IP-RESISTOR

REFERENCE MANUAL

817-10-000-4000
Version 1.0
October 2006

ALPHI TECHNOLOGY CORPORATION

6202 S. Maple Avenue #120
Tempe, AZ 85283 USA
Tel: (480) 838-2428
Fax: (480) 838-4477

NOTICE

The information in this document has been carefully checked and is believed to be entirely reliable. While all reasonable efforts to ensure accuracy have been taken in the preparation of this manual, ALPHI TECHNOLOGY assumes no responsibility resulting from omissions or errors in this manual, or from the use of information contain herein.

ALPHI TECHNOLOGY reserves the right to make any changes, without notice, to this or any of ALPHI TECHNOLOGY's products to improve reliability, performance, function or design.

ALPHI TECHNOLOGY does not assume any liability arising out of the application or use of any product or circuit described herein; nor does ALPHI TECHNOLOGY convey any license under its patent rights or the rights of others.

TABLE OF CONTENTS

1. DESCRIPTION

The IP-RESISTOR module from ALPHI TECHNOLOGY incorporates a network of 16 programmable resistor output channels that simulate resistive transducers. The output is always a true resistor. Each resistor can be shorted by a relay. The IP is designed to accommodate multiple resistor values or mixed configurations up to a total of 16 channels. A 1 Ohm resistor is in series at all time to prevent a short circuit.

The IP-RESISTOR meets the single-wide Industry Pack standard according to the INDUSTRY PACK VITA 4 Specifications.

Features:

- Single-size INDUSTRY PACK module.
- 1 to 128K Ohm Switch
- 0.1% precision resistor
- ½ watt rating
- Fast Switching
- Read back register
- Flash Memory for lookup table
- 2 relay per channel, Low relay contact resistance
- 8/32 Mhz clock
- 16 bit data
- Optional Extended temperature grade (-40°C to + 85°C)

2. BLOCK DIAGRAM

There are three basic section to the IP-RESISTOR

- The INDUSTRY PACK bus interface.
- The relay controllers
- 0.1% Precision resistor

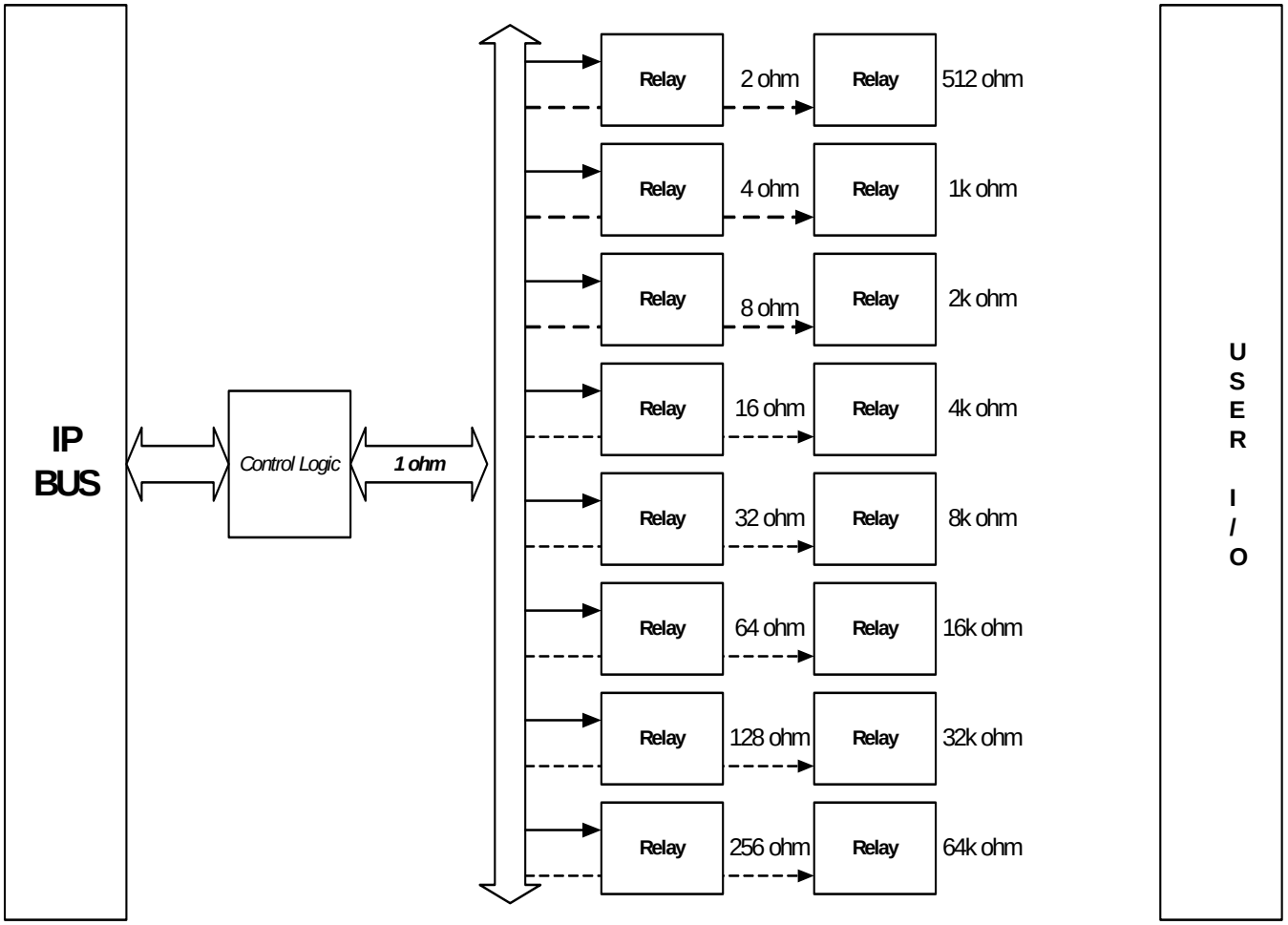

Figure 1: IP-RESISTOR Block Diagram

3. ATC-IP-RELAY SPACES

The following describe the different spaces used by the IP-RESISTOR.

- **ID** space INDUSTRY PACK identification codes
- **I/O** space IP-RESISTOR controllers registers access
- **Mem** space IP-RESISTOR on-board Flash access

The base address of these spaces depends on the specific INDUSTRY PACK carrier used.

3.1.1 ID SPACE

The identification space is defined as follows:

	Description	value
$01	Ascii "I"	$49
$03	Ascii "P"	$50
$05	Ascii "A"	$41
$07	Ascii "C"	$43
$09	Manufacturer identification	$11
$0B	Module type	$22
$0D	Revision module	$0A
$0F	Reserved $00	
$11	Software Driver #	low byte
$13	Software Driver #	high byte
$15	Number of bytes used in ID space	$0A
$17	CRC	
$19-3F	User available	

Correct reading of the first four bytes that contain the ASCII text "IPAC" can be used to identify the presence of an Industries Pack module.

Location $09 provide the Manufacturer identities (ALPHI TECHNOLOGY INDUSTRY PACKs $11).

The next two location identifies the module type and revision.

A 8-bit CHECKSUM (CRC) provide data integrity of the valid ID code set by the manufacturer.

The next bytes $ 19 to $ 3F are free for user data storage.

3.1.2 I/O SPACE

This region is where the access control for each resistor relay resides. By default (power on) each relay is open reading $0 writing a $1 to the corresponding bit will close the relay. Below is a chart to distinguish each relay channel and its corresponding bit. Write $0 opens relay, Write $1 close relay.

Note: Approximate read values are based on the reading of the resistor in series with the impedance of the relay. Values may slightly vary from board to board.

I/O SPACE $00

OFFSET HEX	VALUE HEX	ACTUAL RESISTOR VALUE	APPROXIMATE READ VALUE
$00	**$FFFF**	**1 Ohm**	**3.05 Ohm**
$00	**$FFFE**	**2 Ohm**	**5.19 Ohm**
$00	**$FFFD**	**4 Ohm**	**7.81 Ohm**
$00	**$FFFB**	**8 Ohm**	**10.57 Ohm**
$00	**$FFF7**	**16 Ohm**	**19.14 Ohm**
$00	**$FFEF**	**32 Ohm**	**35.27 Ohm**
$00	**$FFDF**	**64.2 Ohm**	**67.89 Ohm**
$00	**$FFBF**	**128 Ohm**	**129.82 Ohm**
$00	**$FF7F**	**258 Ohm**	**257.62 Ohm**
$00	**$FEFF**	**511 Ohm**	**513.38 Ohm**
$00	**$FDFF**	**1.02K Ohm**	**1.022K Ohm**
$00	**$FBFF**	**2.05K Ohm**	**2.06K Ohm**
$00	**$F7FF**	**4.07K Ohm**	**4.011K Ohm**
$00	**$EFFF**	**8.16K Ohm**	**8.187K Ohm**
$00	**$DFFF**	**16.4K Ohm**	**16.483K Ohm**
$00	**$BFFF**	**32.8K Ohm**	**33.286K Ohm**
$00	**$7FFF**	**65.7K Ohm**	**66.319K Ohm**

Table 1: Direct access registers

3.1.3 MEM SPACE

On the IP-Resistor is a 1 Megabit Flash device, its intention is to be used as a look-up table to record the values of each resistor read.. Included with the board is a program that allows you to read and write to the flash device, below are the instructions on how to use the program.

Note: At the end of the manual is a sample of the code used for the read/write access of the IP-RESISTOR flash device.

This demo program allows to access and program selected data in the first 1000 words of the FLASH.

burn Burn downloaded file into FLASH

This command copies the memory buffer to the FLASH.

read Read FLASH into memory

This command copies the content of the FLASH to the memory buffer.

info Display FLASH from info

This commands displays the manufacturer and product codes from the FLASH.

mmflash Access FLASH directly

This command accesses the memory buffer and allows to interactively read and modify it. The syntax is similar to MM.

mm Memory modify

This command is identical to the one in the processor board Bootrom loader.

bootrom Try to go back to the Bootrom loader.

Returns to the Bootrom loader.

help Help for specific commands

? Help for specific commands

4. ELECTRICAL DRAWING

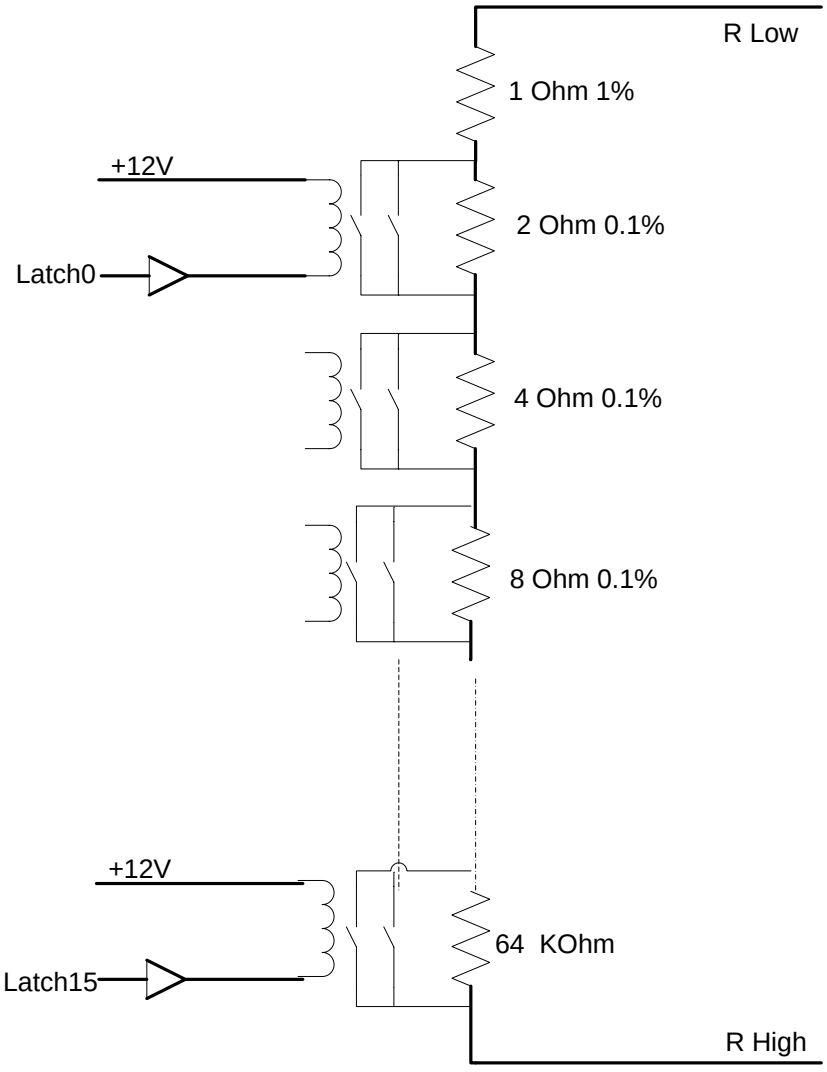

Figure 2: IP-RESISTOR Electrical Drawing

5. CONNECTOR

The following is the pin out for the IP-RESISTOR.

PIN	SIGNAL	PIN	SIGNAL
1		26	
2		27	
3		28	
4		29	
5		30	
6		31	
7	R LOW	32	R HIGH
8	GND	33	GND
9		34	
10		35	
11		36	
12		37	
13		38	
14		39	
15		40	
16		41	
17		42	
18		43	
19		44	
20		45	
21		46	
22		47	
23		48	
24		49	
25		50	

Table 3 - 50 pin connector

6. FLASH DEVICE SAMPLE PROGRAM

The follow is the sample code used to read/write the IP-RESISTOR flash device.

```
// Process command from serial port


#include "stdio.h"
#include "string.h"
#include <ctype.h>
#include <stdlib.h>
#include "stddef.h"
#include "alphi_io.h"
#include "serial.h"
#include "hardware.h"

void FlashWseq( void );

// @define:(alphi_io_impl) ATMEL_MFG_CODE | 0x1F | ATMEL Manufacturer Code
#define ATMEL_MFG_CODE  0x1F

// @define:(alphi_io_impl) AT29C010_DEVICE_CODE | 0xD5 | 1 Megabit device. (Could be AT29C010A.)
#define AT29C010_DEVICE_CODE    0xD5

// @define:(alphi_io_impl) AT29C020_DEVICE_CODE | 0xDA | 2 Megabit device.
#define AT29C020_DEVICE_CODE    0xDA

// @define:(alphi_io_impl) AT29C040_DEVICE_CODE | 0x5B | 4 Megabit device.
#define AT29C040_DEVICE_CODE    0x5B

// @define:(alphi_io_impl) AT29C040A_DEVICE_CODE | 0xA4 | 4 Megabit device.
#define AT29C040A_DEVICE_CODE   0xA4

// @define:(alphi_io_impl) AT29C010_SECTOR_SIZE | 128 | 1 Megabit device sector size. (Could be AT29C010A.)
#define AT29C010_SECTOR_SIZE    128

// @define:(alphi_io_impl) AT29C020_SECTOR_SIZE | 256 | 2 Megabit device sector size.
#define AT29C020_SECTOR_SIZE    256

// @define:(alphi_io_impl) AT29C040_SECTOR_SIZE | 512 | 4 Megabit device sector size.
#define AT29C040_SECTOR_SIZE    512
```



```
// @define:(alphi_io_impl) AT29C040A_SECTOR_SIZE | 256 | 4 Megabit device sector size.
#define AT29C040A_SECTOR_SIZE   256


#define BUFFER_LENGTH 1000
#define FLASH_START (volatile ubyte *)0x400000

ulong flashBuffer[BUFFER_LENGTH];               // local copy of the FLASH
unsigned int FlashSectorSize;

// Do we utilize serial output?
boolean fOutput = TRUE;

const char szHelpMemoryModify[] = "Usage:\n\n\tMM addr\n\nwhere addr is the hex"
" address to display and modify."
"\n'\\n' or '+' modifies the entry and shows the next location.\n'-' modifies the entry"
" and shows the previous location.\n' ' modifies the location and displays it again.\n"
"'.' exits.\nEntries are in Hexadecimal. If no entry, then nothing is written.\n";


void MemoryModify(
        char *szParm
)
{
   char buff[20];
   char *p;
        char c;
   volatile ulong *Addr, *startaddr;
        ulong temp;

        if (!szParm) {
                printf(szHelpMemoryModify);
                return;
                }
        startaddr = (ulong *)0;

        Addr = (ulong *)(startaddr + strtoul(szParm, &p, 16));

   printf("\n");

        while (1) {
```



```
next_location:
        temp = *Addr;
        printf("0x%06x : 0x%08x ", Addr, temp);
                p = buff;

                while (1) {
                        switch (c = getchar()) {
                                case '\n':
                                case '+':
                                case '-':
                                case ' ':
                                        if (c != '\n')
                                                putchar('\n');

                                        if (p != buff) {
                                                *p = '\0';
                                                temp = strtoul(buff, &p, 16);
                                                *Addr = temp;
                                                if (*Addr != temp)
                                                        printf("Location does not read
back correctly!\n");
                                                }

                                        if (c == '-')
                                                Addr--;
                                        else if (c == ' ')
                                                ;
                                        else
                                                Addr++;

                                        goto next_location;

                                case '0': case '1': case '2': case '3':
                                case '4': case '5': case '6': case '7':
                                case '8': case '9': case 'a': case 'b':
                                case 'c': case 'd': case 'e': case 'f':
                                case 'A': case 'B': case 'C': case 'D':
                                case 'E': case 'F':
                                        *p++ = c;
                                        break;

                                case '.':
                                        putchar('\n');
                                        return;

                                case '\b':
```



```
                                        if (p == buff) {
                                                // We are at the first char; beep and
restore prompt.
                                                putchar('\007');
                                                putchar(' ');
                                                }
                                        else {
                                                // backspace was already echoed, print
space and back up one.
                                                putchar(' ');
                                                putchar('\b');
                                                *p--;
                                                }
                                        break;

                                case '\033':
                                        putchar('\b');
                                        putchar(' ');
                                        putchar('\b');
                                        while (p != buff) {
                                                // send back, space, back
                                                putchar('\b');
                                                putchar(' ');
                                                putchar('\b');
                                                *p--;
                                                }
                                        break;

                                default:
                                        // beep and erase the errant character
                                        printf("\007\b \b");
                                        break;
                                };
                        }
                }
}

void FlashID(
        char *mfgcode,      // @parm Where to store the manufacturer code.
        char *devid         // @parm Where to store the device code.
)
{
   volatile ubyte *ptr, *dst;

      /* Send Product and Manufacturer sequences */
   dst = (volatile ubyte *)FLASH_START;
```



```
    ptr = (volatile ubyte *)(FLASH_START + 0x5555);   *ptr = 0xAA;
    ptr = (volatile ubyte *)(FLASH_START + 0x2AAA);   *ptr = 0x55;
    ptr = (volatile ubyte *)(FLASH_START + 0x5555);   *ptr = 0x90;

    delay(MILLI_SEC(20));

        /* Read MFG and DEV id info */
    *mfgcode = dst[0] & 0x000000FF;
    *devid   = dst[1] & 0x000000FF;

        /* Return to standard operating mode */
    ptr = (volatile ubyte *)(FLASH_START + 0x5555);   *ptr = 0xAA;
    ptr = (volatile ubyte *)(FLASH_START + 0x2AAA);   *ptr = 0x55;
    ptr = (volatile ubyte *)(FLASH_START + 0x5555);   *ptr = 0xF0;

    delay(MILLI_SEC(20));
}

const char szHelpReadFlash[] = "Usage:\n\n\tMM addr\n\nwhere addr is the hex"
" address to display and modify."
"\n'\\n' or '+' modifies the entry and shows the next location.\n'-' modifies the entry"
" and shows the previous location.\n' ' modifies the location and displays it again.\n"
"'.' exits.\nEntries are in Hexadecimal. If no entry, then nothing is written.\n";


void ReadFlash(
        char *szParm
)
{
    char buff[20];
    char *p;
        char c;
    volatile ulong *Addr, *startaddr;
        ulong temp;

        if (!szParm) {
                szParm = "0";
                }
        startaddr = (ulong *)flashBuffer;

        Addr = (ulong *)(startaddr + strtoul(szParm, &p, 16));

    printf("\n");
```



```
        while (1) {
next_location:
                if (Addr > (ulong *)((ulong)Addr + BUFFER_LENGTH))
                        Addr = (ulong *)startaddr;
        temp = *Addr;
        printf("0x%06x : 0x%08x ", Addr, temp);
                p = buff;

                while (1) {
                        switch (c = getchar()) {
                                case '\n':
                                case '+':
                                case '-':
                                case ' ':
                                        if (c != '\n')
                                                putchar('\n');

                                        if (p != buff) {
                                                *p = '\0';
                                                temp = strtoul(buff, &p, 16);
                                                *Addr = temp;
                                                if (*Addr != temp)
                                                        printf("Location does not read
back correctly!\n");
                                                }

                                        if (c == '-')
                                                Addr--;
                                        else if (c == ' ')
                                                ;
                                        else
                                                Addr++;

                                        goto next_location;

                                case '0': case '1': case '2': case '3':
                                case '4': case '5': case '6': case '7':
                                case '8': case '9': case 'a': case 'b':
                                case 'c': case 'd': case 'e': case 'f':
                                case 'A': case 'B': case 'C': case 'D':
                                case 'E': case 'F':
                                        *p++ = c;
                                        break;

                                case '.':
                                        putchar('\n');
```



```
                                return;

                        case '\b':
                                if (p == buff) {
                                        // We are at the first char; beep and
restore prompt.
                                        putchar('\007');
                                        putchar(' ');
                                        }
                                else {
                                        // backspace was already echoed, print
space and back up one.
                                        putchar(' ');
                                        putchar('\b');
                                        *p--;
                                        }
                                break;

                        case '\033':
                                putchar('\b');
                                putchar(' ');
                                putchar('\b');
                                while (p != buff) {
                                        // send back, space, back
                                        putchar('\b');
                                        putchar(' ');
                                        putchar('\b');
                                        *p--;
                                        }
                                break;

                        default:
                                // beep and erase the errant character
                                printf("\007\b \b");
                                break;
                        };
                }
        }
}

const char szHelpFlashRead[] = "Usage:\n\n\tREAD\n\nRead the content of the
FLASH in the memory buffer.\n";

// @func Write a entire buffer to the FLASH device, splitting bufer into
consecutive bytes.
// @comm Tracks progress or failure to the standard output.
```



```
// @rvalue 0 | Success.
// @rvalue -1 | Failure.
int WriteFlashSized(
        ulong *src,     // @parm Location in the user's buffer.
        ubyte *dst,     // @parm Location in the FLASH to write the buffer.
        ulong l,        // @parm Number of [bytes;words;longs] to transfer.
        boolean fOutput, // @parm Do we output to the console port?
        int size        // @parm 1, 2, or 4 for size of <p src>: [bytes;words;longs].
)
{
        ulong i;
        ulong Nbytes;
    char mfgcode;
    char devid;
    int cnt = 0;
        long len = l;

        printf("Reading %d words of data from 0x%06x, writing to 0x%06x, %d-
byte wide\n",l, src, dst, size);
        FlashID( &mfgcode, &devid );

        if (mfgcode != ATMEL_MFG_CODE) {
                if (fOutput)
                        printf("Don't know how to program this manufacturer's
device. (0x%x, 0x%x)\n", mfgcode, devid);
                FlashSectorSize = 0;
                }
        else {
                switch(devid) {
                        case AT29C010_DEVICE_CODE:
                                FlashSectorSize =   AT29C010_SECTOR_SIZE;
                                if (fOutput)
                                        printf("FLASH Device is a AT29C010\n");
                                break;

                        case AT29C020_DEVICE_CODE:
                                FlashSectorSize =   AT29C020_SECTOR_SIZE;
                                if (fOutput)
                                        printf("FLASH Device is a AT29C020\n");
                                break;

                        case AT29C040_DEVICE_CODE:
                                FlashSectorSize =   AT29C040_SECTOR_SIZE;
                                if (fOutput)
                                        printf("FLASH Device is a AT29C040\n");
                                break;
```



```
                case AT29C040A_DEVICE_CODE:
                        FlashSectorSize =   AT29C040A_SECTOR_SIZE;
                        if (fOutput)
                                printf("FLASH Device is a AT29C040A\n");
                        break;

                /* Unknown Flash Device Type */
                default:
                        if (fOutput)
                                printf("Don't know how to program this
manufacturer's device. (0x%x, 0x%x)\n", mfgcode, devid);
                        FlashSectorSize = 0;
                        break;
                }
        }

    if (FlashSectorSize == 0) {
        if (fOutput)
                printf("Unable to program device\n");
        return -1;
        }

    if (fOutput)
        printf("Programming Flash Image\n");

    while (len>size) {
        if (len >= FlashSectorSize)
                Nbytes = FlashSectorSize;
        else
                Nbytes = len;

        Nbytes /= size;

        FlashWseq();

        for (i = 0; i < Nbytes; i++)
                switch (size) {
                        case 1:
                         *dst++ = *src++;
                                break;
                        case 2:
                         *dst++ = *src & 0xff;
                         *dst++ = *src++ >> 8 & 0xff;
                                break;
                        case 4:
```

```
                        *dst++ = *src & 0xff;
                        *dst++ = *src >> 8 & 0xff;
                        *dst++ = *src >> 16 & 0xff;
                        *dst++ = *src >> 24 & 0xff;
                                *src++;
                                break;
                        }

            delay(MILLI_SEC(20));
                len -= Nbytes;

                if (fOutput) {
                        if (++cnt&0x3f)
                                fputc('.', stderr);
                        else
                                printf("%d\n", len);
                        }
                }

    delay(MILLI_SEC(20));

        if (fOutput)
                putchar('\n');
    return 0;
}

void ReadFlashChip(
        volatile ubyte *src,   // @parm Location in the FLASH to read to the buffer.
        volatile ulong *dst,   // @parm Location in the user's buffer.
        volatile ulong N,      // @parm Number of [bytes;words;longs] to transfer.
        int size         // @parm 1, 2, or 4 for size of <p dst>: [bytes;words;longs].
)
{
        unsigned long i;

        printf("Reading %d words of data from 0x%06x, writing to 0x%06x, %d-
byte wide\n",N, src, dst, size);
        for (i = 0; i < N; i++)
                switch (size) {
                        case 1:
                 *dst++ = (*src++ & 0xff);
                                break;
                        case 2:
                 *dst++ = (*src++ & 0xff) | (*src++ & 0xff) <<8;
                                break;
                        case 4:
```



```
                *dst++ = (src[0] & 0xff) | (src[1] & 0xff) << 8
                                | (src[2] & 0xff) << 16 | (src[3] & 0xff) << 24;
                        src += 4;
                        break;
                }
}

int WriteFlash(
        ubyte *src,     // @parm Location in the user's buffer.
        ubyte *dst,     // @parm Location in the FLASH to write the buffer.
        ulong N,        // @parm Number of bytes to transfer
        boolean fOutput      // @parm Do we output to the console port?
)
{
        return WriteFlashSized((ulong *)src, dst, N, fOutput, 4);
}

   /* Flash Write Sequence, Enable Flash for sector write */
void FlashWseq( void )
{
   volatile ubyte *ptr;

      /* Software Data Protection ENABLE Algorithm */
   ptr = (volatile ubyte *)(FLASH_START + 0x5555);   *ptr = 0xAA;
   ptr = (volatile ubyte *)(FLASH_START + 0x2AAA);   *ptr = 0x55;
   ptr = (volatile ubyte *)(FLASH_START + 0x5555);   *ptr = 0xA0;

}

void FlashRead(
        char *szParm
)
{
        ReadFlashChip(
                (ubyte *)FLASH_START,   // @parm Location in the FLASH to read
to the buffer.
                (ulong *)flashBuffer, // @parm Location in the user's buffer.
                BUFFER_LENGTH, // @parm Number of [bytes;words;longs] to
transfer.
                4       // @parm 1, 2, or 4 for size of <p dst>: [bytes;words;longs].
        );

}

const char szHelpFlashInfo[] = "Usage:\n\n\tINFO\n\nDisplay the manufacturing
code and the device code of the FLASH.\n";
```



```
void FlashInfo(
        char *szParm
)
{
        char mfgcode;       // Where to store the manufacturer code.
        char devid;         // Where to store the device code.

        printf ("Flash Base Address: 0x%06x\n", FLASH_START);
        FlashID(&mfgcode,&devid);

        if (mfgcode == ATMEL_MFG_CODE) printf ("The manufacturer is Atmel
(0x%02x)\n", mfgcode);
                else printf ("The manufacturer code is unknown (0x%02x)\n",
mfgcode );
        if (devid == AT29C010_DEVICE_CODE) printf ("Device code is correct for
the AT29C010 (0xd5)\n");
                else printf ("unknown code 0x%02x\n", devid );
}

const char szReadyToBurn[] = "Are you sure that you want to program the
FLASH memory? (y to continue)";
const char szAborted[] = "Burn aborted.\n";

const char szHelpFlashBurn[] = "Usage:\n\n\tBURN\n\nBurn the content of the
memory buffer into the FLASH.\n";

void FlashBurn(
        char *szParm
)
{
        char c;
        int status;

        printf(szReadyToBurn);
        c = getchar();
        putchar('\n');

        if (c != 'y' && c != 'Y') {
                printf(szAborted);
                return;
                }

        // write
        status = WriteFlash((ubyte *)flashBuffer, (ubyte *)FLASH_START,
        BUFFER_LENGTH, 4);
```



```
        if (status) printf ("Programming Failed...\n");
        else printf ("Programming succeeded!\n");
}

const char szUnknownCommand[] = "Unknown command. Try '?' for help.\n";

void Help(
        char *szParm
);

typedef void (*CommandFunction_t)(char *);

typedef struct {
        char *szCommand;
        CommandFunction_t Func;
        const char * szHelp;
        const char * szUsage;
} Command_t;

const char szHelpHelp[] = "Help for specific commands";
const char szHelpHelp2[] = "Usage:\n\n\tHELP cmd\n\nDetailed help for specific
command cmd";

Command_t Command[] =
{
        {"burn",            FlashBurn, "Burn downloaded file into FLASH",
szHelpFlashBurn},
        {"read",            FlashRead, "Read FLASH into memory",
szHelpFlashRead},
        {"info",        FlashInfo, "Display FLASH from info", szHelpFlashInfo},
        {"mm",                  MemoryModify, "Memory modify",
szHelpMemoryModify},
        {"mmflash",         ReadFlash, "Access FLASH directly",
szHelpReadFlash},
        {"bootrom",         (CommandFunction_t)0x1002, "Try to go back to the
bootrom", szHelpMemoryModify},
        {"help",            Help, szHelpHelp, szHelpHelp2},
        {"?",               Help, szHelpHelp, szHelpHelp2},
};

void Help(
        char *szParm
)
{
        int i;
```



```
        if (!szParm) {
                printf("The following commands are available.\nType ? cmd for
specific help.\n\n");

                for (i=0; i<table_size(Command); i++)
                        printf("%s\t%s\n",              Command[i].szCommand,
Command[i].szHelp);

                return;
                }

        for (i = 0; i < table_size(Command); i++) {
                if (!strcmp(szParm, Command[i].szCommand)) {
                                printf(Command[i].szUsage);
                                return;
                }
                }

        printf(szUnknownCommand);
}

void ProcessCommand(
        char *szCommand
)
{
        char *szTyped;
        int i;

        for (i=strlen(szCommand); i>=0; i--) {
                szCommand[i] = tolower(szCommand[i]);
                }

        szTyped = strtok(szCommand, " ");

        if (szTyped[0] == '\0')
                return;

        for (i = 0; i < table_size(Command); i++) {
                if (!strcmp(szTyped, Command[i].szCommand)) {
                                Command[i].Func(strtok(NULL, " "));
                                return;
                }
                }

        printf(szUnknownCommand);
}
```



```
main()
{
        char szCommand[40];
        char *p = szCommand;

        // Disable character caching and backspace/esc processing. Setting the
        // output buffer off also forces immediate flush.
        setbuf(stdin, 0);
        setbuf(stdout, 0);

        printf("Accessing IP A\n");
        *((int *) 0xf00010) = 1;

        printf("Flashtest>");
        while (1) {
           if (fOutput && ftell(stdin) >= 0) {
                        *p = getchar();

                        if (*p == '\b') {
                                if (p == szCommand) {
                                        // We are at the first char; beep and restore
prompt.
                                        putchar('\007');
                                        putchar('>');
                                        }
                                else {
                                        // backspace was already echoed, print space
and back up one.
                                        putchar(' ');
                                        putchar('\b');
                                        *p--;
                                        }

                                continue;
                                }

                        else if (*p == '\033') { // escape; kill line
                                putchar('\b');
                                putchar(' ');
                                putchar('\b');
                                while (p != szCommand) {
                                        // send back, space, back
                                        putchar('\b');
                                        putchar(' ');
                                        putchar('\b');
```



```
                                        *p--;
                                        }

                                continue;
                                }

                        else if (*p == '\n') {
                                *p = '\0';
                                if (szCommand[0] != '\0')
                                        ProcessCommand(szCommand);
                                p = szCommand;
                                printf("Flashtest>");
                                }
                        else
                                p++;
                        }

                }
}
```

