AN914

Dynamic Memory Allocation for the MPLAB® C18 C Compiler

Author: Ross M. Fosler
Microchip Technology Incorporated

INTRODUCTION

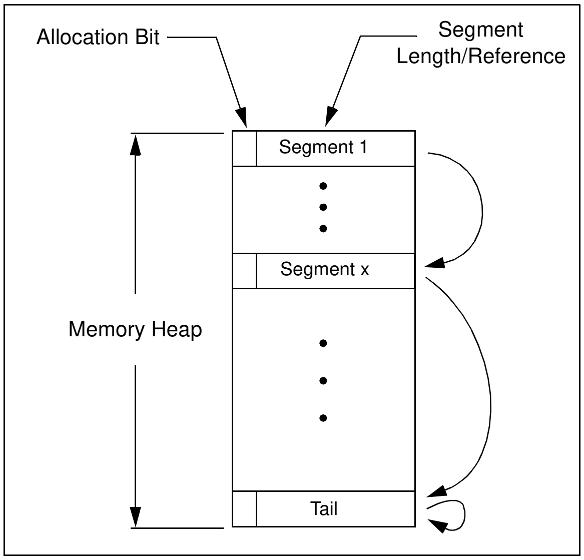
Dynamic memory allocation is a nice functionality that is provided with virtually all PC-based compilers. However, not all microcontroller compilers have such capability, most likely due to the lack of a sophisticated operating system with memory management. Although most applications are static in nature, there are cases where a need for dynamic allocation of memory resources exists. Examples include any number of network protocols that have a dynamically specified nature. This application note presents a simple and efficient method for dynamic memory allocation without the need of an operating system.

THE MODEL

The model is based on a simple form of a linked list. A block of memory referred to as the dynamic heap is split into segments. Each segment has a single-byte header that references the next segment in the list via an offset, as well as indicating whether the segment is allocated. Allocation is specified by a single bit. Figure 1 shows an example. Consequently, the reference implicitly identifies the length of the segment. The heap is terminated with a special header that references itself, referred to as the “tail”.

Why use single-byte headers? The segment headers are specifically designed to be a single byte wide to achieve excellent execution performance, reduce code size and minimize loss of memory space to segment control information. Essentially, one byte references are easier and faster to manipulate than multi-byte relative or absolute references. Plus, they do not consume as much space. However, some fundamental limits are imposed by this methodology. The maximum segment size is 126 bytes, or the size of the heap, whichever is smaller. The smallest segment size is one byte, resulting in a maximum number of segments of one-half of the number of bytes in the heap minus one. For example, in a 512-byte heap, one could expect to dynamically allocate as many as 255 single-byte segments.

FIGURE 1: SIMPLE HEAP EXAMPLE



Although this model will allow dynamic allocation down to a single byte, doing so sacrifices performance and memory. With more segments within the heap, more time is required to attempt to allocate memory. In addition, every segment requires a header byte; therefore, a large number of smaller segments require more memory than a small number of large segments. In the 255-segment example mentioned previously, 50% of the heap is lost to segment header information.

There is also one other potential problem, especially with smaller segments: memory fragmentation. Fragmentation could ultimately doom an application by reducing the largest allocatable block of memory. Thus, dynamic allocation should be restricted to larger blocks to maintain efficiency and effective use of the heap.

Applications that are likely to encounter fragmentation issues should provide a method to handle allocation failures. The implementation depends on the complexity of the application. For some applications, a system Reset may be sufficient. For applications with more advanced memory requirements, it may be necessary to provide allocation management functions. An example might be to force non-critical tasks to give up their memory allocations as needed, then re-allocate memory to them as required.

SUPPORTING FUNCTIONS

There are three functions that manage the heap:

- `SRAMAlloc`: Allocate memory
- `SRAMFree`: Free previously allocated memory
- `SRAMInitHeap`: Initialize the dynamic heap

SRAMAlloc

```
unsigned char * NEAR SRAMAlloc(NEAR unsigned
char nBytes)
```

`SRAMAlloc` is used to allocate a segment of memory within the heap. When it is called, a new segment is created in the heap. Essentially, larger non-allocated segments are split to achieve the requested segment size. If there are a number of smaller non-allocated segments, they will be merged together to create a single larger segment. If a segment of sufficient size cannot be allocated, then an error is returned to the calling application; otherwise, a 16-bit pointer to the segment is returned, which is the next address after the stored segment header. Figure 2 outlines the basic program flow. The application must remember the pointer to successfully free the memory.

SRAMFree

```
void SRAMFree(unsigned char * NEAR pSRAM)
```

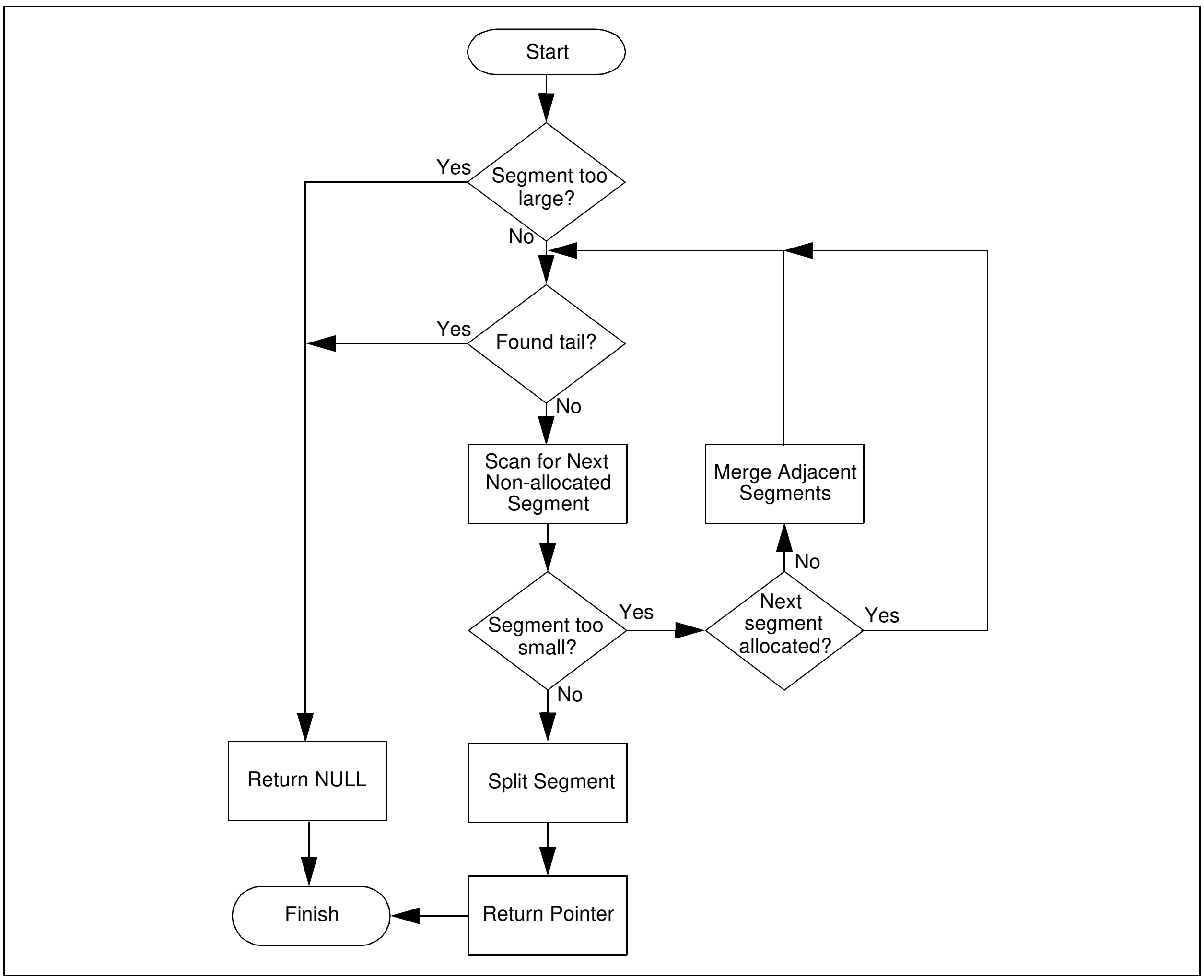
This function is used to free a previously allocated memory segment. It allows future calls to `SRAMAlloc` to merge or split this segment as necessary. The pointer returned from allocation must be passed to successfully free the block of memory.

SRAMInitHeap

```
void SRAMInitHeap(void)
```

This function must be called at least one time to initialize the heap with the minimum number of segment headers and the tail. This function could also be called to initialize the heap. The minimum number is always the value of (`MAX_HEAP_SIZE`/126), rounded up for any remainder. For example, a 256-byte heap will be initialized with three segments.

FIGURE 2: SEGMENT ALLOCATION FLOWCHART



 © 2004 Microchip Technology Inc.

SETTING UP

Compile Time Options

There are only two compile time options to be set:

- `NEAR_MODEL`: This specifies whether the code uses access registers or normal data space for general processing. There is some small performance improvement using access memory.
- `MAX_HEAP_SIZE`: This specifies the size of the dynamic heap. This value should correlate with the section size specified in the linker script.

The Linker Script

The source code reserves a block of memory specified by a section in the linker file named `_SRAM_ALLOC_HEAP`. Refer to the attached linker script in **Appendix D: "Linker Script"** for an example.

PERFORMANCE

The performance of dynamic allocation varies significantly depending on the build options, the number of segments in the heap, the positions and sizes of the segments and the size of the heap. In the example code, with the build options selected in the example project, allocation can occur in as little as 100 instruction cycles. In other basic tests, with 4 to 5 segments previously allocated, allocation can occur in as much as 450 instruction cycles.

Freeing allocated segments is relatively fixed compared to allocation. In the example code, with the build options selected in the example project, freeing allocated memory only requires 18 instruction cycles.

MEMORY USAGE

The memory usage varies depending on the build options, the number of segments in the heap, the positions and sizes of the segments and the size of the heap. In the example code presented here and with the build options selected in the example project, only 452 bytes of program memory and 20 bytes of data memory are used. In addition, another 512 bytes of data memory are reserved for the dynamic heap. Note that the heap size can be increased or decreased to meet the needs of the application.

© 2004 Microchip Technology Inc.

APPENDIX A: ABOUT THE SOURCE CODE

A complete listing of the source code (in C) and the accompanying linker script for the application described here follows in Appendices B, C and D.

The complete code project, including all required linker and header files, is also available from Microchip in electronic format; it may be downloaded from the corporate web site as a Zip archive file. Additionally, this application is included as modular code with Microchip's Application Maestro™ software.

To download the archive, or to get more information on Application Maestro, please visit the Microchip corporate web site at:

www.microchip.com

© 2004 Microchip Technology Inc.

APPENDIX B: MEMORY ALLOCATION SOURCE CODE

```
/*****************************************************************************
 *
 *              Simple SRAM Dynamic Memory Allocation
 *
 *****************************************************************************
 * FileName:        smalloc.c
 * Dependencies:
 * Processor:       PIC18F with CAN
 * Compiler:        C18 02.20.00 or higher
 * Linker:          MPLINK 03.40.00 or higher
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the "Company") is intended and supplied to you, the Company's
 * customer, for use solely and exclusively with products manufactured
 * by the Company.
 *
 * The software is owned by the Company and/or its supplier, and is
 * protected under applicable copyright laws. All rights are reserved.
 * Any use in violation of the foregoing restrictions may subject the
 * user to criminal sanctions under applicable laws, as well as to
 * civil liability for the breach of the terms and conditions of this
 * license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 * TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 * IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 *
 * This is a simple dynamic memory allocation module. The following are the
 * supported services:
 *
 * unsigned char * NEAR SRAMalloc(NEAR unsigned char nBytes)
 * void SRAMfree(unsigned char * NEAR pSRAM)
 * void SRAMInitHeap(void)
 *
 * This version of the dynamic memory allocation limits the segment size
 * to 126 bytes. This is specifically designed such to enable better
 * performance by limiting pointer manipulation.
 *
 *
 * How it works:
 * The model is based on a simple form of a linked list. A block of memory
 * refered to as the dynamic heap is split into segments. Each segment
 * has a single byte header that references the next segment in the list
 * as well as indicating whether the segment is allocated. Consiquently
 * the reference implicitly identifies the length of the segment.
 *
 * This method also enables the possibility of allowing a large number
 * of memory allocations. The maximum is limited by the defined heap size.
 *
 * SRAMalloc() is used to split or merge segments to be allocated.
 * SRAMfree() is used to release segments.
 *
```

```
* Example:
*  ----------
*  |  0x7F  |0x200    Header Seg1
*  |        |
*  |        |
*  |        |
*  |        |
*  |        |
*  |        |
*  |  0x89  |  0x27F  Header Seg2 (allocated)
*  |        |
*  |        |
*  |  0x77  |  0x288  Header Seg3
*  |        |
*  |        |
*  |        |
*  |        |
*  |        |
*  |        |
*  |        |
*  |  0x00  |0x2FF    Tail
*  ----------
*
*
*  Bit 7  Bit 6  Bit 5  Bit 4  Bit 3  Bit 2  Bit 1  Bit 0
*
*  Alloc------------- reference to next Header --------------
*
*
* Recomendations:
* Although this model will allow dynamic allocation down to a single byte,
* doing so sacrifices performance. With more segments within the heap, more
* time is required to attempt to allocate memory. Plus every segment requires
* a header byte; therefore, smaller segments require more memory. There is
* also the possibility of fragmentation, which could ultimately doom an
* application by reducing the largest allocatable block of memory. Thus the
* recomendation is to allocate at least 8 bytes of memory.
*
*
*
* Author             Date           Version    Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Ross Fosler        05/25/03       v1.03      First release
*
*****************************************************************************/
```

 © 2004 Microchip Technology Inc.

```
#define NEAR_MODEL
#define MAX_HEAP_SIZE      0x200

#if defined(NEAR_MODEL)
#define NEAR    near
#else
#define NEAR
#endif

#define _MAX_SEGMENT_SIZE  0x7F
#define _MAX_HEAP_SIZE     MAX_HEAP_SIZE-1

/*********************************************************************
 * Segment header data type
 ********************************************************************/
typedef union _SALLOC
{
    unsigned char byte;
    struct _BITS
    {
        unsigned count:7;
        unsigned alloc:1;
    }bits;
}SALLOC;

/*********************************************************************
 * Reserve the memory heap
 ********************************************************************/

#pragma udata_SRAM_ALLOC_HEAP
unsigned char _uDynamicHeap[MAX_HEAP_SIZE];

/*********************************************************************
 * Set the memory type
 ********************************************************************/

#if defined(NEAR_MODEL)
#pragma udata access_SRAM_ALLOC
#else
#pragma udata _SRAM_ALLOC
#endif

/*********************************************************************
 * Private function declarations
 ********************************************************************/

NEAR unsigned char _SRAMmerge(SALLOC * NEAR pSegA);
```

```
/*********************************************************************
 * Function:       unsigned char * SRAMalloc(unsigned char length)
 *
 * PreCondition:   A memory block must be allocated in the linker,
 *                 and the memory headers and tail must already be
 *                 set via the function SRAMInitHeap().
 *
 * Input:          unsigned char nBytes - Number of bytes to allocate.
 *
 * Output:         unsigned char * - A pointer to the requested block
 *                 of memory.
 *
 * Overview:       This functions allocates a chunk of memory from
 *                 the heap. The maximum segment size for this
 *                 version is 126 bytes. If the heap does not have
 *                 an available segment of sufficient size it will
 *                 attempt to create a segment; otherwise a NULL
 *                 pointer is returned. I allocation is succeessful
 *                 then a pointer to the requested block is returned.
 *
 * Note:           The calling function must maintain the pointer
 *                 to correctly free memory at runtime.
 *
 ********************************************************************/

unsigned char * NEAR SRAMalloc(NEAR unsigned char nBytes)
{
    SALLOC * NEAR pHeap;
    SALLOC * NEAR temp;
    NEAR SALLOC segHeader;
    NEAR unsigned char segLen;

    // Do not allow allocation above the max minus one bytes
    if (nBytes > (_MAX_SEGMENT_SIZE - 1)) return (0);

    // Init the pointer to the heap
    pHeap = (SALLOC *)_uDynamicHeap;

    while (1)
    {
        // Get the header of the segment
        segHeader = *pHeap;

        // Extract the segment length from the segment
        segLen = segHeader.bits.count - 1;

        // A null segment indicates the end of the table
        if (segHeader.byte == 0) return (0);

        // If this segment is not allocated then attempt to allocate it
        if (!(segHeader.bits.alloc))
        {
            // If the free segment is too small then attempt to merge
            if (nBytes > segLen)
            {
                // If the merge fails them move on to the next segment
                if (!(_SRAMmerge(pHeap))) pHeap += segHeader.bits.count;
            }
            else
```


© 2004 Microchip Technology Inc.

```
            // If the segment length matches the request then allocate the
            // header and return the pointer
            if (nBytes == segLen)
            {
                // Allocate the segment
                (*pHeap).bits.alloc = 1;

                // Return the pointer to the caller
                return ((unsigned char *)(pHeap + 1));
            }

            // Else create a new segment
            else
            {
                // Reset the header to point to a new segment
                (*pHeap).byte = nBytes + 0x81;

                // Remember the pointer to the first segment
                temp = pHeap + 1;

                // Point to the new segment
                pHeap += (nBytes + 1);

                // Insert the header for the new segment
                (*pHeap).byte = segLen - nBytes;

                // Return the pointer to the user
                return ((unsigned char *) temp);
            }
        }

        // else set the pointer to the next segment header in the heap
        else
        {
            pHeap += segHeader.bits.count;
        }
    }
}

/*********************************************************************
 * Function:       void SRAMfree(unsigned char * pSRAM)
 *
 * PreCondition:   The pointer must have been returned from a
 *                 previously allocation via SRAMalloc().
 *
 * Input:          unsigned char * pSRAM - pointer to the allocated
 *
 * Output:         void
 *
 * Overview:       This function de-allocates a previously allocated
 *                 segment of memory.
 *
 * Note:           The pointer must be a valid pointer returned from
 *                 SRAMalloc(); otherwise, the segment may not be
 *                 successfully de-allocated, and the heap may be
 *                 corrupted.
 ********************************************************************/

void SRAMfree(unsigned char * NEAR pSRAM)
{
    // Release the segment
    (*(SALLOC *)(pSRAM - 1)).bits.alloc = 0;
}
```

```
/*********************************************************************
 * Function:       void SRAMInitHeap(void)
 *
 * PreCondition:   none
 *
 * Input:          void
 *
 * Output:         void
 *
 * Overview:       This function initializes the dynamic heap. It
 *                 inserts segment headers to maximize segment space.
 *
 * Note:           This function must be called at least one time.
 *                 And it could be called more times to reset the
 *                 heap.
 ********************************************************************/
void SRAMInitHeap(void)
{
    unsigned char * NEAR pHeap;
    NEAR unsigned int count;

    pHeap = _uDynamicHeap;
    count = _MAX_HEAP_SIZE;

    while (1)
    {
        if (count > _MAX_SEGMENT_SIZE)
        {
            *pHeap = _MAX_SEGMENT_SIZE;
            pHeap += _MAX_SEGMENT_SIZE;
            count = count - _MAX_SEGMENT_SIZE;
        }
        else
        {
            *pHeap = count;
            *(pHeap + count) = 0;
            return;
        }
    }
}

/*********************************************************************
 * Function:       unsigned char _SRAMmerge(SALLOC * NEAR pSegA)
 *
 * PreCondition:   none
 *
 * Input:          SALLOC * NEAR pSegA - pointer to the first segment.
 *
 * Output:         unsigned char - returns the length of the
 *                 merged segment or zero if failed to merge.
 *
 * Overview:       This function tries to merge adjacent segments
 *                 that have not been allocated. The largest possible
 *                 segment is merged if possible.
 ********************************************************************/

NEAR unsigned char _SRAMmerge(SALLOC * NEAR pSegA)
{
    SALLOC * NEAR pSegB;
    NEAR SALLOC uSegA, uSegB, uSum;


    // Init the pointer to the heap
    pSegB = pSegA + (*pSegA).byte;
```


© 2004 Microchip Technology Inc.

```
    // Extract the headers for faster processing
    uSegA = *pSegA;
    uSegB = *pSegB;

    // Quit if the tail has been found
    if (uSegB.byte == 0) return (0);

    // If either segment is allocated then do not merge
    if (uSegA.bits.alloc || uSegB.bits.alloc) return (0);

    // If the first segment is max then nothing to merge
    if (uSegA.bits.count == _MAX_SEGMENT_SIZE) return (0);

    // Get the sum of the two segments
    uSum.byte = uSegA.byte + uSegB.byte;


    // If the sum of the two segments are > than the largest segment
    // then create a new segment equal to the max segment size and
    // point to the next segments
    if ((uSum.byte) > _MAX_SEGMENT_SIZE)
    {
        (*pSegA).byte = _MAX_SEGMENT_SIZE;
        pSegA += _MAX_SEGMENT_SIZE; //(*pSeg1).byte;
        pSegB += uSegB.byte; //(*pSeg2).byte ;
        (*pSegA).byte = pSegB - pSegA;

        return (_MAX_SEGMENT_SIZE);
    }
    // Else combine the two segments into one segment and
    // do not adjust the pointers to the next segment
    else
    {
        return ((*pSegA).byte = uSum.byte);
    }
}
```

APPENDIX C: TEST CODE

```
#include "sralloc.h"

void main(void)
{

    unsigned char * pTest1;
    unsigned char * pTest2;
    unsigned char * pTest3;
    unsigned char * pTest4;
    unsigned char * pTest5;
    unsigned char * pTest6;
    unsigned char * pTest7;

    SRAMInitHeap();

    while (1)
    {
        pTest1 = SRAMalloc(1);
        pTest2 = SRAMalloc(126);
        SRAMfree(pTest2);
        SRAMfree(pTest1);

        pTest1 = SRAMalloc(8);
        pTest2 = SRAMalloc(40);
        pTest3 = SRAMalloc(8);
        pTest4 = SRAMalloc(20);
        pTest5 = SRAMalloc(12);
        pTest6 = SRAMalloc(56);
        pTest7 = SRAMalloc(92);

        SRAMfree(pTest2);
        SRAMfree(pTest1);

        pTest1 = SRAMalloc(30);
        pTest2 = SRAMalloc(120);

        SRAMfree(pTest1);
        SRAMfree(pTest4);
        SRAMfree(pTest3);
        SRAMfree(pTest7);
        SRAMfree(pTest6);
        SRAMfree(pTest5);
        SRAMfree(pTest2);
    }

}
```


© 2004 Microchip Technology Inc.

APPENDIX D: LINKER SCRIPT

```
// Dynamic Memory Allocation Linker Script Example

LIBPATH  .

FILES c018i.o
FILES clib.lib
FILES p18f458.lib

CODEPAGE NAME=vectors         START=0x0      END=0x29      PROTECTED
CODEPAGE NAME=page            START=0x2A     END=0x7FFF
CODEPAGE NAME=idlocs          START=0x200000 END=0x200007  PROTECTED
CODEPAGE NAME=config          START=0x300000 END=0x30000D  PROTECTED
CODEPAGE NAME=devid           START=0x3FFFFE END=0x3FFFFF  PROTECTED
CODEPAGE NAME=eedata          START=0xF00000 END=0xF000FF  PROTECTED

ACCESSBANK NAME=accessram     START=0x0      END=0x5F
DATABANK NAME=gpr0            START=0x60     END=0xFF
DATABANK NAME=gpr1            START=0x100    END=0x1FF
//DATABANK NAME=gpr2          START=0x200    END=0x2FF
//DATABANK NAME=gpr3          START=0x300    END=0x3FF
DATABANK NAME=gpr4            START=0x400    END=0x4FF
DATABANK NAME=gpr5            START=0x500    END=0x5FF
DATABANK NAME=bankedsfr       START=0xF00    END=0xF5F     PROTECTED
ACCESSBANK NAME=accesssfr     START=0xF60    END=0xFFF     PROTECTED

SECTION NAME=CONFIG    ROM=config

DATABANK NAME=sramalloc       START=0x200    END=0x3FF
SECTION NAME=_SRAM_ALLOC_HEAP RAM=sramalloc

STACK SIZE=0x100       RAM=gpr5
```

NOTES:

 © 2004 Microchip Technology Inc.

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart and rfPIC are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, microID, MXDEV, MXLAB, PICMASTER, SEEVAL, SmartShunt and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Application Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICkit, PICDEM, PICDEM.net, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, rfLAB, Select Mode, SmartSensor, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2004, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company’s quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Atlanta
3780 Mansell Road, Suite 130
Alpharetta, GA 30022
Tel: 770-640-0034
Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848
Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, IN 46902
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888
Fax: 949-263-1338

San Jose
1300 Terra Bella Avenue
Mountain View, CA 94043
Tel: 650-215-1444
Fax: 650-961-0286

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Australia
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Unit 706B
Wan Tai Bei Hai Bldg.
No. 6 Chaoyangmen Bei Str.
Beijing, 100027, China
Tel: 86-10-85282100
Fax: 86-10-85282104

China - Chengdu
Rm. 2401-2402, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200
Fax: 86-28-86766599

China - Fuzhou
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506
Fax: 86-591-7503521

China - Hong Kong SAR
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Shanghai
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700
Fax: 86-21-6275-5060

China - Shenzhen
Rm. 1812, 18/F, Building A, United Plaza
No. 5022 Binhe Road, Futian District
Shenzhen 518033, China
Tel: 86-755-82901380
Fax: 86-755-8295-1393

China - Shunde
Room 401, Hongjian Building, No. 2
Fengxiangnan Road, Ronggui Town, Shunde
District, Foshan City, Guangdong 528303, China
Tel: 86-757-28395507 Fax: 86-757-28395571

China - Qingdao
Rm. B505A, Fullhope Plaza,
No. 12 Hong Kong Central Rd.
Qingdao 266071, China
Tel: 86-532-5027355 Fax: 86-532-5027205

India
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-22290061 Fax: 91-80-22290062

Japan
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5932 or
82-2-558-5934

Singapore
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan
Kaohsiung Branch
30F - 1 No. 8
Min Chuan 2nd Road
Kaohsiung 806, Taiwan
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan
Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Austria
Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45-4420-9895 Fax: 45-4420-9910

France
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany
Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy
Via Quasimodo, 12
20025 Legnano (MI)
Milan, Italy
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands
P. A. De Biesbosch 14
NL-5152 SC Drunen, Netherlands
Tel: 31-416-690399
Fax: 31-416-690340

United Kingdom
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44-118-921-5869
Fax: 44-118-921-5820

02/17/04

© 2004 Microchip Technology Inc.