RX Family

M3S-TFS-Tiny: Original File System Software

R20AN0039EJ0100
Rev.1.00
Oct.08.2010

Introduction

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

Target device

RX family

Contents

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 the details about the type definitions used in the library.

Datatype	Typedef
unsigned char	uint8_t
unsigned short	uint16_t
unsigned long	uint32_t
signed char	int8_t
signed short	int16_t
signed long	int32_t

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 the 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 the 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
uint8_t	is_mounted	For TFS internal usage
uint8_t	drv	
uint16_t	rootents	
uint16_t	blocks	
uint16_t	bsize	
uint32_t	start	
uint32_t	vsize	
uint32_t	rsize	
uint32_t	hsize	
uint32_t	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
uint8_t	is_open	For TFS internal usage
uint8_t	id	
uint8_t	drv	
uint8_t	flags	
uint16_t	ent	
uint32_t	size	
uint32_t	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
uint8_t	cnt	For TFS internal usage
uint8_t	drv	
uint32_t	lsec	
uint8_t	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
uint16_t	drives	Number of drives to be used(≥1)
uint16_t	files	Number of file descriptors to be used i.e. no. of files to be opened simultaneously.(≥1)
uint16_t	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
uint32_t	TotSec	Total number of sectors in the volume
uint16_t	SecPerTrk	Number of sectors per track
uint16_t	NumHeads	Total number of heads
const uint8_t*	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)
uint16_t	rootents	Number of root directory entries
uint16_t	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 `tfs_stati` function.

Structure

Datatype	Structure element	Explanation
uint32_t	st_size	File size
uint16_t	st_mdate	Date when the file was last modified
uint16_t	st_mtime	Time when the file was last modified
uint16_t	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 `tfs_statfs` function.

Structure

Datatype	Structure element	Explanation
uint16_t	f_bsize	Block size (in KB)
uint16_t	f_blocks	Total number of blocks
uint16_t	f_bfree	Number of free blocks available
uint16_t	f_files	Total number of root directory entries
uint16_t	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 R_tfs_init

Prototype

```
int16_t R_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
int16_t	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
};
int16_t ret_val;
ret_val  = R_tfs_init( &conf );
```

6.2 R_tfs_exit

Prototype

```
int16_t R_tfs_exit (uint16_t 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 `R_tfs_init` function).

Arguments

Argument	Type	Explanation
force	uint16_t	Set 0 to perform a normal end. Set any other value to perform a force end.

Return value

Type	Explanation
int16_t	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int16_t ret_val;
// Other code before end processing
ret_val = R_tfs_exit(0);
```

6.3 R_tfs_format1

Prototype

```
int16_t R_tfs_format1 (uint16_t 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	uint16_t	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
int16_t	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) */
};
int16_t ret_val;
// Library initialization
ret_val = R_tfs_format1( 0, &test );
```

6.4 R_tfs_attach

Prototype

```
int16_t R_tfs_attach(uint16_t drv)
```

Explanation

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

Arguments

Argument	Type	Explanation
drv	uint16_t	Drive number on which the TFS volume is to be mounted

Return value

Type	Explanation
int16_t	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int16_t ret_val;
// Library initialization
ret_val = R_tfs_attach(0);
```

6.5 R_tfs_detach

Prototype

```
int16_t R_tfs_detach(uint16_t drv, uint16_t 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	uint16_t	Drive number from which the TFS volume is to be unmounted
force	uint16_t	Set 0 to perform a normal end. Set any other value to perform a force end.

Return value

Type	Explanation
int16_t	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int16_t ret_val;
// Initialization
R_tfs_attach(0);
// Processing
ret_val = R_tfs_detach(0,0);
```

6.6 R_tfs_alloci

Prototype

```
uint16_t R_tfs_alloci(uint16_t drv, uint16_t did, uint16_t 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	uint16_t	Drive number on which file is to be created
did	uint16_t	Must be set to the value 0 (root directory)
fid	uint16_t	File number beyond which first available file number is to be searched for.

Return value

Type	Explanation
uint16_t	Returns the available file number if function execution is successful. Return value TFS_NONUM if an error occurs.

Sample Usage

```
uint16_t file_no;
// Initialization and other processing
file_no = R_tfs_alloci(0,0,0);
```

6.7 R_tfs_openi

Prototype

```
int16_t R_tfs_openi(uint16_t drv, uint16_t did, uint16_t fid, int16_t 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	uint16_t	Drive number on which file is to be opened
did	uint16_t	Must be set to the value 0 (root directory)
fid	uint16_t	File number retrieved from `R_tfs_alloci` funtion
flags	int16_t	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
int16_t	Returns the file descriptor if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int16_t fd;
uint16_t file_no;
// Initialization
file_no = R_tfs_alloci(0,0,0);
fd = R_tfs_openi(0, 0, file_no, TFS_O_RDWR|TFS_O_CREAT);
```

6.8 R_tfs_close

Prototype

```
int16_t R_tfs_close (int16_t fd)
```

Explanation

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

Arguments

Argument	Type	Explanation
fd	int16_t	File descriptor associated with the file to be closed.

Return value

Type	Explanation
int16_t	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int16_t ret_val, fd;
uint16_t file_no;
// Initialization
file_no = R_tfs_alloci(0,0,0);
fd = R_tfs_openi(0, 0, file_no, TFS_O_RDWR|TFS_O_CREAT);
ret_val = R_tfs_close(fd);
```

6.9 R_tfs_write

Prototype

```
int16_t R_tfs_write (int16_t fd, const void *buf, uint32_t 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	int16_t	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	uint32_t	Number of bytes of data that is to be written.

Return value

Type	Explanation
int16_t	Returns the actual number of bytes written if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int16_t ret_val, fd;
uint16_t file_no;
// Initialization
fd = R_tfs_openi(0, 0, file_no, TFS_O_RDWR|TFS_O_CREAT);
ret_val = R_tfs_write(fd,"123456789",9);
R_tfs_close(fd);
```

6.10 R_tfs_read

Prototype

```
int16_t R_tfs_read (int16_t fd, void *buf, uint32_t 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	int16_t	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	uint32_t	Number of bytes of data that is to be read.

Return value

Type	Explanation
int16_t	Returns the actual number of bytes read if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int16_t ret_val, fd;
uint16_t file_no;
// Initialization and other processing
fd = R_tfs_openi(0, 0, file_no, TFS_O_RDWR);
ret_val = R_tfs_read(fd,rw_buff,9);
```

6.11 R_tfs_lseek

Prototype

```
int16_t R_tfs_lseek (int16_t fd, int32_t offset, int16_t 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	int16_t	File descriptor associated with the file.
offset	int32_t	Number of bytes by which the file pointer is to be moved.
whence	int16_t	Position from where file pointer is to be moved.

Return value

Type	Explanation
int16_t	Returns the file pointer position if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int16_t fd;
uint16_t file_no;
int32_t fp;
// Initialization and other processing
fd = R_tfs_openi(0, 0, file_no, TFS_O_RDWR|TFS_O_CREAT);
fp = R_tfs_lseek(fd, 5,TFS_SEEK_SET);
```

6.12 R_tfs_removei

Prototype

```
int16_t R_tfs_removei (uint16_t drv, uint16_t did, uint16_t 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	uint16_t	Drive number from which the file is to be deleted
did	uint16_t	Must be set to the value 0 (root directory)
fid	uint16_t	File number of the file to be deleted.

Return value

Type	Explanation
int16_t	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int16_t ret_val;
uint16_t file_no;
// Initialization and other processing
ret_val = R_tfs_removei(0,0,file_no);
```

6.13 R_tfs_stati

Prototype

```
int16_t R_tfs_stati(uint16_t drv, uint16_t did, uint16_t fid, struct
tfs_stat *buf)
```

Explanation

This function retrieves the file information of file `fid` and stores it in the `R_tfs_stat` structure `buf`.

Arguments

Argument	Type	Explanation
drv	uint16_t	Drive number of the file.
did	uint16_t	Must be set to the value 0 (root directory).
fid	uint16_t	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
int16_t	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
uint16_t file_no;
struct tfs_stat stat;
int16_t ret_val;
// Initialization and other processing
ret_val = R_tfs_stati(0,0,file_no,&stat);
```

6.14 R_tfs_statfs

Prototype

```
int16_t R_tfs_statfs (uint16_t drv, struct R_tfs_statfs *buf)
```

Explanation

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

Arguments

Argument	Type	Explanation
drv	uint16_t	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
int16_t	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

Sample Usage

```
int16_t ret_val;
struct R_tfs_statfs statfs;
// Initialization and other processing
ret_val = R_tfs_statfs(0,&statfs);
```

6.15 R_tfs_get_errno

Prototype

```
int16_t R_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
int16_t	TFS Library error number (as explained in Sec. 5)

Sample Usage

```
int16_t err_code, fd;
// Initialization and other processing
R_tfs_write(fd,"123456789123456789123456789",27);
err_code = R_tfs_get_errno(); //Returns error code corresponding to
                              //R_tfs_write
```

6.16 R_tfs_get_date

Prototype

```
uint16_t R_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
uint16_t	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 `R_tfs_get_date` function.

6.17 R_tfs_get_time

Prototype

```
uint16_t R_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
uint16_t	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 `R_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 R_tfs_write_lsec

Prototype

```
int16_t R_tfs_write_lsec (uint16_t drv, uint32_t 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	uint16_t	Drive on which the volume is mounted
lsec	uint32_t	Specifies the logic sector number.
buf	const void*	Pointer to the data to be written.

Return Value

Type	Explanation
int16_t	Return value is 0 if function execution is successful. Return value is -1 if function ends with an error.

7.1.2 R_tfs_read_lsec

Prototype

```
int16_t R_tfs_read_lsec (uint16_t drv, uint32_t 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	uint16_t	Drive on which the volume is mounted
lsec	uint32_t	Specifies the logic sector number.
buf	void*	Pointer to the buffer to store the read data

Return Value

Type	Explanation
int16_t	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 RX610

(**) 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

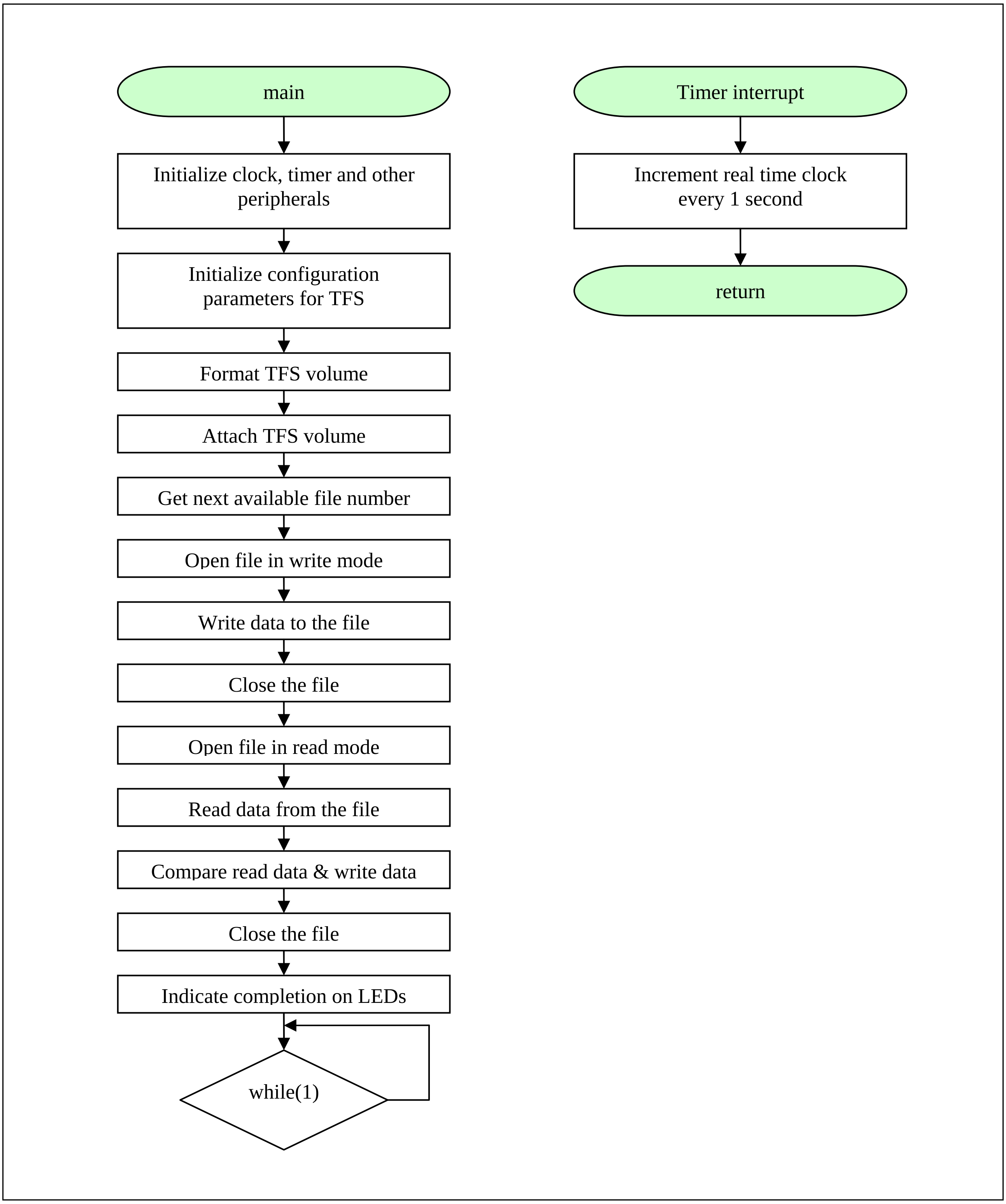

Figure 1: Flow of sample program

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	R_init_clock	The clock of the microcomputer and other clock related registers are initialized.
1.2	R_init_portpins	Initializes the port pins for peripherals
1.3	R_init_1sTimer	The timer is set up for Real Time Clock implementation.
1.4	R_error	Error handling function
1.5	R_mmc_drv_init	Memory driver initialization
1.6	R_tfs_init	Initializes the library configuration – Library function
1.7	R_tfs_format1	Formats the memory card – Library function
1.8	R_tfs_attach	Mounts the drive on TFS volume – Library function
1.9	R_tfs_alloci	Retrieves the next available file number – Library function
1.10	R_tfs_openi	Opens a file – Library function
1.11	R_tfs_write	Writes data to a file – Library function
1.12	R_tfs_read	Reads data from a file – Library function
1.13	R_tfs_close	Closes a file – Library function
1.14	R_tfs_detach	Unmounts the drive – Library function
1.15	R_tfs_exit	End processing for the library – Library function
2.0	R_int_timer_CMI0A	Increments the Real Time Clock every second.

8.4 Function chart

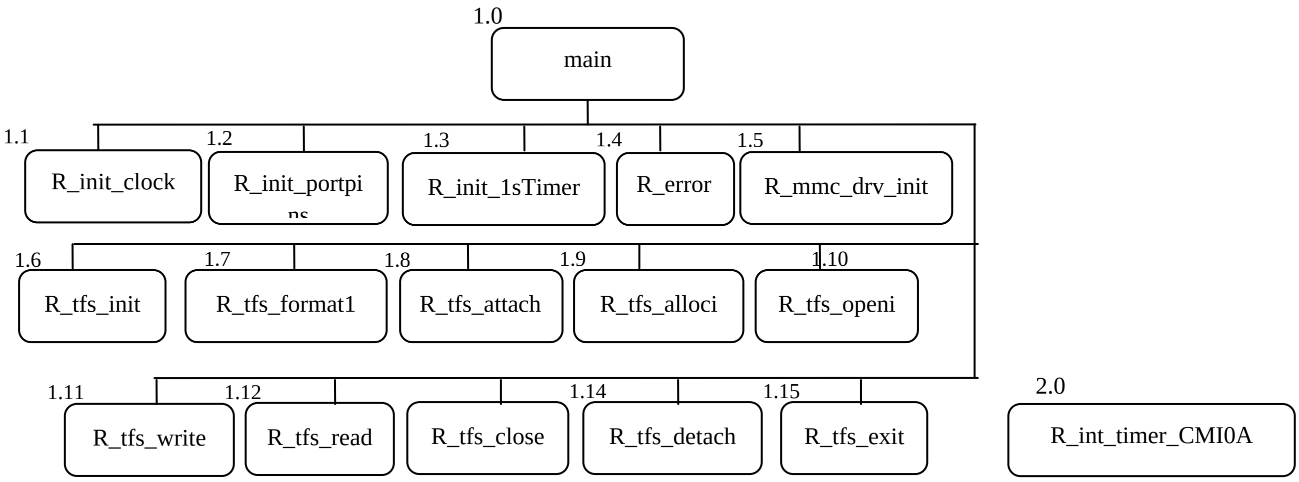

Figure 2: Function chart

8.5 Folder composition in workspace

```
tfs_sample_RX600                        Workspace directory
|
|
|-- R5F56108                            Project directory
      |
      |-- Debug                         Configuration directory
      |
      |-- Debug_RX600_E1_E20_SYSTEM     Configuration directory
      |
      |-- lib                           TFS FileSystem library storage directory
      |
      |-- Release                       Configuration directory
      |
      |-- sample                        Sample source storage directory.
      |
      |-- hew_files                     HEW auto-generated files storage directory.
```

9. Sample software usage

This section explains the details related to sample software execution.

9.1 Sample software execution

- Build the sample software workspace and download the abs file to the RSK.
- After the "Reset Go" button is clicked, 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.
- 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 sec 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	ROM	RAM
RX600	5649	158

Unit: Byte

10.2 Occupied stack size

Function	RX600
R_tfs_init	16
R_tfs_exit	8
R_tfs_format1	156
R_tfs_attach	60
R_tfs_detach	16
R_tfs_alloci	52
R_tfs_openi	88
R_tfclose	32
R_tfs_write	128
R_tfs_read	124
R_tfs_lseek	4
R_tfs_removei	72
R_tfs_stati	40
R_tfs_statfs	60
R_tfs_get_errno	4

Unit:Byte

10.3 Memory occupied by filesystem data structures

Structure	Memory for one structure variable
	RX600
tfs_volume	28
tfs_file	16
tfs_buff	72
tfs_config	12
tfs_format_param1	16
tfs_stat	12
tfs_statfs	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 Electronics Website
http://www.renesas.com/

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Oct.08.10	—	First edition issued

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan, R.O.C.
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141