
Austin, Texas

Introduction

With more microcontroller users moving to high level languages like C, macro definition files like the one outlined in this document can speed software development efforts.

The file reproduced in the following pages is available on the Freeware Data System. Download and unzip the hc11c0h.zip file from the MCU11 directory. The hc11c0h.zip file includes an ASCII text copy of this documentation and the actual hc11c0.h text file. The hc11c0.h file, and others like it, use Motorola's designated register and bit names for each device described. Any user already familiar with MC68HC11 assembly language and architecture (a requirement even for those who think they will only program in C), will be able to make use of this file readily.

Conventions

The contents of the actual file will be designated with the Courier typeface while commentary will appear in the Helvetica typeface used in this paragraph.

Thus, these lines appear in the hc11c0.h file:

```
/* MOTOROLA INC.
 *
 * FILENAME: hc11c0.h
 *
 * DESCRIPTION: Register and bit macro definitions for the
MC68HC11C0 microcontroller.
 *
 * CREATED: 12/03/93
 *
 * NOTE: Your comments, suggestions, and corrections are requested *
* and greatly appreciated.
 */
```

First references to key terms appear in bold type, and C keywords and expressions appear in italics.

Concepts, Development, and Usage

In C language, we can make just about anything an lvalue, that is, something that appears to the left of the equal sign in an assignment expression. We can even use a number as an lvalue. In particular, we would like to use register addresses as lvalues. To do this, we must cast the lvalue as a pointer to a particular data type.

For example

```
(unsigned char *) 0x1000
```

would be an lvalue that points to an unsigned character (an 8-bit unsigned value) at memory location 0x1000 ($1000 for those used to assembly language). In this particular form, however, we cannot yet assign a value to the memory location. To do this, we must de-reference the pointer. De-referencing a pointer specifies the value that is pointed

to and not the pointer itself. So, to assign the value 0xFF to memory location 0x1000, we would use this C assignment expression:

```
*(unsigned char *) 0x1000 = 0xFF
```

Likewise, if we wish to assign the contents of memory location 0x1000 to the variable A, we would use this assignment:

```
A = *(unsigned char *) 0x1000
```

This is all that is really necessary to manipulate the memory mapped registers of the MC68HC11. Unfortunately, *(unsigned char *) `0x1000` is not particularly indicative of the function memory location `0x1000` performs (PORT A on most MC68HC11 devices). The extra typing required to use this memory location can also be a source of minor, but unnecessar, compilation errors. A better idea is to use the following line (remember, lines appearing in the hc11c0.h file appear in the Courier typeface):

```
#define REGISTER unsigned char
```

Thus to access memory location 0x1000, we can now type:

```
A = *(REGISTER *) 0x1000
```

This is an improvement, but it would be even better if we could define a register as PORTA or DDRC as we do when programming in assembly. Thus, this line

```
#define SOMEDEVICE *(REGISTER *)0x1000
```

will allow us to address 0x1000 in a very convenient fashion. For example, we can now type

```
SOMEDEVICE = 0xFF;
```

to assign 0xFF to memory location 0x1000, and we can also type

```
A = SOMEDEVICE;
```

to assign the contents of 0x1000 to the variable A.

The MC68HC11 has an INIT register which is used to remap internal RAM and registers to the beginning of any 4-K page of memory. Some applications may require register remapping, so it would be convenient if we could make a simple change to the macro definition file to account for this.

This line (part of hc11c0.h) allows us to do this:

```
#define REG_BASE 0x0000
```

We can thus use this macro definition to handle register relocation:

```
#define SOMEDEVICE *(REGISTER *)(REG_BASE + 0x00)
```

If we leave REG_BASE as 0x0000, then pointers to the MC68HC11's peripheral registers will be addressed at 0x0000 in our source code. If we decide to remap the registers to 0x4000, we can simply replace 0x0000 in the #define REG_BASE macro with 0x4000.

NOTE: *This does not actually modify the MC68HC11's INIT register. This must be done by modifying your C compiler's run-time startup code. Refer to your compiler's documentation before making any such changes.*

Before proceeding with the rest of the hc11c0.h file, we need to understand the use of C's volatile keyword. By specifying a variable as volatile, we tell the C compiler not to optimize expressions using that variable.

```
#define PORT *(REGISTER *)(REG_BASE + 0xA0)
void main()
{
        PORT = 0x00;
        etc... /* PORT is not used until while(PORT) */
        while (PORT)
        {
                etc...
        }
}
```

In this program fragment, we immediately initialize PORT to 0x00, but we will not reference it again until the while (PORT) expression. Unless PORT were to somehow change, while (PORT) would be false, and code in the braces immediately following would not execute.

Some C compilers may view this as unnecessary if PORT never changes, and it is possible these lines could be optimized out of the resulting object code.

On the MC68HC11, PORT may point to a bi-directional I/O port whose inputs may change during the course of program execution, thus the while (PORT) expression could actually be true when it is executed. As a precaution, we can designate the PORT pointer as volatile so that the optimizer will not attempt to remove any questionable references to it. We would thus change the #define macro to be

```
#define PORT *(volatile REGISTER *)(REG_BASE + 0xA0)
```

By doing this, references to PORT will not be optimized. Several registers on the MC68HC11 can change without the intervention of user code. These registers include port data registers (PORTC), peripheral status registers (SPSR), peripheral data registers (SCDR, ADR1), flag registers (TFLG1), and timer registers (TCNT, TIC3).

We could use the volatile keyword with every register macro definition to simplify matters, but this runs counter to good code documentation. By specifying only those registers which require it as volatile, the resulting code will be better documented. Only registers which can receive data externally or be changed by the processor without user intervention will be declared volatile. Write-only registers will be recognized easily because they will lack the volatile declaration.

These macro definitions are used for the registers on the MC68HC11C0 device:

```
#define PORTA   (*(volatile REGISTER*)(REG_BASE + 0x00))
#define DDRA    (*(REGISTER *)(REG_BASE + 0x01))
#define PORTF   (*(volatile REGISTER *)(REG_BASE + 0x02))
#define DDRF    (*(REGISTER *)(REG_BASE + 0x03))
#define FISTAT (*(volatile REGISTER *)(REG_BASE + 0x04))
#define FINTEN (*(REGISTER *)(REG_BASE + 0x05))
#define DIOCTL (*(REGISTER *)(REG_BASE + 0x07))
#define PORTD   (*(volatile REGISTER *)(REG_BASE + 0x08))
```

```
#define DDRD    (*(REGISTER *)(REG_BASE + 0x09))
#define PORTE   (*(volatile REGISTER *)(REG_BASE + 0x0A))
#define CFORC   (*(REGISTER *)(REG_BASE + 0x0B))
#define OC1M    (*(REGISTER *)(REG_BASE + 0x0C))
#define OC1D    (*(REGISTER *)(REG_BASE + 0x0D))
```

The following registers (TCNT, TICx, and TOCx) are declared as unsigned integers because they are 16-bit registers and should be accessed as such. It is much simpler and clearer to change, for example, the output compare 4 register by using
TOC4 = 0x4000, TOC4 = TCNT + 0x20FF, or TOC4 += 0x3200.

```
#define TCNT    (*(volatile unsigned int *)(REG_BASE + 0x0E))
#define TIC1    (*(volatile unsigned int *)(REG_BASE + 0x10))
#define TIC2    (*(volatile unsigned int *)(REG_BASE + 0x12))
#define TIC3    (*(volatile unsigned int *)(REG_BASE + 0x14))
#define TOC1    (*(unsigned int *)(REG_BASE + 0x16))
#define TOC2    (*(unsigned int *)(REG_BASE + 0x18))
#define TOC3    (*(unsigned int *)(REG_BASE + 0x1A))
#define TOC4    (*(unsigned int *)(REG_BASE + 0x1C))
#define TI4O5   (*(volatile unsigned int *)(REG_BASE + 0x1E))
#define TCTL1   (*(REGISTER *)(REG_BASE + 0x20))
#define TCTL2   (*(REGISTER *)(REG_BASE + 0x21))
#define TMSK1   (*(REGISTER *)(REG_BASE + 0x22))
#define TFLG1   (*(volatile REGISTER *)(REG_BASE + 0x23))
#define TMSK2   (*(REGISTER *)(REG_BASE + 0x24))
#define TFLG2   (*(volatile REGISTER *)(REG_BASE + 0x25))
#define PACTL   (*(REGISTER *)(REG_BASE + 0x26))
#define PACNT   (*(volatile REGISTER *)(REG_BASE + 0x27))
#define SPCR    (*(REGISTER *)(REG_BASE + 0x28))
#define SPSR    (*(volatile REGISTER *)(REG_BASE + 0x29))
#define SPDR    (*(volatile REGISTER *)(REG_BASE + 0x2A))
#define BAUD    (*(REGISTER *)(REG_BASE + 0x2B))
```

SCCR1 is declared volatile because it has the R8 bit, the ninth data bit received when SCI mode 1 is used. The remaining bits in this register are write only.

```
#define SCCR1   (*(volatile REGISTER *)(REG_BASE + 0x2C))
#define SCCR2   (*(REGISTER *)(REG_BASE + 0x2D))
#define SCSR    (*(volatile REGISTER *)(REG_BASE + 0x2E))
#define SCDR    (*(volatile REGISTER *)(REG_BASE + 0x2F))
```

ADCTL is declared volatile because bit 7, the conversion complete flag (CCF), is changed without user intervention. The remaining bits in this register are write only.

```
#define ADCTL   (*(volatile REGISTER *)(REG_BASE + 0x30))
#define ADR1    (*(volatile REGISTER *)(REG_BASE + 0x31))
#define ADR2    (*(volatile REGISTER *)(REG_BASE + 0x32))
#define ADR3    (*(volatile REGISTER *)(REG_BASE + 0x33))
#define ADR4    (*(volatile REGISTER *)(REG_BASE + 0x34))
#define INIT2   (*(REGISTER *)(REG_BASE + 0x37))
#define OPT2    (*(REGISTER *)(REG_BASE + 0x38))
#define OPTION (*(REGISTER *)(REG_BASE + 0x39))
#define COPRST (*(REGISTER *)(REG_BASE + 0x3A))
#define HPRIO   (*(REGISTER *)(REG_BASE + 0x3C))
#define INIT    (*(REGISTER *)(REG_BASE + 0x3D))
#define TEST1   (*(REGISTER *)(REG_BASE + 0x3E))
#define CONFIG (*(REGISTER *)(REG_BASE + 0x3F))
#define VCSADR (*(REGISTER *)(REG_BASE + 0x40))
#define PGSADR (*(REGISTER *)(REG_BASE + 0x42))
#define PGEADR (*(REGISTER *)(REG_BASE + 0x43))
#define MXHADR (*(REGISTER *)(REG_BASE + 0x44))
#define MXLADR (*(REGISTER *)(REG_BASE + 0x45))
#define GP1SADR (*(REGISTER *)(REG_BASE + 0x46))
#define GP1EADR (*(REGISTER *)(REG_BASE + 0x47))
```

```
#define GP2SADR (*(REGISTER *)(REG_BASE + 0x48))
#define GP2EADR (*(REGISTER *)(REG_BASE + 0x49))
#define GP3SADR (*(REGISTER *)(REG_BASE + 0x4A))
#define GP3EADR (*(REGISTER *)(REG_BASE + 0x4B))
#define GP4SADR (*(REGISTER *)(REG_BASE + 0x4C))
#define GP4EADR (*(REGISTER *)(REG_BASE + 0x4D))
#define GP5SADR (*(REGISTER *)(REG_BASE + 0x4E))
#define GP5EADR (*(REGISTER *)(REG_BASE + 0x4F))
#define PWCLK   (*(REGISTER *)(REG_BASE + 0x60))
#define PWPOL   (*(REGISTER *)(REG_BASE + 0x61))
#define PWSCAL (*(volatile REGISTER *)(REG_BASE + 0x62))
#define PWEN    (*(REGISTER *)(REG_BASE + 0x63))
#define PWCNT1 (*(volatile REGISTER *)(REG_BASE + 0x66))
#define PWCNT2 (*(volatile REGISTER *)(REG_BASE + 0x67))
#define PWPER1 (*(REGISTER *)(REG_BASE + 0x6A))
#define PWPER2 (*(REGISTER *)(REG_BASE + 0x6B))
#define PWDTY1 (*(REGISTER *)(REG_BASE + 0x6E))
#define PWDTY2 (*(REGISTER *)(REG_BASE + 0x6F))
#define PPAR    (*(REGISTER *)(REG_BASE + 0x70))
#define PGEN    (*(REGISTER *)(REG_BASE + 0x71))
#define DODM    (*(REGISTER *)(REG_BASE + 0x75))
#define PORTH   (*(volatile REGISTER *)(REG_BASE + 0x7C))
#define DDRH    (*(REGISTER *)(REG_BASE + 0x7D))
#define PORTG   (*(volatile REGISTER *)(REG_BASE + 0x7E))
#define DDRG    (*(REGISTER *)(REG_BASE + 0x7F))
```

C also allows us to declare individual bit fields as constants. This allows us to make simple register bit assignments and comparisons. For instance

```
while (!(SPSR & SPIF))
```

can be used to halt program execution until the SPI status register SPIF bit has set.

Likewise, we can use

```
SPCR = SPIE + SPE + MSTR + CPHA + SPR
```

to configure the SPI for master operation with interrupts using clock phase 1 and a baud rate of E clock divided by 4. We can also use these constants to clear individual bit fields in the timer flag registers.

```
TFLG1 &= OC3F
```

This clears output compare flag 3 without affecting the other bits in the TFLG1 register.

A partial list of the macro definitions that are used for the register bit fields on the MC68HC11C0 device follows. For a complete list, download the file .hc11c0h.zip from the Freeware Data System.

```
/* Bit names for general use */.

#define bit7    0x80
#define bit6    0x40
#define bit5    0x20
#define bit4    0x10
#define bit3    0x08
#define bit2    0x04
#define bit1    0x02
#define bit0    0x01
```

```
/* PORTA bit definitions 0x00 */

    #define PA7     bit7
    #define PA6     bit6
    #define PA5     bit5
    #define PA4     bit4
    #define PA3     bit3
    #define PA2     bit2
    #define PA1     bit1
    #define PA0     bit0

/* DDRG bit definitions 0x7F */

    #define DDG7    bit7
    #define DDG6    bit6
    #define DDG5    bit5
    #define DDG4    bit4
    #define DDG3    bit3
    #define DDG2    bit2
    #define DDG1    bit1
    #define DDG0    bit0
```

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution, P.O. Box 5405, Denver, Colorado 80217, 1-800-441-2447 or 1-303-675-2140. Customer Focus Center, 1-800-521-6274

JAPAN: Nippon Motorola Ltd.: SPD, Strategic Planning Office, 141, 4-32-1 Nishi-Gotanda, Shinigawa-Ku, Tokyo, Japan. 03-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd., 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298

Mfax™, Motorola Fax Back System: RMFAX0@email.sps.mot.com; http://sps.motorola.com/mfax/; TOUCHTONE, 1-602-244-6609; US and Canada ONLY, 1-800-774-1848

HOME PAGE: http://motorola.com/sps/

MOTOROLA