DATA SHEET / MPPC-C
Version 3 Data
Compression Software

hifn

Hi/fn[TM] supplies two of the Internet's most important raw materials: compression and encryption. Hi/fn is also the world's first company to put both on a single chip, creating a processor that performs compression and encryption at a faster speed than a conventional CPU alone could handle, and for much less than the cost of a Pentium or comparable processor.

As of October 1, 1998, our address is:

Hi/fn, Inc.
750 University Avenue
Los Gatos, CA 95032
info@hifn.com
http://www.hifn.com
Tel: 408-399-3500
Fax: 408-399-3501

Hi/fn Applications Support Hotline:
408-399-3544

Disclaimer

Table of Contents

Figures

THIS PAGE INTENTIONALLY BLANK

1 Product Description

The MPPC-C Data Compression Software Library provides a processor independent software implementation of the MPPC algorithm in a C source code format. The software is compatible with ANSI C.

This library supports the simultaneous use of multiple compression and decompression histories. Each history is completely independent of other histories. In addition, this software is re-entrant.

MPPC-C is fully compatible with Hi/fn's data compression processor chips that support the MPPC algorithm. Files compressed or decompressed with MPPC hardware or software may be compressed or decompressed interchangeably with MPPC hardware or software

Features
- MPPC compression format
- Multiple history support
- Windows® NT compatible

2 MPPC-C Files

The MPPC-C library is composed of several files. They are summarized below:

MPPC.H - This header file contains the function prototypes and constant definitions. This header file should be included in all source modules that access the MPPC-C library.

MPPCC.C - This source file contains the functions required for the compression operations.

MPPCD.C - This source file contains the functions required for the decompression operations.

3 Function Summary

Functions related to data compression are:

MPPC_SizeOfCompressionHistory - Returns amount of memory required for each compression history.

MPPC_InitCompressionHistory - Initializes a compression history.

MPPC_Compress - Compresses a block of data.

Functions related to data decompression are:

MPPC_SizeOfDecompressionHistory - Returns amount of memory required for each decompression history.

MPPC_InitDecompressionHistory - Initializes a decompression history.

_____

hi/fn

MPPC_Decompress - Decompresses a block of data.

4 Predefined Constants

In addition to the compile-time options described previously, the following constants are defined in the MPPC.H header file. See the function definitions for further information concerning these constants.

Constant	Value
MPPC_SAVE_HISTORY	0x04
MPPC_INTERNAL_DECOMPRESS	0x10
MPPC_MANDATORY_COMPRESS_FLAGS	0x01
MPPC_MANDATORY_DECOMPRESS_FLAGS	0x04
MPPC_INVALID	0x00
MPPC_SOURCE_EXHAUSTED	0x01
MPPC_DEST_EXHAUSTED	0x02
MPPC_FLUSHED	0x04
MPPC_RESTART_HISTORY	0x08
MPPC_EXPANDED	0x10
MPPC_SOURCE_MAX	8192

Note: The values listed are for this version of the software only. These values are listed here for information purposes only. These values may change in future versions. Do not write software that relies on a particular value of these constants.

Figure 1. Predefined constants

Note: All unused bits in function return values must be ignored. All unused bits in input parameters must be set to zero.

5 Performance

The data presented in this section was generated by compiling MPPC-C with all optimization switches turned on.

Figure 2 lists the approximate speed of compression and decompression over a range of processors. This performance is based on compressing a typical ASCII text file. In this case a text file containing the U.S. Constitution was used.
.

Processor	compress (Kbytes/s)	decompress (Kbytes/s)
80486DX2-66	350	1654
Pentium@ 150MHz	1125	5589

Figure 2. Typical speed

6 Hi/fn MPPC Compression

The MPPC compression algorithm compresses and decompresses data without sacrificing data integrity. The MPPC algorithm reduces the size of data by replacing redundant sequences of characters with tokens that represent those sequences. When the data is decompressed, the original sequences are substituted

_____

for the tokens in a manner that preserves the integrity of all data. MPPC differs significantly from "lossy" schemes, such as those used often for video images, which discard information that is deemed unnecessary.

The efficiency of data compression depends on the degree of redundancy within a given file. Although very high compression ratios are possible, an average compression ratio for mass storage applications is typically 2:1. For data communication applications, a compression ratio of 3:1 is more common.

7   Compression & Decompression Histories

This software requires a reserved block of memory in order to calculate and maintain compression information. This is referred to as a "history". The compression operation requires a compression history. The decompression operation requires a decompression history.

Some applications may want to maintain multiple compression and decompression histories. For example a data communications product may associate a different history for each data channel. This may be used to maximize the redundancy in each individual history, which in turn maximizes the compression ratio that is obtained.

7.1   History Maintenance

Before a history may be used for the first time, it must be initialized. This is accomplished using the MPPC_InitCompressionHistory or MPPC_InitDecompressionHistory commands.   This will place the history in a *start state*. A start state allows the history to be used when starting to process a new block of data. For multiple histories, each history must be initialized to the start state before it can be used for compression or decompression.

To properly finish compressing a block of data, a *flush* operation must be performed. A flush operation forces the compression algorithm to complete the compression of all the data it has read from the source. A flush operation guarantees that all the data read by the compression algorithm will be represented in the compressed data stream. A flush operation also places a compression history into a start state.

For this version of MPPC software, a flush operation must be performed on every call to the compression function.

8   MPPC_SizeOfCompressionHistory

unsigned short MPPC_SizeOfCompressionHistory(void);

This function must be called to determine the number of bytes required to be allocated for one compression history. If multiple compression histories are to be used, simply multiply the value returned by this function by the number of compression histories desired.
Note: For informational purposes only, the number of bytes required to be allocated for of each compression history is approximately 42 Kbytes. This is in-

_____

formational only, and subject to change. The MPPC_ SizeOfCompressionHistory function must be used to determine the actual byte count.

9　　　　　MPPC_InitCompressionHistory

```
unsigned short MPPC_InitCompressionHistory(
void *history                              /* Pointer to compression history */
);
```

This function must be called to initialize a compression history before it can be used with the MPPC_Compress function. Each compression history must be initialized separately.

If this function is called with a compression history that has been used previously, the history will be re-initialized to its beginning state. Any pending compression data within this compression history will be lost.

The *history parameter is a pointer to the memory allocated for a compression history. The size of this allocated memory was determined by the MPPC_SizeOfCompressionHistory function.

The return value will always be non-zero.

10　　　　　MPPC_Compress

```
unsigned short MPPC_Compress(
unsigned char * *source,        /* Pointer to pointer to source buffer */
unsigned char * *dest,          /* Pointer to pointer to destination buffer */
unsigned long *sourceCnt,       /* Pointer to source count */
unsigned long *destCnt,         /* Pointer to destination buffer size */
void *history                   /* Pointer to compression history */
unsigned short flags,           /* Special flags */
unsigned short performance      /* Performance parameter */
);
```

This function will compress data from the source buffer into the dest buffer. The function will stop when sourceCnt bytes have been read from the source buffer. A flush operation will take place after the source data has been processed.

sourceCnt will decrement and *source will increment for each byte that is read from the source buffer. destCnt will decrement and *dest will increment for each byte that is written to the dest buffer.

The valid range of sourceCnt is 0 through MPPC_SOURCE_MAX.

The destination buffer (allocation size and destCnt parameter) must be large enough to hold all the compressed data. To ensure that the destination buffer is large enough to accommodate the worst case expansion, the destCnt parameter must be equal to or greater than the following formula:

$(sourceCnt * 9/8) + 4$

If this function is called with an invalid value of sourceCnt or destCnt, the function will immediately terminate without performing any compression and the return value will be MPPC_INVALID.

If the data block expands during compression (meaning the number of bytes generated is greater than sourceCnt), then the MPPC_EXPANDED flag in the return value will be set when the function returns. In this case, the destination data should be discarded, and the compression history will be re-initialized automatically.

If the MPPC_SAVE_HISTORY bit of the flags parameter is set to zero, the Compression History will be cleared at the end of a flush operation. If this bit is set to one, the Compression History will NOT be cleared. This will allow a higher compression ratio for the next block to be compressed because it will continue to use the same history information.

Note: Blocks must be decompressed in the same order as they were compressed if the Compression History was not cleared between blocks during compression.

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
0	0	0	0	0	MPPC_SAVE_HISTORY	0	1

Figure 3. MPPC_Compress flags parameter

The performance parameter is not used for this version of software. The value passed here is ignored.

The return value will be MPPC_INVALID (zero) if the sourceCnt, destCnt, or flags calling parameters are invalid. The MPPC_EXPANDED bit in the return value will be set to one if the function has been terminated by data expansion. The MPPC_FLUSHED and MPPC_SOURCE_EXHAUSTED bits will be set if the compression operation was successful. The MPPC_RESTART_HISTORY bit is information required by the decompression function. The value of the MPPC_RESTART_HISTORY bit must be saved and passed to the flags parameter of the decompression function. If successful, the *source and *dest pointers, and sourceCnt, and destCnt values will be updated.

15	14	13	12	11	10	9	8
x	x	x	x	x	x	x	x
7	6	5	4	3	2	1	0
x	x	x	MPPC_ EXPANDED	MPPC_ RESTART_HISTORY	MPPC_ FLUSHED	0	MPPC_SOURCE_ EXHAUSTED

Figure 4. MPPC_Compress return value

Note: If the MPPC_EXPANDED bit is set to one, the InitDecompressionHistory function must be called for the corresponding decompression history.

> Read a block of data into the source buffer;
> returnCode = MPPC_Compress(&source, &dest, &sourceCnt, &destCnt,
>   compHistory, flags, performance);
> Write dest buffer to output device;

Figure 5. MPPC_Compress example pseudocode

The pseudocode in Figure 5 illustrates an example of how to call this function. For a more detailed example, please refer to the example software supplied with this release.

11   MPPC_SizeOfDecompressionHistory

unsigned short MPPC_SizeOfDecompressionHistory(void);

This function must be called to determine the number of bytes required to allocate for one decompression history. If multiple decompression histories are to be used, simply multiply the value returned by this function by the number of decompression histories desired.

Note: For informational purposes only, the number of bytes required to be allocated for each decompression history is approximately 8 Kbytes. This is informational only, and subject to change. The MPPC_ SizeOfDecompressionHistory function must be used to determine the actual byte count.

12   MPPC_InitDecompressionHistory

unsigned short MPPC_InitDecompressionHistory(
void *history   /* Pointer to decompression history */
);

This function must be called to initialize a decompression history before it can be used with the MPPC_Decompress function. In addition, this function must be called if the MPPC_InitCompressionHistory function was called prior to this compressed data being produced. This would occur if the MPPC_Compress function returned with the MPPC_EXPANDED bit in the return value set to one for the corresponding compression operation, or if the MPPC_Compress function was called with the MPPC_SAVE_HISTORY bit in the flags parameter set to zero. In either of these two cases the MPPC_InitDecompressionHistory function must be called prior to processing this compressed data.

The *history parameter is a pointer to the memory allocated for a decompression history. The size of this allocated memory was determined by the MPPC_SizeOfDecompressionHistory function.

The return value will always be non-zero.

_____

13 MPPC_Decompress

```
unsigned short MPPC_Decompress(
unsigned char * *source,          /* Pointer to pointer to source buffer */
unsigned char * *dest,            /* Pointer to pointer to destination buffer */
unsigned long *sourceCnt,         /* Pointer to source count */
unsigned long *destCnt,           /* Pointer to destination buffer size */
void *history                     /* Pointer to decompression history */
unsigned short flags              /* Special flags */
);
```

This function will decompress data from the source buffer into the dest buffer. The function will stop when sourceCnt bytes have been read from the source buffer.

sourceCnt will decrement and *source will increment when each byte is read from the source buffer. destCnt will decrement and *dest will increment when each byte is written to the dest buffer.

The valid range of destCnt is 0 through MPPC_SOURCE_MAX. The calling value of sourceCnt must be less than the calling value of destCnt. The value of destCnt must be equal to or greater than the actual number of raw data originally compressed. If this function is called with invalid values of sourceCnt or destCnt the function will immediately terminate without performing any compression and the return value will be MPPC_INVALID.

If the value of destCnt is too small, the function will terminate with the MPPC_DEST_EXHAUSTED bit set to one and the MPPC_DECOMP_OK bits set to zero.

The MPPC_RESTART bit in the flags parameter must contain the value returned in the MPPC_RESTART bit from the compress function.

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0

7	6	5	4	3	2	1	0
0	0	0	0	MPPC_ RESTART	1	0	0

Figure 6. MPPC_Decompress flag parameters

Note: Blocks must be decompressed in the same order as they were compressed if the Compression History has not been cleared between blocks during compression (i.e. the MPPC_SAVE_HISTORY bit was set during MPPC_Compress function calls).

The return value will be MPPC_INVALID (zero) if the sourceCnt, or flags calling parameters are invalid. If the decompression operation is successful, and the destination buffer does not become full, the MPPC_DECOMP_OK bits will be set to one. If successful, and the destination buffer is full (but does not overflow), The MPPC_DECOMP_OK and the MPPC_DEST_EXHAUSTED bits

_____

will be set to one. If the destination buffer overflows, only the
MPPC_DEST_EXHAUSTED bit will be set to one.

15	14	13	12	11	10	9	8
x	x	x	x	x	x	x	x
7	6	5	4	3	2	1	0
x	x	x	x	x	MPPC_ DECOMP_OK	MPPC_DEST_ EXHAUSTED	MPPC_ DECOMP_OK

Figure 7. MPPC_Decompress return value

If successful, the *source and *dest pointers and sourceCnt and destCnt will be
updated.

```
Read a block of data from an input device;
returnCode = MPPC_Deompress(&source, &dest, &sourceCnt, &destCnt,
        compHistory, flags, performance);
Write the destination buffer to packet memory;
```

Figure 8. MPPC_Compress example pseudocode

The pseudocode in Figure 8 illustrates an example of how to call this function.