To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

R8C Family, H8/300H Tiny Series and M16C/Tiny Series

M3S-TFS-Tiny: Original File System Software for Microcontrollers

Introduction

This document explains the usage of the TFS FileSystem software library along with a sample program.

Target Devices

Tiny microcomputers (R8C Family, H8/300H Tiny Series and M16C/Tiny Series)

Contents

1. Library specifications 3

2. Library type definitions 3

3. Explanation of terms 3
3.1 Logic sector / Logic Sector Number 3
3.2 Drive / Drive number 3

4. Library structures 4
4.1 tfs_volume – Volume structure 4
4.2 tfs_file – File structure 4
4.3 tfs_buff – Buffer structure 5
4.4 tfs_config – File system configuration 5
4.5 tfs_format_param – FAT16 parameters 5
4.6 tfs_format_param1 – File system format parameters 6
4.7 tfs_stat – File status 7
4.8 tfs_statfs – File system status 7

5. Library error codes 8

6. Library functions 9
6.1 tfs_init 9
6.2 tfs_exit 10
6.3 tfs_format1 11
6.4 tfs_attach 12
6.5 tfs_detach 13
6.6 tfs_alloci 14
6.7 tfs_openi 15
6.8 tfs_close 16
6.9 tfs_write 17
6.10 tfs_read 18
6.11 tfs_lseek 19
6.12 tfs_removei 20
6.13 tfs_stati 21
6.14 tfs_statfs 22
6.15 tfs_get_errno 23
6.16 tfs_get_date 24
6.17 tfs_get_time 25

7. Memory driver interface ... 26
7.1 Functions ... 26
7.1.1 tfs_write_lsec ... 26
7.1.2 tfs_read_lsec ... 27

8. Sample Program ... 28
8.1 Outline ... 28
8.2 Flow ... 29
8.3 Function list ... 30
8.4 Function chart ... 30
8.5 Folder composition in workspace ... 31

9. Sample software usage ... 32
9.1 Sample software execution ... 32
9.2 Real Time Clock ... 32
9.3 Sample Data for File Read / Write ... 32

10. Library Characteristics ... 33
10.1 Occupied memory size ... 33
10.2 Occupied stack size ... 33
10.3 Memory occupied by filesystem data structures ... 33
Website and Support ... 34

1. Library specifications

Following are the main specifications of the Tiny Filesystem library:

Specification	Value
Compatible media sizes	32 MB, 64 MB, 128 MB, 256 MB, 512 MB, 1 GB
FAT Wrapping FAT Type	FAT16
Multiple drive support	Yes (work area required to be set during initialization)
Directory	Root directory only
No. of directory entries	65,534 blocks maximum (set and save directory area size during formatting)
Directory entry size	128 byte fixed length
File designation	File number (file names cannot be used)
Number of files that can be opened simultaneously	Multiple (work area required to be set during initialization)
File size	Variable (allocated in blocks)
No. of blocks that can be allocated per file	4
Block size	Select block size from 8 KB, 16 KB, 32 KB, 64 KB, 128 KB or 256 KB while formatting
Block limit	65,534 blocks maximum
I/O buffer size	64 byte fixed length (logic sector)
Number of I/O buffers	At least 1

2. Library type definitions

This section gives details about the type definitions used in the library.

Datatype	Typedef
unsigned char	TFS_UCHAR
unsigned short	TFS_USHORT
unsigned long	TFS_ULONG

3. Explanation of terms

This section explains some of the terms related to the TFS library.

3.1 Logic sector / Logic Sector Number

The TFS reads/writes to the drive which is assumed to be divided into 64-byte fixed length blocks. This 64-byte fixed length block is called the logic sector. Each logic sector is identified with a logic sector number in ascending order starting from zero.

3.2 Drive / Drive number

The TFS is identified as a drive in which the FAT volume (similar to a DOS partition) is stored in the file system. If the TFS has more than one drive, the additional drives should be identified with numbers starting from 0. The drive number is this drive identification number.

4. Library structures

This section gives details of the structures used in the library.

4.1 tfs_volume – Volume structure

Explanation

This structure is used to hold the drive information. The number of structures required will be equal to the number of drives to be use. For instance, if the number of drives is 1, only one structure variable will be required; if the number of drives is 2, two structures will be required and so on.

The members of this structure should not be accessed directly from the user program. The user program should only declare a structure variable array with array size equal to the number of drives to be used.

Structure

Datatype	Structure element	Explanation
TFS_UCHAR	is_mounted	For TFS internal usage
TFS_UCHAR	drv	
TFS_USHORT	rootents	
TFS_USHORT	blocks	
TFS_USHORT	bsize	
TFS_ULONG	start	
TFS_ULONG	vsize	
TFS_ULONG	rsize	
TFS_ULONG	hsize	
TFS_ULONG	dsize	

4.2 tfs_file – File structure

Explanation

This structure is used to hold the file information. The number of structures required will be equal to the number of files to be opened simultaneously. For instance, if the number of files to be used at a time is only 1, only one structure variable will be required; if the number of files to be used at a time is 2, two structures will be required and so on.

The members of this structure should not be accessed directly from the user program. The user program should only declare a structure variable array with array size equal to the number of files to be used simultaneously.

Structure

Datatype	Structure element	Explanation
TFS_UCHAR	is_open	For TFS internal usage
TFS_UCHAR	id	
TFS_UCHAR	drv	
TFS_UCHAR	flags	
TFS_USHORT	ent	
TFS_ULONG	size	
TFS_ULONG	ptr	

4.3 tfs_buff – Buffer structure

Explanation

This structure is used to hold the logic sector buffer information.

The members of this structure should not be accessed directly from the user program. The user program should declare a buffer structure variable array with only one element. The number of array elements required is only one irrespective of the number of drives or files to be used.

Structure

Datatype	Structure element	Explanation
TFS_UCHAR	cnt	For TFS internal usage
TFS_UCHAR	drv	
TFS_ULONG	lsec	
TFS_UCHAR	buf[]	

4.4 tfs_config – File system configuration

Explanation

This structure is used to set the file system configuration as per the user's requirements. The user should initialize this structure with the desired values and then call the *tfs_init* function to set these values.

Structure

Datatype	Structure element	Explanation
TFS_USHORT	drives	Number of drives to be used (≥1)
TFS_USHORT	files	Number of file descriptors to be used i.e. no. of files to be opened simultaneously. (≥1)
TFS_USHORT	buffs	Number of logic sector buffers to be used (≥1)
struct tfs_volume*	volume	Start address of volume structure array. The number of array elements should be equal to the number of drives to be used.
struct tfs_file*	file	Start address for file structure array. The number of array elements should be equal to the number of files to be used.
struct tfs_buff*	buff	Start address for buffer structure array. It is sufficient to have only one element in this array.

4.5 tfs_format_param – FAT16 parameters

Explanation

This structure is a member of the *tfs_format_param1* structure. It holds the FAT16 parameters used while formatting the drive.

Structure

Datatype	Structure element	Explanation
TFS_ULONG	TotSec	Total number of sectors in the volume
TFS_USHORT	SecPerTrk	Number of sectors per track
TFS_USHORT	NumHeads	Total number of heads
const char*	VolLab	Volume label

4.6 tfs_format_param1 – File system format parameters

Explanation

This structure holds the formatting parameters for the memory drive.

Structure

Datatype	Structure element	Explanation
struct tfs_format_param	fat	FAT16 parameters (as explained in 4.5)
TFS_USHORT	rootents	Number of root directory entries
TFS_USHORT	bsize	Block size in KB

Members

fat.TotSec

Set the total number of sectors in the volume (512 bytes/sector).

fat.SecPerTrk

Set the number of sectors per track on the drive. (BIOS Parameter)

fat.NumHeads

Set the number of heads on the drive.

fat.VolLab

Set the FAT Volume label. Setting NULL will use the label “NO␣NAME␣␣␣” track on the drive.

rootents

Set the number of entries in the root directory. Set value which is an integral multiple of 4.

bsize

Set the data block size in kilobytes (KB). Valid values are 8, 16, 32, 64, 128 and 256.

4.7 tfs_stat – File status

Explanation

This structure holds the file information returned by the *tfs_stati* function.

Structure

Datatype	Structure element	Explanation
TFS_ULONG	st_size	File size
TFS_USHORT	st_mdate	Date when the file was last modified
TFS_USHORT	st_mtime	Time when the file was last modified
TFS_USHORT	st_mode	File mode

Members

st_size

Stores the size of file in bytes.

st_mdate

Stores the date when the file was modified.
bit15:9 - Year from 1980 (Value in the range of 0 to 127)
bit8:5 - Month (Value in the range 1 to 12)
bit4:0 - Day (Value in the range 1 to 31)

st_mtime

Stores the time when the file was modified or the directory was created.
bit15:9 - Hour (Value in the range 0 to 23)
bit8:5 - Minutes (Value in the range 0 to 59)
bit4:0 – Seconds are displayed in two second intervals. (Value in the range 0 to 29 and displayed as 0-58)

st_mode

File mode is used to indicate whether the file is a normal file or a directory.

4.8 tfs_statfs – File system status

Explanation

This structure holds the file system information returned by the *tfs_statfs* function.

Structure

Datatype	Structure element	Explanation
TFS_USHORT	f_bsize	Block size (in KB)
TFS_USHORT	f_blocks	Total number of blocks
TFS_USHORT	f_bfree	Number of free blocks available
TFS_USHORT	f_files	Total number of root directory entries
TFS_USHORT	f_ffree	Number of free directory entries

5. Library error codes

This section gives the significance of the macros corresponding to the error codes returned by the library functions.

Macro	Value	Significance
TFS_EPERM	1	Operation not permitted
TFS_ENOENT	2	No such file or directory
TFS_ESRCH	3	No such process
TFS_EINTR	4	Interrupted system call
TFS_EIO	5	I/O error
TFS_ENXIO	6	No such device or address
TFS_E2BIG	7	Argument list too long
TFS_EBADF	9	Bad file number
TFS_EAGAIN	11	Try again
TFS_ENOMEM	12	Out of memory
TFS_EACCES	13	Permission denied
TFS_EFAULT	14	Bad address
TFS_EBUSY	16	Device or resource busy
TFS_EEXIST	17	File exists
TFS_EXDEV	18	Cross-device link
TFS_ENODEV	19	No such device
TFS_ENOTDIR	20	Not a directory
TFS_EISDIR	21	Is a directory
TFS_EINVAL	22	Invalid argument
TFS_ENFILE	23	File table overflow
TFS_EMFILE	24	Too many open files
TFS_EFBIG	27	File too large
TFS_ENOSPC	28	No space left on device
TFS_EROFS	30	Read-only file system
TFS_ERANGE	34	Math result not representable
TFS_EDEADLK	35	Resource deadlock occurred
TFS_ENAMETOOLONG	36	File name too long
TFS_ENOLCK	37	No record locks available
TFS_ENOTEMPTY	39	Directory not empty
TFS_ETIMEDOUT	100	Operation timed out

6. Library functions

6.1 tfs_init

Prototype

```
int tfs_init (const struct tfs_config *config)
```

Explanation

This function initializes the TFS library with the configuration given by the structure *tfs_config*. This function must be called before calling any other library function.

Arguments

Argument	Type	Explanation
config	const struct tfs_config*	Initialize this structure with the desired values as explained in section 4.4

Return value

Type	Explanation
int	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
struct tfs_volume volume[1];
struct tfs_file file[1];
struct tfs_buff buff[1];
struct tfs_config conf = {
    1,            //No. of drives
    1,            //No. of file descriptors
    1,            //No. of buffers
    volume,       //Start address of volume array
    file,         //Start address of file descriptor array
    buff          //Start address of buffer array
};
int ret_val;
ret_val  = tfs_init( &conf );
```

6.2 tfs_exit

Prototype

```
int tfs_exit (unsigned short force)
```

Explanation

This function is the end processing of the library. However, this function can be called only when the drive is unmounted. If this function is called when the drive is mounted, it will result in an error.

Normally, value 0 is set to the argument *force*. If a value other than zero is set, the function will perform a force end. After this function is called, no other function can be called without initializing the library again (by calling the *tfs_init* function).

Arguments

Argument	Type	Explanation
Force	unsigned short	Set 0 to perform a normal end. Set any other value to perform a force end.

Return value

Type	Explanation
Int	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int ret_val;
// Other code before end processing
ret_val = tfs_exit(0);
```

6.3 tfs_format1

Prototype

```
int tfs_format1 (unsigned short drv, const struct tfs_format_param1 *param)
```

Explanation

This function formats the drive *drv* with the parameters set in the structure *param*.

The drive can be formatted only when the drive is unmounted. If this function is called when the drive is mounted, it will result in an error. Also during formatting, all the open files must be closed.

The formatting takes place in the following order:

- The entire volume is first formatted as a FAT16 file system.
- Next, the TFS area is saved as a single file in the FAT16 file system that was just created.
- Last, the internal TFS area is formatted and initialized.

Arguments

Argument	Type	Explanation
drv	unsigned short	Number of the drive to be formatted.
param	const struct tfs_format_param1*	Initialize this structure with the desired values as explained in section 4.5 and 4.6

Return value

Type	Explanation
int	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
const struct tfs_format_param1 test = {
{
  (unsigned long)64*1024*2,   /* Total no. of sectors (512B/sector) */
  63,                          /* Sectors per track */
  255,                         /* Number of heads */
  "TINYFS      "              /* Volume label */
},
64,                            /* No. of root directory entries */
128                            /* Size of data block (KB) */
};
int ret_val;
// Library initialization
ret_val = tfs_format1( 0, &test );
```

6.4 tfs_attach

Prototype

```
int tfs_attach(unsigned short drv)
```

Explanation

This function mounts the TFS volume on the drive number *drv* passed as argument.

Arguments

Argument	Type	Explanation
drv	unsigned short	Drive number on which the TFS volume is to be mounted

Return value

Type	Explanation
int	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int ret_val;
// Library initialization
ret_val = tfs_attach(0);
```

6.5 tfs_detach

Prototype

```
int tfs_detach(unsigned short drv, unsigned short force)
```

Explanation

This function unmounts the drive *drv* passed as argument. The drive cannot be unmounted if the drive is in use. The function returns an error if the drive is in use.

Normally, value 0 is set to the argument *force*. If a value other than zero is set, the function will perform a force unmount.

Arguments

Argument	Type	Explanation
drv	unsigned short	Drive number from which the TFS volume is to be unmounted
force	unsigned short	Set 0 to perform a normal end. Set any other value to perform a force end.

Return value

Type	Explanation
int	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int ret_val;
// Initialization
tfs_attach(0);
// Processing
ret_val = tfs_detach(0,0);
```

6.6 tfs_alloci

Prototype

```
int tfs_alloci(unsigned short drv, unsigned short did, unsigned short fid)
```

Explanation

This function returns the first available file number greater than *fid* on the drive *drv* passed as argument. When file number is to be retrieved from the top of the directory, set the *fid* value to 0. Value 0 (root directory) must be set to the directory number *did*.

Arguments

Argument	Type	Explanation
Drv	unsigned short	Drive number on which file is to be created
Did	unsigned short	Must be set to value 0 (root directory)
Fid	unsigned short	File number beyond which first available file number is to be searched for.

Return value

Type	Explanation
Int	Returns the available file number if function execution is successful. Return value TFS_NONUM if an error occurs.

Sample Usage

```
unsigned short file_no;
// Initialization and other processing
file_no = tfs_alloci(0,0,0);
```

6.7 tfs_openi

Prototype

```
int tfs_openi(unsigned short drv, unsigned short did, unsigned short fid,
int flags)
```

Explanation

This function opens the file *fid* on the drive *drv*. Value 0 (root directory) must be set to the directory number *did*. The file can be opened in different modes using logical OR combination of the *flags*.

Arguments

Argument	Type	Explanation
drv	unsigned short	Drive number on which file is to be opened
did	unsigned short	Must be set to value 0 (root directory)
fid	unsigned short	File number retrieved from *tfs_alloci* funtion
flags	int	The following values can be appointed to the flags: TFS_O_RDONLY – Open as read-only TFS_O_WRONLY – Open as write-only TFS_O_RDWR – Open as read / write TFS_O_CREAT – Create a new file if it is non-existent.

Return value

Type	Explanation
int	Returns the file descriptor if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int fd;
unsigned short file_no;
// Initialization
file_no = tfs_alloci(0,0,0);
fd = tfs_openi(0, 0, file_no, TFS_O_RDWR|TFS_O_CREAT);
```

6.8 tfs_close

Prototype

```
int tfs_close (int fd)
```

Explanation

This function closes the file associated with the file descriptor *fd*.

Arguments

Argument	Type	Explanation
fd	int	File descriptor associated with the file to be closed.

Return value

Type	Explanation
int	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int ret_val, fd;
unsigned short file_no;
// Initialization
file_no = tfs_alloci(0,0,0);
fd = tfs_openi(0, 0, file_no, TFS_O_RDWR|TFS_O_CREAT);
ret_val = tfs_close(fd);
```

6.9 tfs_write

Prototype

```
int tfs_write (int fd, const void *buf, unsigned long count)
```

Explanation

This function writes *count* bytes from the buffer *buf* to the file associated with the file descriptor *fd*.

Arguments

Argument	Type	Explanation
fd	int	File descriptor associated with the file in which data is to be written
buf	const void*	Pointer to the buffer containing the data to be written.
count	unsigned long	Number of bytes of data that is to be written.

Return value

Type	Explanation
int	Returns the actual number of bytes written if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int ret_val, fd;
unsigned short file_no;
// Initialization
fd = tfs_openi(0, 0, file_no, TFS_O_RDWR|TFS_O_CREAT);
ret_val = tfs_write(fd,"123456789",9);
tfs_close(fd);
```

6.10 tfs_read

Prototype

```
int tfs_read (int fd, void *buf, unsigned long count)
```

Explanation

This function reads *count* bytes of data from the file associated with the file descriptor *fd* into the buffer *buf*.

Arguments

Argument	Type	Explanation
fd	int	File descriptor associated with the file from which data is to be read.
buf	void*	Pointer to the buffer in which the read data is to be stored.
count	unsigned long	Number of bytes of data that is to be read.

Return value

Type	Explanation
int	Returns the actual number of bytes read if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int ret_val, fd;
unsigned short file_no;
// Initialization and other processing
fd = tfs_openi(0, 0, file_no, TFS_O_RDWR);
ret_val = tfs_read(fd,rw_buff,9);
```

6.11 tfs_lseek

Prototype

```
int tfs_lseek (int fd, long offset, int whence)
```

Explanation

This function moves the file pointer associated with the file descriptor *fd* by *offset* number of bytes from the position given by *whence*. The argument *whence* can take the following values:

Whence value	File pointer position
TFS_SEEK_SET	Start of the file
TFS_SEEK_CUR	Current file pointer position
TFS_SEEK_END	End of the file

Arguments

Argument	Type	Explanation
fd	int	File descriptor associated with the file.
offset	long	Number of bytes by which the file pointer is to be moved.
whence	int	Position from where file pointer is to be moved.

Return value

Type	Explanation
int	Returns the file pointer position if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int fd;
unsigned short file_no;
long fp;
// Initialization and other processing
fd = tfs_openi(0, 0, file_no, TFS_O_RDWR|TFS_O_CREAT);
fp = tfs_lseek(fd, 5,TFS_SEEK_SET);
```

6.12 tfs_removei

Prototype

```
int tfs_removei (unsigned short drv, unsigned short did, unsigned short fid)
```

Explanation

This function removes/deletes the file *fid* from the drive *drv*. Value 0 (root directory) must be set to the directory number *did*.

Arguments

Argument	Type	Explanation
drv	unsigned short	Drive number from which the file is to be deleted
did	unsigned short	Must be set to the value 0 (root directory)
fid	unsigned short	File number of the file to be deleted.

Return value

Type	Explanation
int	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int ret_val;
unsigned short file_no;
// Initialization and other processing
ret_val = tfs_removei(0,0,file_no);
```

6.13 tfs_stati

Prototype

```
int tfs_stati(unsigned short drv, unsigned short did, unsigned short fid,
struct tfs_stat *buf)
```

Explanation

This function retrieves the file information of file *fid* and stores it in the *tfs_stat* structure *buf*.

Arguments

Argument	Type	Explanation
drv	unsigned short	Drive number of the file.
did	unsigned short	Must be set to the value 0 (root directory).
fid	unsigned short	File whose information is to be retrieved.
buf	struct tfs_stat*	Return value received from the function consisting of the file information.

Return value

Type	Explanation
int	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
unsigned short file_no;
struct tfs_stat stat;
int ret_val;
// Initialization and other processing
ret_val = tfs_stati(0,0,file_no,&stat);
```

6.14 tfs_statfs

Prototype

```
int tfs_statfs (unsigned short drv, struct tfs_statfs *buf)
```

Explanation

This function retrieves the space availability information on the mounted volume.

Arguments

Argument	Type	Explanation
drv	unsigned short	Drive on which the volume is mounted
buf	struct tfs_statfs*	Return value received from the function consisting of the volume information.

Return value

Type	Explanation
int	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int ret_val;
struct tfs_statfs statfs;
// Initialization and other processing
ret_val = tfs_statfs(0,&statfs);
```

6.15 tfs_get_errno

Prototype

```
int tfs_get_errno (void)
```

Explanation

This function returns the error number corresponding to the immediately preceding library function. 0 is returned if the preceding library function execution was successful.

Arguments

None

Return value

Type	Explanation
int	TFS Library error number (as explained in Sec. 5)

Sample Usage

```
int err_code, fd;
// Initialization and other processing
tfs_write(fd,"123456789123456789123456789",27);
err_code = tfs_get_errno();   //Returns error code corresponding to tfs_write
```

6.16 tfs_get_date

Prototype

```
unsigned short tfs_get_date (void)
```

Explanation

This is a **user-defined function.** The library does not include the definition for this function. The user needs to implement this function based on the working environment. The implementation should be such that the function returns the current date in the format as explained in the Sec. 4.7.

Arguments

None

Return value

Type	Explanation
unsigned short	Current date in the format as given in Sec. 4.7

Sample Usage

Please refer to the sample software for a sample implementation of the *tfs_get_date* function.

6.17 tfs_get_time

Prototype

```
unsigned short tfs_get_time (void)
```

Explanation

This is a **user-defined function.** The library does not include the definition for this function. The user needs to implement this function based on the working environment. The implementation should be such that the function returns the current time in the format as explained in the Sec. 4.7.

Arguments

None

Return value

Type	Explanation
unsigned short	Current time in the format as given in Sec. 4.7

Sample Usage

Please refer to the sample software for a sample implementation of the *tfs_get_time* function.

7. Memory driver interface

This section explains the details of the memory driver interface functions. The prototype of these functions along with the processing necessary in the implementation of each function has been explained. The implementation of these functions should be written by the user such that they can be used in conjunction with the memory driver available with the user.

7.1 Functions

Drives used by TFS are single volume (DOS partition) compatible. Partition table information is concealed from the TFS, so if the partition table needs to be used, the driver must process it. The TFS library uses the drive as a 64-byte fixed length logic sector array, and requests I/O with in these logic sectors.

7.1.1 tfs_write_lsec

Prototype

```
int tfs_write_lsec (unsigned short drv, unsigned long lsec, const void *buf)
```

Explanation

This function should consist of the code to write data to the disk drive. The details about the data to be written are given by the arguments. This function writes data from the buffer *buf* to the volume (DOS partition suitable) logic sector given by *lsec* in the drive *drv*.

Arguments

Argument	Type	Explanation
drv	unsigned short	Drive on which the volume is mounted
lsec	unsigned long	Specifies the logic sector number.
buf	const void*	Pointer to the data to be written.

Return Value

Type	Explanation
int	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

7.1.2 tfs_read_lsec

Prototype

```
int tfs_read_lsec (unsigned short drv, unsigned long lsec, void *buf)
```

Explanation

This function should consist of the code to read data from the disk drive. The details about the data to be read are given by the arguments. This function reads data from the volume (DOS partition suitable) logic sector given by *lsec* in the drive *drv* into the buffer *buf*

Arguments

Argument	Type	Explanation
drv	unsigned short	Drive on which the volume is mounted
lsec	unsigned long	Specifies the logic sector number.
buf	void*	Pointer to the buffer to store the read data

Return Value

Type	Explanation
int	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

8. Sample Program

This section explains the sample program for Tiny FS library usage. The sample program is in the form of a HEW (High-Performance Embedded Workshop) workspace. Change the initialization of the microcomputer and its peripherals according to the system in use.

8.1 Outline

The sample program creates a text file, writes data to the file and then confirms the data that is actually written to the file.

When the program is run, a Tiny Filesystem volume is mounted on the external memory card. The memory card is connected to the RSK(*) by means of an external add-on board (**). A file is created on the memory card and text data of 2 KB is written to the file. The file is then closed. For confirmation of the data that is written, the file is opened again in the read mode. The entire contents of the file are read and they are compared with the write buffer data in the program. Whether the contents of the data are matching or not is indicated on the LEDs on board the RSK.

The data is defined in the header file data_file.h.

(*)RSK refers to

Renesas Starter Kit for R8C/25

Renesas Starter Kit for M16C Tiny/26A

Renesas Starter Kit for H8/36079

(**) The external add-on board has a slot for inserting the memory medium. The pins of the memory medium are connected to the appropriate pins of the RSK. This circuit board will not be included with the Renesas Solutions Kits that the user intends to buy and is not available from Renesas.

8.2 Flow

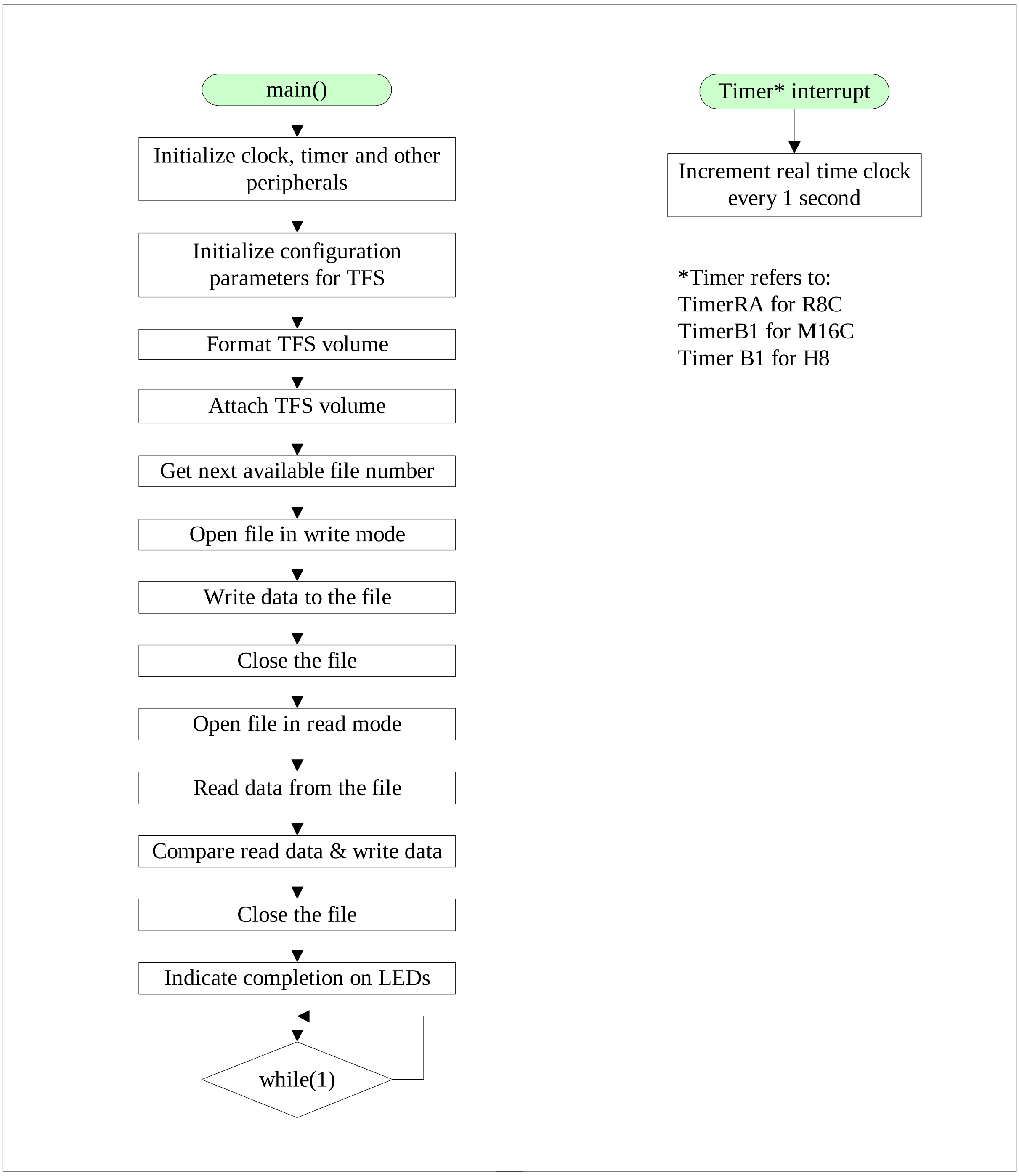

Figure 1: Flow of sample program

8.3 Function list

This following table gives a list of functions present in the sample program.

No.	Function name	Outline
1.0	main	Writes data to a file; reads and confirms the written data.
1.1	init_clock	The clock of the microcomputer and other clock related registers are initialized.
1.2	init_portpins	Initializes the port pins for peripherals
1.3	init_1sTimer	The timer is set up for Real Time Clock implementation.
1.4	error	Error handling function
1.5	mmc_drv_init	Memory driver initialization
1.6	tfs_init	Initializes the library configuration – Library function
1.7	tfs_format1	Formats the memory card – Library function
1.8	tfs_attach	Mounts the drive on TFS volume – Library function
1.9	tfs_alloci	Retrieves the next available file number – Library function
1.10	tfs_openi	Opens a file – Library function
1.11	tfs_write	Writes data to a file – Library function
1.12	tfs_read	Reads data from a file – Library function
1.13	tfs_close	Closes a file – Library function
1.14	tfs_detach	Unmounts the drive – Library function
1.15	tfs_exit	End processing for the library – Library function
2.0	timerRA_isr*	Increments the Real Time Clock every second.

* Function name for the sample software for R8C Family.
"INT_TimerB1" in the sample program of H8/Tiny. "timerB1_isr " in the sample program of M16C/Tiny.

8.4 Function chart

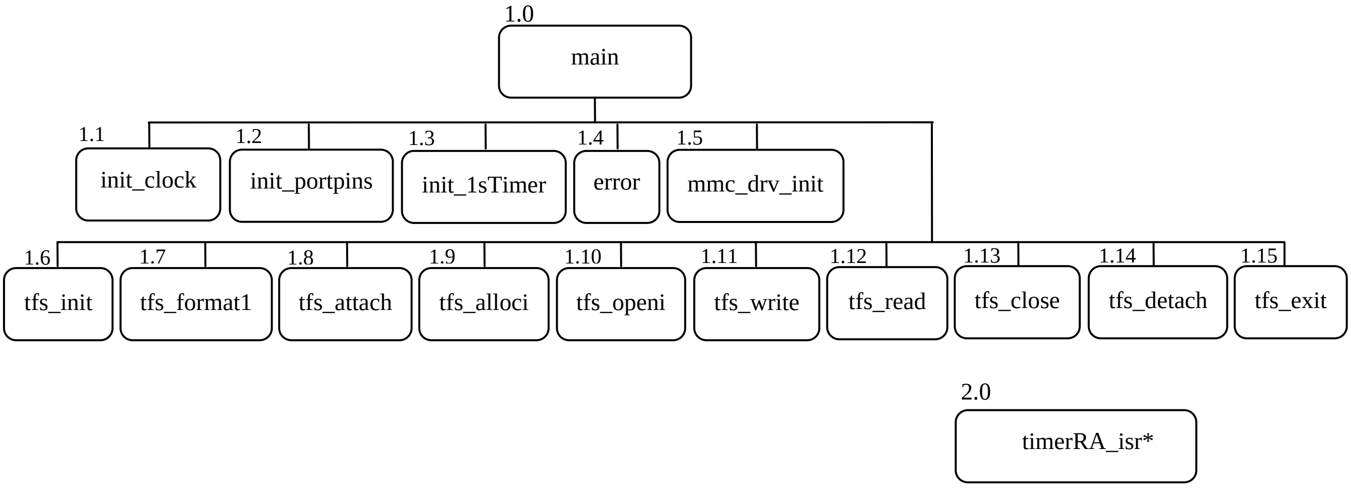

Figure 2: Function chart

(*) Function name for the sample software for R8C Family.
"INT_TimerB1" in the sample program of H8/Tiny. "timerB1_isr" in the sample program of M16C/Tiny.

8.5 Folder composition in workspace

```
tfs_sample_***                      Workspace directory (*** is microcomputer name)
|
|
|-- R5F21256(*)                     Project directory
        |
        |-- Debug                   Configuration directory
        |
        |-- Debug_***_E8_SYSTEM     Configuration directory (*** is microcomputer name)
        |
        |-- Debug_***_E8a_SYSTEM    Configuration directory (*** is microcomputer name)
        |
        |-- lib                     TFS FileSystem library storage directory
        |
        |-- Release                 Configuration directory
        |
        |-- src                     Sample source storage directory.
        |
        |-- hew_files               HEW auto-generated files storage directory.
```

(*) Project directory name for R8C. Following are the Project directory names for the other MCUs.

R5F21258 – R8CE

36077GF – H8N

36079GF – H8A

M30260F8AGP – M16C

9. Sample software usage

This section explains details related to sample software execution.

9.1 Sample software execution

- Build the sample software workspace and download the x30 file to the RSK.
- After the “Reset Go” button is clicked, the program starts running.
- First the file write operation takes place. A new text file is created on the memory card and 2 KB text data is written in it. The file is then closed.
- The same file is opened again in the read mode. The contents of the file are read and compared with the data that was passed while writing the file. This is done to confirm whether the data written to the file through the write function was actually written to the file as expected.
- The current state of the program is indicated by the LEDs on board the RSK.
- The following table gives the LED indications corresponding to program execution.

LED0	LED1	Significance
ON	OFF	Program running
ON	ON	Execution successful
OFF	ON	Error occurred

9.2 Real Time Clock

The sample software includes a real time clock implementation with the help of a timer. The timer is configured to generate an interrupt every second. In the corresponding Interrupt Service Routine, the current time and date are incremented. This time and date is used for some of the file manipulation operations. For details related to time and data storage, please refer to section 4.7

9.3 Sample Data for File Read / Write

The sample data for file read / write is stored in the header file data_file.h. The data is stored in an array of 2048 elements giving a total size of 2 KB (2048 Bytes). The data array consists of the text string “Renesas” written repeatedly. If required, the user can modify this array and the corresponding macro FILESIZE.

10. Library Characteristics

This section gives details about the memory consumption of the library.

10.1 Occupied memory size

Microcomputer	Mode/Option	ROM	RAM
R8C Family	R8C	8893	311
	R8CE	9418	321
H8/Tiny	Normal	6729	354
	Advanced	7561	432
M16C/Tiny	-	9418	321

Unit: Byte

10.2 Occupied stack size

Function	R8C	R8CE	H8 Normal	H8 Advanced	M16C
tfs_init	19	25	14	38	25
tfs_exit	16	24	12	34	24
tfs_format1	141	149	184	248	149
tfs_attach	61	61	90	108	61
tfs_detach	22	22	24	52	22
tfs_alloci	65	65	66	98	65
tfs_openi	86	88	74	118	88
tfs_close	50	52	50	80	52
tfs_write	106	110	102	178	110
tfs_read	106	108	104	186	108
tfs_lseek	7	7	24	28	7
tfs_removei	80	82	94	128	82
tfs_stati	52	52	50	84	52
tfs_statfs	64	64	50	80	64
tfs_get_errno	3	3	2	4	3

Unit:Byte

10.3 Memory occupied by filesystem data structures

Structure	Memory for one structure variable				
	R8C	R8CE	H8N	H8A	M16C
tfs_volume	28	28	28	28	28
tfs_file	14	14	14	14	14
tfs_buff	70	70	70	70	70
tfs_config	12	12	12	18	12
tfs_format_param1	14	16	14	16	16
tfs_stat	10	10	10	10	10
tfs_statfs	10	10	10	10	10

Unit:Byte

The table given above can be used to calculate the memory required for the different TFS library structure variables in the user’s application. Memory required for one structure variable multiplied by the number of variables will give the memory required for all variables of that particular structure.

Website and Support

Renesas Technology Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jun.30.09	—	First edition (M3S-TFS-Tiny Ver.1.01 Release00)

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.