



Application Note:
Flash Data storage with the
Microchip MC104 Family
PC104 Single board computer

©2007 Calmotion LLC, All rights reserved

Calmotion LLC
9909 Topanga Canyon Blvd. #322
Chatsworth, CA 91311
www.calmotion.com

Introduction

This application note is intended to assist users of the MC104 family of products to implement flash data storage while the MC104 family of product is in operation. Typical usage includes storage of variable data and data logging.

Storage of data in the MC104 family of products is fairly straightforward as long as a few general guidelines are observed.

1. Interrupts must be disabled during a flash write operation.
2. The location selected to write data to does not conflict with program space dedicated to proper operation of the user's program.
3. The number of write operations to flash does not exceed the specified limit.
4. The user accounts for the 64 byte minimum block erase sequence.

Example

The following example implements a combination of C and assembly programming. Other array/pointer programming techniques than the example below can be used. The following was intended to facilitate understanding of writing to flash rather than more efficient pointer notation.

```
...
unsigned char *ftptr;           // flash pointer
unsigned char flash_test[] = "Writing to flash is easy!"; // declared array of text,
// could be an array of data
...

/* initialize pointer to the first location of the array */

ftptr = flash_test;

/* call a function to write the array of data program memory */

write_flash(0x5000,ftptr);

...
```

If using the ICD2 programmer and in debugger mode, the user can check to see that the write to flash operation was successful by first refreshing the ICD2 memory screen. Click "Read" from the "Debugger" drop down menu. Then, click "View" and select "Program Memory". Scroll down to the appropriate memory location.

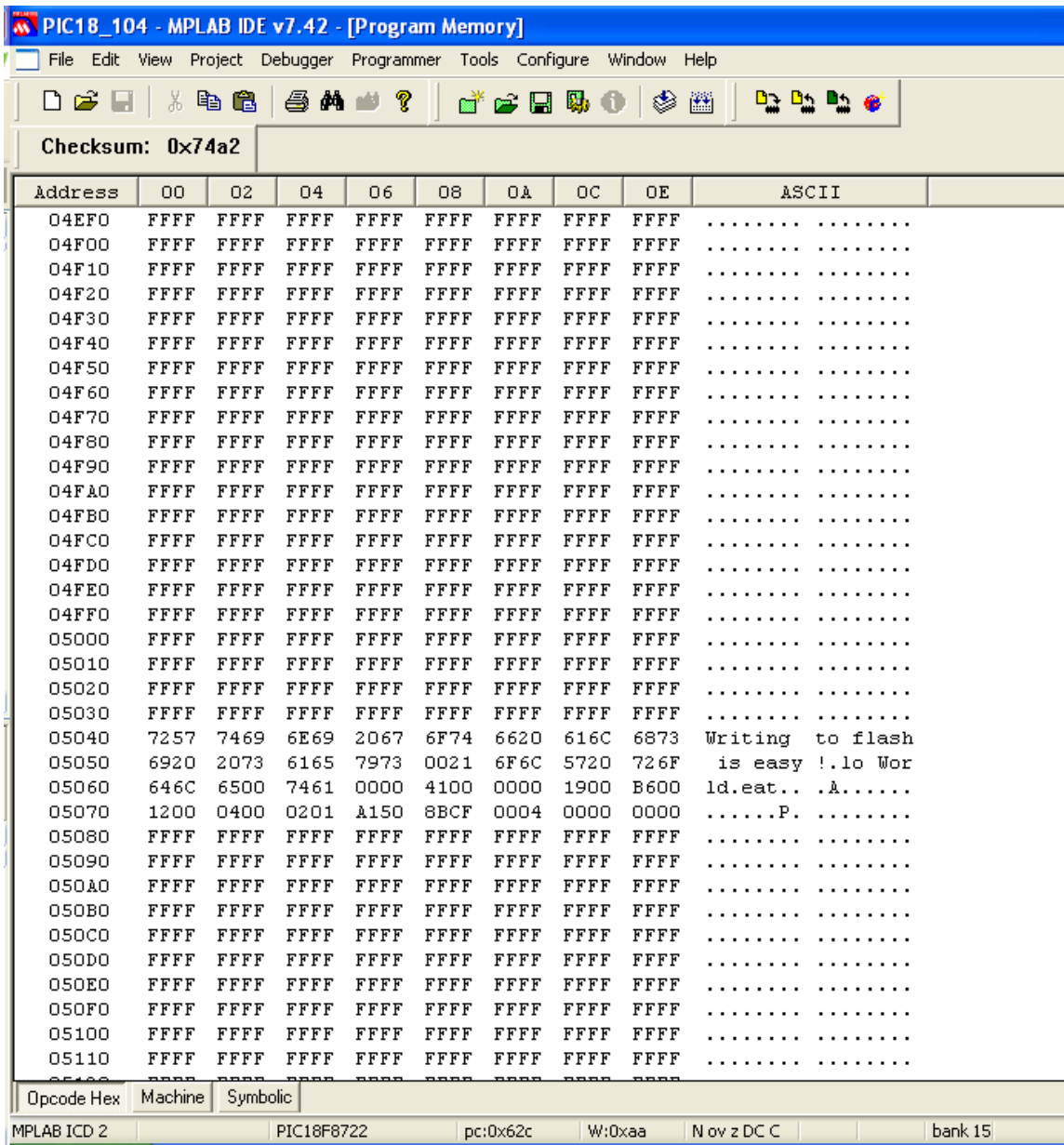


Figure 1

Note the offset of 64 bytes from the location written 0x5040 versus 0x5000. The user may re-write the write_flash routine if desired. Also, beware that random data will be placed in the 64 byte block if not all the byte locations are accounted for as is the case in this example.

Reading the flash can easily be accomplished by declaring a rom pointer and reading the data back using pointer arithmetic.

```
...  
flash_Ptr = (rom unsigned char*)0x5040;
```

```
Flash_data = *(flash_Ptr + offset);
```

```
.....
```

```
void write_flash(unsigned long address, unsigned char *buffer)
```

```
{  
    TBLPTR = address;  
    EECON1bits.EEPGD = 1;  
    EECON1bits.CFGS = 0;  
    EECON1bits.FREE = 1;  
    EECON1bits.WREN = 1;  
    EECON2 = 0x55;  
    EECON2 = 0xAA;  
    EECON1bits.WR = 1;  
    Nop();  
    while(EECON1bits.WR);
```

```
    TBLPTR = address;  
    for(i=0; i<64; i++) {  
        TABLAT = buffer[i];  
        _asm  
            TBLWTPOSTINC  
        _endasm  
    }
```

```
    EECON1bits.EEPGD = 1;  
    EECON1bits.CFGS = 0;  
    EECON1bits.FREE = 0;  
    EECON1bits.WREN = 1;  
    EECON2 = 0x55;  
    EECON2 = 0xAA;  
    EECON1bits.WR = 1;  
    Nop();  
    while(EECON1bits.WR);  
    EECON1bits.WREN = 0;  
}
```