V850E2/ML4

MultiMediaCard SPI Mode Device Driver Adoption Guide

R01AN1026EJ0101
Rev.1.01
Jul 23, 2012

Introduction

This application note describes the V850E2/ML4 MultiMediaCard device driver and its use.

Target Device

V850E2/ML4 (μPD70F4022)

Contents

1. Overview

1.1 Purpose

This application note presents an example of control in SPI mode of a MultiMediaCard (MMC) by the V850E2/ML4.

This application note presents an example of application creation.

1.2 Functional Overview

This device driver (MMC driver) is software that implements communication with an MMC on the V850E2/ML4.

This software has a FAT file system function.

Of the V850E2/ML4 built-in communication functions, this software uses the clock synchronous serial interface (CSIH) and implements MMC access in SPI mode.

MMC driver specifications:

- MMC System Specification, Version 3.x.
- Only SPI mode is used.
- This is a block-type device driver that uses a sector size of 512 bytes.
  The READ_MULTIPLE_BLOCK and WRITE_MULTIPLE_BLOCK commands are used.
  When the mounted card is not supported the multiple block access, this device driver uses the READ_SINGLE_BLOCK and WRITE_SINGLE_BLOCK commands.
- The driver does not depend on the OS.
- MMC driver described by this manual: V850E2M V.1.00 Release00

TFAT Library Specifications*:

- FAT types: FAT12 and FAT16
- Supports 8.3 type filenames (8-character file name plus 3-character extension)
- Number of drives: 1
- Sector size: 512 bytes
- The TFAT library is dedicated for the V850E2M core. It does not support other cores (e.g. V850 or V850ES).

Note: * The TFAT library may be updated. Download the latest version from the Renesas Electronics Corporation web site. (See related application note 1.)

1.3 File Structure

The figure below shows the directory and file structure.

Directory Structure, Directory Name/File Name	Notes
\doc <DIR>	**Document directory**
r01an1026ej1011_v850e2ml4.pdf	Application note (this document)
\source <DIR>	**MMC driver source program**
\common <DIR>	**Common functions directory**
r_mtl_com.c	Common functions (log recording)
r_mtl_com2.h	Common function definitions
r_mtl_endi.c	Common functions (endian related)
r_mtl_mem.c	Common functions (standard library functions)
r_mtl_str.c	Common functions (standard library functions)
r_mtl_tim.c	Common functions (software loop timer)
r_mtl_tim.h	Common functions (software loop timer definitions)
r_mtl_com.h	V850E2/ML4 operation definition header file
\mmc_driver <DIR>	**MMC device driver directory**
r_mmc.h	MMC driver common definitions
r_mmc_io.c r_mmc_io.h	MMC driver SPI mode I/O module
r_mmc_spi.c	MMC driver SPI mode communication module using SPI
r_mmc_mmc.c	MMC driver SPI mode MMC module
r_mmc_sub.c r_mmc_sub.h	MMC driver SPI mode • sub-module
r_mmc_usr.c	MMC driver SPI mode • API source program
\V850E2_ML4 <DIR>	**V850E2_ML4 SFR definitions for SPI directory**
r_mmc_sfr.h	MMC driver SFR definitions
r_mmc_user_config.h	MMC user definitions header file
\code <DIR>	**CSIH module directory**
CG_main.c	The main() function
CG_port.c CG_port.h	CSIH port initialization module
CG_serial.c CG_serial.h	CSIH operation control module
CG_serial_user.c	CSIH user settings module
CG_macrodriver.h	State control definitions header file
\tfat_sample <DIR>	**TFAT test sample program directory**
r_sample_tfat.c	MMC driver operation verification sample program
r_tfat_drv_if.c r_tfat_drv_if.h	TFAT driver source file
r_data_file.h	Write data definitions header file
r_target_io.h	Peripheral I/O definitions header file
\Library <DIR>	**Library directory**
r_tfat_lib.h	TFAT file system header file
libtfat_v850e2m.lib	MMC driver TFAT file system library
\sample <DIR>	**Sample project directory**
TFAT_MMC_sample_for_V850E2ML4.mtpj	Project file

2. Program Type Definitions

This program defines the following integer types.

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. Device Driver

3.1 Device Driver Functions Overview

Initialization Functions

Function	Functional Overview
R_mmc_Init_Driver()	MMC driver initialization

Device Manipulation Functions

Function	Functional Overview
R_mmc_Init_Slot()	MMC slot initialization
R_mmc_Detach()	MMC slot stop processing
R_mmc_Chk_Detect()	MMC insertion check

Data Access and Manipulation Functions

Function	Functional Overview
R_mmc_Read_Data()	Data read
R_mmc_Write_Data()	Data write
R_mmc_Get_MmcInfo()	MMC information obtaining

Internally Used Commands

Command Index	Command Name
CMD0	GO_IDLE_STATE
CMD1	SEND_OP_COND
CMD9	SEND_CSD
CMD10	SEND_CID
CMD12	STOP_TRANSMISSION
CMD13	SEND_STATUS
CMD17	READ_SINGLE_BLOCK
CMD18	READ_MULTIPLE_BLOCK
CMD24	WRITE_BLOCK
CMD25	WRITE_MULTIPLE_BLOCK
CMD58	READ_OCR
CMD59	CRC_ON_OFF

Note: Unsupported functions must be supported by the user.

3.2 Function Descriptions

3.2.1 MMC driver initialization (R_mmc_Init_Driver())

Item	Description
Prototype	void R_mmc_Init_Driver(void)
Arguments	None
Description	Initializes the MMC driver. Initializes the MMC control SFR and initializes the RAM and control ports for each slot. This function must be called once and only once at system startup.
Return value	None

3.2.2 MMC slot initialization (R_mmc_Init_Slot())

Item	Description
Prototype	int16_t R_mmc_Init_Slot(uint8_t SlotNo)
Arguments	uint8_t SlotNo: The slot number
Description	Initializes RAM and ports for the slot specified by the argument. It also performs MMC initialization. This function should be called when a card insertion is detected.
Return value	Returns the initialization result MMC_OK : Normal completion MMC_ERR_PARAM : Parameter error MMC_ERR_HARD : Hardware error MMC_ERR_CRC : CRC error MMC_ERR_IDLE : Idle state error MMC_ERR_OTHER : Other error

3.2.3 MMC slot stop processing (R_mmc_Detach())

Item	Description
Prototype	int16_t R_mmc_Detach(uint8_t SlotNo)
Arguments	uint8_t SlotNo: The slot number
Description	Performs the MMC removal processing for the specified slot. Performs MMC control SFR initialization, control port open, and control RAM initialization. This function should be called when a card removal is detected.
Return value	Returns the result of removal processing. MMC_OK : Normal completion MMC_ERR_PARAM : Parameter error

3.2.4 MMC insertion check (R_mmc_Chk_Detect())

Item	Description
Prototype	int16_t R_mmc_Chk_Detect(uint8_t SlotNo, uint8_t *pDetSts)
Arguments	uint8_t SlotNo : The slot number uint8_t *pDetSts : MMC insertion status storage buffer pointer
Description	Checks the MMC insertion state for the slot specified by the argument. When the return value is MMC_OK, the MMC insertion detection pin state will be stored in the MMC insertion status storage buffer (pDetSts). • MMC_TRUE: The MMC insertion detection pin is active. • MMC_FALSE: The MMC insertion detection pin is not active. Chattering exclusion is not performed in this processing. Chattering exclusion should be performed as needed at a higher level in the application. We recommend verifying media insertion by poling at a fixed period.
Return value	Returns the result of the check. MMC_OK : Normal completion MMC_ERR_PARAM : Parameter error

3.2.5 Data read (R_mmc_Read_Data())

Item	Description
Prototype	int16_t R_mmc_Read_Data(uint8_t SlotNo, uint32_t BlkNo, uint32_t BlkCnt, uint8_t *pData, uint8_t Mode)
Arguments	uint8_t SlotNo : The slot number uint32_t BlkNo : Read start block number uint32_t BlkCnt : Number of blocks to read uint8_t *pData : Read data storage buffer pointer uint8_t Mode : Read data transfer mode
Description	Reads data in block units (512 bytes) from the MMC. Reads the specified number of blocks of data starting with the specified block. The transfer mode (Mode) must be MMC_MODE_NORMAL (mode in which data is transferred to the data storage buffer argument (pData). Reading from the MMC is only possible when the card type (MmcInfo.Card) in the MMC information transferred from the function R_mmc_Get_MmcInfo() is not MMC_CARD_UNDETECT. The maximum block number is pMmcInfo.MaxBlkNum passed from the function R_mmc_Get_MmcInfo(). The maximum number of blocks is pMmcInfo.MaxBlkNum + 1.
Return value	Returns the result of the read operation. MMC_OK : Normal completion MMC_ERR_PARAM : Parameter error MMC_ERR_HARD : Hardware error MMC_ERR_CRC : CRC error MMC_ERR_OTHER : Other error

3.2.6 Data write (R_mmc_Write_Data())

Item	Description	
Prototype	int16_t R_mmc_Write_Data(uint8_t SlotNo, uint32_t BlkNo, uint32_t BlkCnt, uint8_t *pData, uint8_t Mode)	
Arguments	uint8_t SlotNo	: The slot number
	uint32_t BlkNo	: Write start block number
	uint32_t BlkCnt	: Number of blocks to write
	uint8_t *pData	: Write data storage buffer pointer
	uint8_t Mode	: Write data transfer mode
Description	Writes data in block units (512 bytes) to the MMC. Writes the specified number of blocks of data starting with the specified block. The transfer mode (Mode) must be MMC_MODE_NORMAL (mode in which data is transferred to the data storage buffer argument (pData). Writing to the MMC is only possible when the card type (MmcInfo.Card) in the MMC information transferred from the function R_mmc_Get_MmcInfo() is not MMC_CARD_UNDETECT. The maximum block number is pMmcInfo.MaxBlkNum passed from the function R_mmc_Get_MmcInfo(). The maximum number of blocks is pMmcInfo.MaxBlkNum + 1.	
Return value	Returns the MMC information acquisition result.	
	MMC_OK	: Normal completion
	MMC_ERR_PARAM	: Parameter error
	MMC_ERR_HARD	: Hardware error
	MMC_ERR_WP	: Write protection error
	MMC_ERR_OTHER	: Other error

3.2.7 MMC information obtaining (R_mmc_Get_MmcInfo())

Item	Description
Prototype	int16_t R_mmc_Get_MmcInfo(uint8_t SlotNo, MMC_INFO* pMmcInfo)
Arguments	uint8_t SlotNo : The slot number MMC_INFO* pMmcInfo : MMC information storage buffer pointer
Description	Returns the MMC information. Stores the MMC information in the MMC information buffer (pMmcInfo). pMmcInfo.Card : Card type MMC_CARD_UNDETECT : Card not detected MMC_CARD_MMC : MMC MMC_CARD_OTHER : Other card pMmcInfo.WProtect : Write protected state MMC_NO_PROTECT : Write protection released MMC_W_PROTECT_SOFT : Software write protection pMmcInfo.MemSize : Card capacity (in bytes) pMmcInfo.MaxBlkNum : Maximum block number (maximum block number for the media) If pMmcInfo.MemSize is 0xFFFFFFFF, 'pMmcInfo.MaxBlkNum' + 1 indicates the number of blocks the media holds (1 block = 512 bytes). If it is necessary to calculate the card's capacity, use the formula (pMmcInfo.MaxBlkNum + 1) × 512.
Return value	Returns the write result. MMC_OK : Normal completion MMC_ERR_PARAM : Parameter error MMC_ERR_OTHER : Other error

3.3 Data Structures

This section presents the data structures used.

MMC information structure definition

```
typedef struct {
  uint8_t Card;                 /* Card type                      */
  uint8_t WProtect;             /* Write-protection status        */
  uint32_t   MemSize;           /* Card capacity                  */
  uint32_t   MaxBlkNum;         /* The number of the max blocks   */
} MMC_INFO;                     /* total 12byte                   */
```

3.4 Macro Definitions

This section presents the macro definitions.

```
/*----- Definitions of return value -----*/
#define MMC_OK              (int16_t)( 0)       /* Successful operation     */
#define MMC_ERR_PARAM       (int16_t)(-1)       /* Parameter error          */
#define MMC_ERR_HARD        (int16_t)(-2)       /* Hardware error           */
#define MMC_ERR_CRC         (int16_t)(-3)       /* CRC error                */
#define MMC_ERR_WP          (int16_t)(-4)       /* Write-protection error   */
#define MMC_ERR_MBLKCMD     (int16_t)(-5)       /* Multi-block command error*/
#define MMC_ERR_IDLE        (int16_t)(-6)       /* Idle state error         */
#define MMC_ERR_OTHER       (int16_t)(-7)       /* Other error              */

/*----- Definitions of log type -----*/
#define MMC_LOG_ERR         1                   /* Log type : Error        */

/*----- Definitions of flag -----*/
#define MMC_TRUE            (uint8_t)0x01       /* Flag "ON"                */
#define MMC_FALSE           (uint8_t)0x00       /* Flag "OFF"               */

/*----- Definition of card type -----*/
#define MMC_CARD_UNDETECT   (uint8_t)0x00       /* Card is not found        */
#define MMC_CARD_MMC        (uint8_t)0x01       /* MMC                      */
#define MMC_CARD_OTHER      (uint8_t)0xFF       /* Other card               */

/*----- Definitions of write-protection status -----*/
#define MMC_NO_PROTECT      (uint8_t)0x00       /* None setting             */
#define MMC_W_PROTECT_SOFT  (uint8_t)0x02       /* Software write-protection*/
```

4. Settings Example

4.1 Common Function r_mtl_XXX() Variable Data Settings

The items in this section must be set to match the resources of each system.

The places that must be set are indicated by the comment "/**SET**/" in each file.

Selections are indicated within each file and detailed explanations are added.

4.1.1 r_mtl_com.h

This is the header file for the shared common functions.

Separate Copies of r_mtl_com.h for each microcontroller and for each system clock must be provided.

Include the appropriate version of r_mtl_com.h for the environment used.

The corresponding files must be provided by the user when different microcontrollers are used or if the system is operated using different clocks.

Microcontroller Used - System Clock	Include Directory (under the source directory)
V850E2/ML4 – 50 MHz	\common\V850E2_50MHz

1. Loop timer definition
   — If the software loop timer is used, include the file r_mtl_tim.h.
   This timer is mainly used for assuring the wait times required by device drivers.
   If the software loop timer is not used, comment out the include directive shown below.
   In the example below, the software loop time is used.
   When this driver is used, include this header file. Also, define the macro from r_mtl_tim.h that matches the system clock. If a V850E2 series microcontroller is operated at 50 MHz, define MTL_TIM_V850E2_50_0MHz.

```
/* When not using the loop timer, put the following 'include' as comments. */
#define MTL_TIM_V850E2__50_0MHz
#include "r_mtl_tim.h"
```

2. Endian type definition
   — Specify either little endian or big endian.

```
#if ( (defined(__LIT)) || (!defined(__BIG)) )
#define MTL_MCU_LITTLE              /* Little Endian           */ /** SET **/
#endif
```

3. Definition of the type of the standard library used
   — Define the type of the standard library used.
   If the processing shown below is used with the standard library, comment out the following macro definition.
   In the example shown below, the library provided with the compiler is not used.

```
/* Specify the type of user standard library.                  *//** SET **/
/* When using the compiler-bundled library for the following   *//** SET **/
/* processes, put the following 'define' as comments.          *//** SET **/
/* memcmp()/memmove()/memcpy()/memset()/strcat()/strcmp()/strcpy()/strlen()   *//** SET **/
#define MTL_USER_LIB             /* use optimized library      *//** SET **/
```

4.1.2 r_mtl_tim.h

If the loop timer is defined in r_mtl_com.h, it is included.

These values depend on the microcontroller used, the clock frequency, and the number of wait states.

The user must provide a macro appropriate for the environment if one is not present.

```
/* Define the counter value for the timer.                                  */
/* Specify according to the user MCU, clock and wait requirements.          */
/* Set the reference value to 10% more than the actual calculated value.    */
/*==========================================================================*/
/* Macro definitions                                                        */
/*==========================================================================*/
#ifdef MTL_TIM_V850E2__50_0MHz
/* Setting for 50.0MHz no wait (Compile Option "-optimize=2 -size")         */
#define MTL_T_250NS          2       /* loop times of 250ns     */ /** SET **/
#define MTL_T_500NS          5       /* loop times of 500ns     */ /** SET **/
#define MTL_T_1US           11       /* loop times of   1us     */ /** SET **/
#define MTL_T_2US           24       /* loop times of   2us     */ /** SET **/
#define MTL_T_4US           49       /* loop times of   4us     */ /** SET **/
#define MTL_T_5US           61       /* loop times of   5us     */ /** SET **/
#define MTL_T_10US         124       /* loop times of  10us     */ /** SET **/
#define MTL_T_20US         249       /* loop times of  20us     */ /** SET **/
#define MTL_T_30US         374       /* loop times of  30us     */ /** SET **/
#define MTL_T_50US         624       /* loop times of  50us     */ /** SET **/
#define MTL_T_100US       1249       /* loop times of 100us     */ /** SET **/
#define MTL_T_200US       2499       /* loop times of 200us     */ /** SET **/
#define MTL_T_300US       3749       /* loop times of 300us     */ /** SET **/
#define MTL_T_400US       4999       /* loop times of 400us     */ /** SET **/
#define MTL_T_1MS        12499       /* loop times of   1ms     */ /** SET **/
#endif
```

4.2 MMC Driver Variable Data Settings

The items in this section must be set to match the resources of each system.

The places that must be set are indicated by the comment "/**SET**/" in each file.

Selections are indicated within each file and detailed explanations are added.

4.2.1 r_mmc.h (driver common definitions)

1. Device count and device number definitions
   - Define the number of memory card slots.
   - Multiple devices are not supported since it is expected that only one MMC slot will be used with the V850E2/ML4.

```
/* Define number of required card slots. (1-N slots)                        */
/* Define slot number in accordance with the number of card slots to be connected.  */
/* Define number of slots (devices). */
#define MMC_SLOT_NUM          1              /* 1 slots        */ /** SET **/

/* Define slot number. */
#define MMC_SLOT0             0              /* Slot 0         */ /** SET **/
#define MMC_SLOT1             1              /* Slot 1         */ /** SET **/
```

2. Definitions for cards that do not support multiblock commands in SPI mode
   - For cards that do not support the MULTIPLE_BLOCK commands, the default setting is to support these cards by using the READ_SINGLE_BLOCK and WRITE_SINGLE_BLOCK commands. Therefore we recommend using these settings without change.

```
/*  When use the card which does not support a multi-block command, please define it.   */
/*  Use single block commands in the case of the card which does not support multiple block  */
/*  commands.                                                                           */
#define MMC_SBLK_CMD       /* Support single block commands    */ /** SET **/
```

3. Supported media definition
   - The MMC_SUPPORT_MMC macro must be defined.

```
/*  Please define the media to support.                                     */
#define MMC_SUPPORT_MMC                    /* MMC            */ /** SET **/
```

4.2.2 r_mmc_user_setting.h (Microcontroller-specific settings)

1. Selection of the microcontroller used
   The file r_mmc_user_setting.h depends on the microcontroller used.
   Include the mmc_user_setting.h file in the directory indicated below according to the environment used.
   If there is no file that matches the environment used, the user must create the appropriate file.

Microcontroller Used - Communications Environment	Include Directory (under the source directory)
V850E2/ML4	\mmc_driver\V850E2_ML4

2. Communication module channel number definition
   Define the MMC_CSIH_CHANNEL macro (if CSIH is used) to be the number of the communication module used.

   When the V850E2/ML4 - CSIH is used

```
/*----- CSIH Channel Select (CSIHx / x = 0 or 1) -----*/
#define MMC_CSIH_CHANNEL   1                      /** SET **/
```

3. Used control port definitions
   — Define these macros to match the DETECT signal (card detection) and CS signal (card select) used.

```
/*-----  Define the control port.-----*/
#define MMC_CS_PORT        P2          /* CS Port Setting */      /** SET **/
#define MMC_CS_BIT         0x0020U     /* CS Bit Select   */      /** SET **/

#define MMC_DETECT_PORT    PPR7(Note)  /* DETECT Port Setting */  /** SET **/
#define MMC_DETECT_BIT     0x0001U     /* DETECT Bit Select   */  /** SET **/
```

Note: This is a read-only register that is used when the I/O port is in input mode. See the V850E2/ML4 hardware manual for details.

4. Communication timeout detection processing definitions
   — The timeout detection processing used during communication may be omitted.
   If omitted, define the macro MMC_NOCHK_TIMEOUT. Defining this macro has the advantage that the processing speed is increased. The program may, however, stop if an error occurs in the communication function.
   If not omitted, set the timeout time.
   - The time units are set with MMC_T_SPI_WAIT. Select the macro to set from r_mtl_tim.h.
   - Define the timeout time for data transmission with the MMC_SPI_TX_WAIT (transmission) macro
   - Define the timeout time for data reception with the MMC_SPI_RX_WAIT (reception) macro
   - The set value of each timeout time macro will have the value (timeout time/units).

```
/*-----------------------------------------------------------------------*/
/* Macro "MMC_NOCHK_TIMEOUT" omits detecting timeout during communication.  */
/* If user omits detecting timeout, please define this macro.              */
/* If this macro is defined, processing speed would be increased.          */
/*-----------------------------------------------------------------------*/
#define MMC_NOCHK_TIMEOUT  /* No Check Communication Timeout   */ /** SET **/

/*-----------------------------------------------------------------------*/
/* If MMC_NOCHK_TIMEOUT would be not defined, please set timeout time.
/* MMC_T_SCI_WAIT is unit of measuring timeout.
/* Please select value from "r_mtl_tim.h"
/* Please set value of (timeout time/unit) to MMC_SCI_TX_WAIT(transmitting)
/* and MMC_SCI_RX_WAIT(receiving).
/*-----------------------------------------------------------------------*/
/* SCI transmit&receive completion waiting polling time        */ /** SET **/
#define MMC_T_SCI_WAIT     (uint32_t)MTL_T_100US
/* SCI transmission completion waiting time 50 * 1ms = 50ms    */ /** SET **/
#define MMC_SCI_TX_WAIT    (uint32_t)(MTL_T_1MS*50)
/* SCI receive completion waiting time 50 * 1ms = 50ms         */ /** SET **/
#define MMC_SCI_RX_WAIT    (uint32_t)(MTL_T_1MS*50)
```

5. Communication baud rate definitions
   — The user must define the communication baud rate.
   If the definitions for the SIO used are modified, SFR settings for each SIO register will be required. Refer to the microcontroller datasheet and set the SFRs appropriately for the system.
   In particular, it is necessary to set the communication speed settings so that the tODLY values from the card specifications for both Identification mode and Data Transfer mode are met.
   It is additionally necessary to set the parameters so that the conditions for tOD (100 kHz ≤ tOD ≤ 400 kHz) in Identification mode and tPP (0.1 MHz ≤ tPP ≤ 20 MHz(*)) in Data Transfer mode are met.
   Note that in this system tOD and tPP indicate the clock frequency.
   Since the SIO clock frequencies that can be supported differ with the microcontroller used, refer to the microcontroller's data sheet for these values.
   Set the MMC_UBRG_IDENTIFICATION macro to the clock setting in Identification mode.
   Set the MMC_UBRG_D_TRANSFER macro to the clock setting in Transfer mode.
   Set the MMC_CLK_D_TRANSFER macro to the clock frequency used in Transfer mode.
   In this program, these are set to 312.5 kHz for Identification mode and 12.5 MHz for Transfer mode.

```
/*-----------------------------------------------------------------------------------------*/
/* Define the value of the bit rate register according to a communication baud rate.      */
/* Set the frequency of CLK to 6MHz or less.                                               */
/* The possible maximum transfer frequency of CLK is depends on hardware circuit          */
/* and MCU conditions.                                                                     */
/* Refer to MCU hardware manual/memory card specifications and specify the buad rate.      */
/* When operating card with SPI mode,                                                      */
/* specify the following two definitions of Identification mode and Data Transfer mode.    */
/* Specify the definition to meet tODLY of both Identification mode and Data Transfer mode.*/
/* In addition, meet tOD (100kHz <= tOD <= 400kHz) at Identification mode                  */
/* and tPP (0.1MHz <= tPP <= 20MHz ) at Data Transfer mode.                                */
/* The maximum frequency depends on MCU type.                                              */
/*                                                                                         */
/* BRR = PCLK / (2m * B * 2)                                                               */
/* PCLK: Operating frequency [MHz]                                                         */
/* B   : Bit rate [bit/s]                                                                  */
/* m   : Determined by the SMR settings shown in the following table.                      */
/*                                                                                         */
/*-----------------------------------------------------------------------------------------*/
/* PCLK = 50MHz, n=0 */
#define MMC_UBRG_IDENTIFICATION (uint16_t)0x0050 /* BRR identification mode setting */ /** SET **/
                                      /* ++++-------- <= 400kHz                      */ /** SET **/
#define MMC_UBRG_D_TRANSFER     (uint16_t)0x0002 /* BRR data Transfer mode setting  */ /** SET **/
                                      /* ++++-------- <= 20MHz                       */ /** SET **/
#define MMC_CLK_D_TRANSFER      (uint32_t)600000 /* Data Transfer mode clock frequency */ /** SET **/
```

5. Microcontroller Connection and Used Microcontroller Resources

5.1 Used Microcontroller Resources

This program performs the following control operations.

It controls data I/O using a clock synchronous interface (CSIH, operated with the P bus clock).

When allocating the CSIH, allocate pins that support CMOS output to enable high-speed operation and set those pins to CMOS output.

For transmission control, the transmission buffer empty state is detected and the transmission interrupt request bit is used without using interrupts. Therefore the interrupt related setting must be set as shown below.

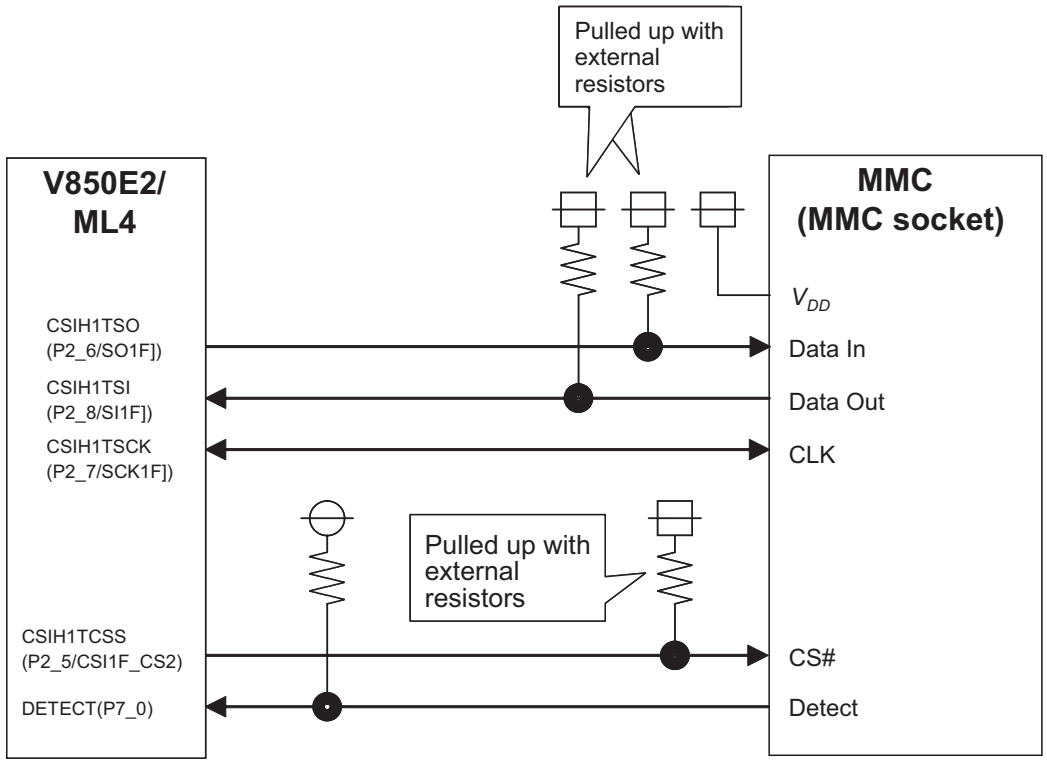
- Set the interrupt priority level to level 0 (interrupts disabled).
- Connect the MMC CS# pin to a microcontroller port and control it with microcontroller general-purpose port output.

Used Resources
CSIH module
Ports for CS# signals: 1 signal per card
Card detection ports: 1 signal per card
Power supply control ports: 1 signal per card

5.2 Connection with the Microcontroller

This section shows connection with the V850E2/ML4 microcontroller.

Similar connections are used with other microcontrollers.



A power supply control circuit is required for hot swapping control.
(This power supply control circuit is not shown in the figure.)
After card detection due to a card insertion, the application should provide power to the card.
Turn off power to the card before removing the card.
If multiple cards are used on the same SPI bus, provide a CS# pin port and a CardDetect port for each card.

6. Notes on Application Creation

6.1 Usage Notes

- When using this program, the software must be set up to match the hardware used.
- Only remove the memory card after setting the memory card to the inactive state, setting the signals between the microcontroller and the memory card to the high-impedance state, and stopping power supply to the device. The device may be damaged if it is removed while operating.
- If circuits that support hot swapping have not been implemented and a hot swap is performed, the power supply and other circuits may become unstable and the microcontroller may transition to the reset state.

6.2 Notes on Embedding

6.2.1 Development environment

[Development host]

Windows XP, Windows NT 4.0, Windows 200, Windows ME, Window 98, or Windows 95

The following Renesas development environment is used.

Use versions at least as recent as the ones listed below when developing a user application.

[Software Tools]

- Integrated development environment
  CubeSuite+ V1.00.01
- C compiler
  CubeSuite+ CX Compiler V1.20

[Debugging Tools]

- Emulator/debugger
  V850E2M E1 (JTAG)
- Emulator software
  V850E2 Debugger V1.00.00

[Evaluation board]

V850E2/ML4 CPU board (catalog number: R0K0F4022C000BR)

6.2.2 Included files

When an MMC driver is embedded, include the header files r_mtl_com.h and r_mmc.h.

The file r_mtl_com.h must be included first.

6.3 ROM, RAM, and Stack Sizes

The ROM, RAM, and stack sizes used by this program are listed below.

ROM and RAM Sizes

Classification (section name)	Size
ROM (.text .data)	About 30.8 KB
RAM (.sbss)	About 12.0 KB

Stack Sizes (MMC driver)

Function	Stack Size (bytes)
R_mmc_Init_Driver	16
R_mmc_Init_Slot	136
R_mmc_Detach	16
R_mmc_Chk_Detect	8
R_mmc_Read_Data	124
R_mmc_Write_Data	132

Stack Sizes (TFAT library)

Function	Stack Size (bytes)
R_tfat_f_mount	0
R_tfat_f_open	224
R_tfat_f_close	64
R_tfat_f_read	84
R_tfat_f_write	128
R_tfat_f_lseek	112
R_tfat_f_truncate	96
R_tfat_f_sync	56
R_tfat_f_opendir	148
R_tfat_f_readdir	80
R_tfat_f_getfree	100
R_tfat_f_stat	164
R_tfat_f_mkdir	228
R_tfat_f_unlink	172
R_tfat_f_chmod	164
R_tfat_f_utime	160
R_tfat_f_rename	236
R_tfat_f_forward	80

6.4 Notes on Card Insertion/Removal Detection

The R_mmc_Chk_Detect() function can be used to detect card insertion/removal using the card detection pin in the card connector. Therefore we recommend using poling at a fixed period to verify card insertion for card detection.

Also, if the reader goes to the card removed state during communication, the state will be seen as an abnormal response to a command and as a result, the driver will return an error.

Therefore one item that should be considered is the possibility that the card was removed during communication.

The driver may not return an error in the following cases.

- If the driver is unable to detect a card insertion/removal within the period and there is no response abnormality from the card, normal operation will continue.
- If a brief hot swap is performed during a write operation, the driver may incorrectly recognize a write completion. This is because the specifications are that the write busy signal clear is detected when the DataIn pin is high. (The DataIn pin is pulled up.)

Problems should be resolved with a method appropriate to the system by, for example, hardware interrupt based control or adjusting the poling period.

6.5 Notes on Port Allocation and Making Ports High-Impedance for Card Swapping

- For card insertion, set the card CS#, DataIn, DataOut, and CLK signals to the high-impedance state and then insert the card. After that, provide power to the card.
- For card removal, first stop power supply to the card, then set the card CS#, DataIn, DataOut, and CLK signals to the high-impedance state, and then remove the card.
- Although the card CS#, DataIn, DataOut, and CLK signals are allocated to microcontroller SIO and port pins, it is assumed that those ports may also be allocated to other resources. Therefore, this driver does not set these lines to the high-impedance state. Accordingly, these microcontroller pins must be set to the high-impedance state for card insertion and removal at a higher level in the application software.

7. TFAT Library

7.1 TFAT Library Function Overview

The table below lists the TFAT library API functions used in this program.

File Manipulation Functions

Function	Overview
R_tfat_disk_initialize ()	External media initialization function
R_tfat_f_mount ()	FAT file system working area register function
R_tfat_f_open ()	File open function
R_tfat_f_read ()	File data read function
R_tfat_f_write ()	File data write function
R_tfat_f_close ()	File close function
R_tfat_f_lseek ()	File read/write pointer move function

7.2 Function Descriptions

7.2.1 Media initialization (R_tfat_disk_initialize())

Item	Description
Prototype	DSTATUS R_tfat_disk_initialize (void)
Arguments	None
Description	Initializes a MultiMediaCard (MMC). This function calls the MMC driver MMC insertion check function R_mmc_Chk_Detect() and the MMC slot initialization function R_mmc_Init_Slot(). This function may be called as many times as needed until it returns normally.
Return value	Returns the initialization result. TFAT_RES_OK : Initialization completed normally TFAT_STA_NODISK : No MMC inserted TFAT_STA_NOINIT : Uninitialized state

7.2.2 FAT file system working area register (R_tfat_f_mount())

Item	Description
Prototype	FRESULT R_tfat_f_mount (uint8_t drv, FATFS *fs)
Arguments	uint8_t drv : Logical drive number FATFS *fs : Pointer to working area
Description	Registers a FAT file system working area on the MMC. If there are any open files or directories on the target drive, those all become invalid. This function always completes normally regardless of the state of the drive
Return value	Returns the registration state. TFAT_FR_OK : Normal completion

7.2.3 File open (R_tfat_f_open())

Item	Description
Prototype	FRESULT R_tfat_f_open (FIL *fp, const uint8_t *path, uint8_t mode)
Arguments	FIL *fp : Pointer to a structure object const uint8_t *path : Pointer to a character string that specifies the file to open uint8_t mode : Access/open mode flag
Description	Opens an existing file or creates a new file. If this function succeeds, a file object is created and can then be used to access the file. Before starting a file access, it is necessary to provide a FAT file system working area, and after that initialization, all file functions will use that area for that drive.
Return value	Returns the result TFAT_FR_OK : Normal completion TFAT_FR_NO_FILE : File not found TFAT_FR_EXIST : Object with same name exists TFAT_FR_DENIED : Illegal object manipulation TFAT_FR_RW_ERROR: Read or write error

7.2.4 File data read (R_tfat_f_read())

Item	Description
Prototype	FRESULT R_tfat_f_read (FIL *fp, void *buff, uint16_t btr, uint16_t *br)
Arguments	FIL *fp : Pointer to a structure object void *buff : Pointer to a data storage buffer uint16_t btr : Number of bytes to read uint16_t *br : Pointer to byte count storage variable
Description	Reads data from a file. This function calls the MMC driver function R_tfat_disc_read(). The read start position is the current read/write pointer. The read/write pointer is advanced by just the number of bytes read. We recommend checking *br after this function completes normally. If *br is smaller than btr, it indicates that end of file was reached during the file read operation.
Return value	Returns the result TFAT_FR_OK : Normal completion TFAT_FR_DENIED : Illegal object manipulation TFAT_FR_RW_ERROR: Read or write error TFAT_FR_NOT_READY: Drive initialization failure

7.2.5 File write (R_tfat_f_write())

Item	Description
Prototype	FRESULT R_tfat_f_write (FIL *fp, const void *buff, uint16_t btw, uint16_t *bw)
Arguments	FIL *fp : Pointer to a structure object const void *buff : Pointer to a data storage buffer uint16_t btw : Number of bytes to write uint16_t *bw : Pointer to byte count storage variable
Description	Writes data to a file. This function calls the MMC driver function R_tfat_disc_write(). The write start position is the current read/write pointer. The read/write pointer is advanced by just the number of bytes written. We recommend verifying *bw to determine whether or not the requested number of bytes were written after this function completes normally. If *bw is smaller than btw, it indicates that the MMC was full.
Return value	Returns the result TFAT_FR_OK : Normal completion TFAT_FR_DENIED : Illegal object manipulation TFAT_FR_RW_ERROR: Read or write error

7.2.6 File close (R_tfat_f_close())

Item	Description
Prototype	FRESULT R_tfat_f_close(FIL *fp)
Arguments	FIL *fp : Pointer to a structure object
Description	Closes an open file. For files to which something has been written, the cached state (data in the read/write buffers, the modified FAT) is returned to the disk. When this function completes normally, the file object will be invalid and its memory can be released.
Return value	Returns the result TFAT_FR_OK : Normal completion

7.2.7 Move file read/write pointer (R_tfat_f_lseek())

Item	Description
Prototype	FRESULT R_tfat_f_lseek(FIL *fp, uint32_t ofs)
Arguments	FIL *fp : Pointer to a structure object Uint32_t ofs : Offset to pointer move target
Description	Moves the file read/write pointer (the offset to the byte to be read/written next). The offset origin is the start of file. In write mode, if a value larger than the file size is specified, the file size is expanded to that size and the data in the expanded part of the file will be undefined. When it is desirable to write data quickly with no delay, we recommend using this function to increase the file size to the required size in advance.
Return value	Returns the result TFAT_FR_OK : Normal completion TFAT_FR_RW_ERROR: Read or write error

8. Sample Program

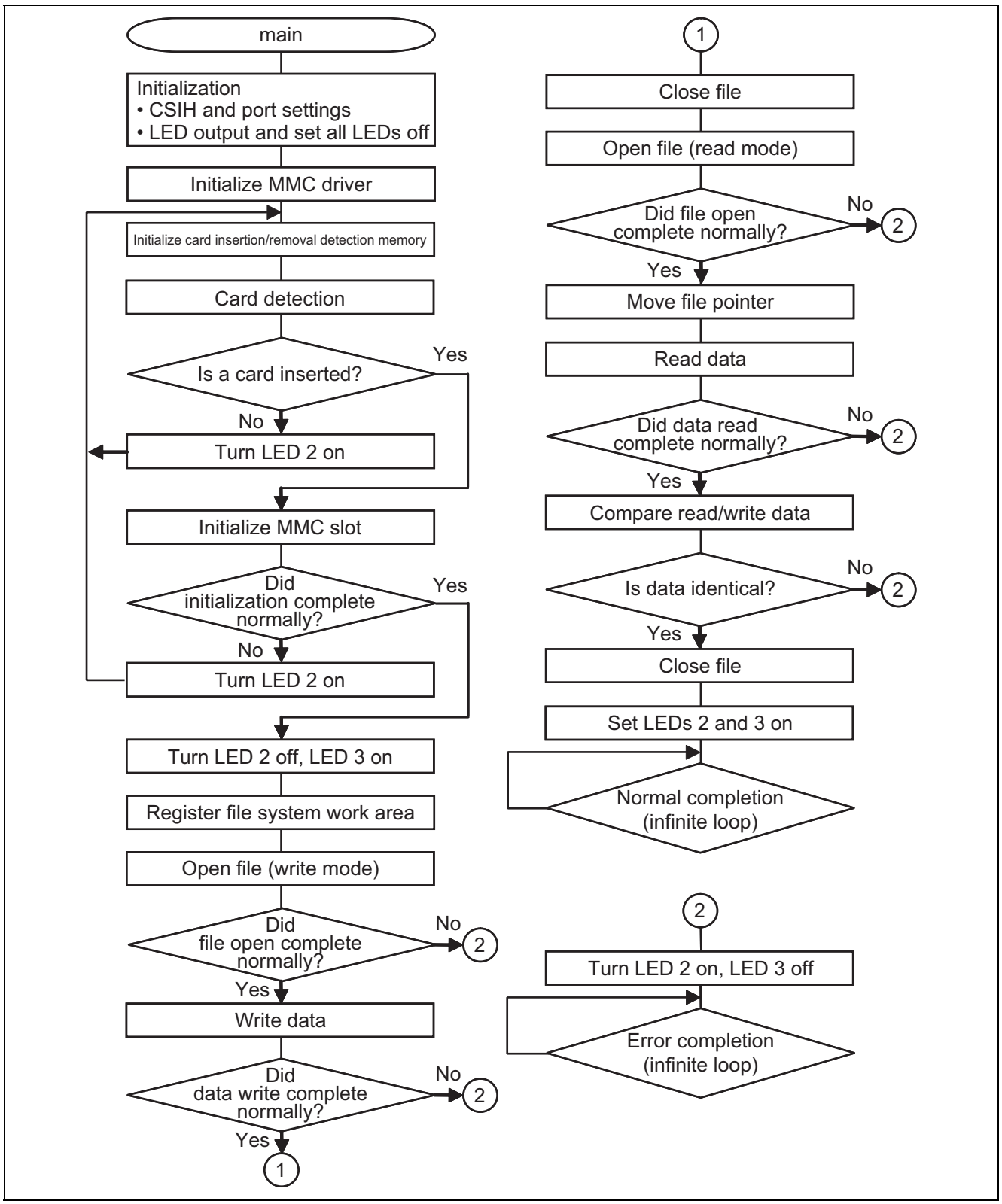
This section describes the sample program.

8.1 Processing

- When an MMC is inserted, this program creates a file on the MMC and writes 2 KB of text data. To verify the written data, it reads the whole contents of the file and compares it to the program's write buffer.
- After executing through the last block mentioned above, operation completes when the card is removed.
- The program's execution progress and results are displayed in the LEDs. The table below lists the display content.

LED3 (P4_4)	LED2 (P4_3)	Meaning
ON	OFF	Execution in progress
OFF	ON	No card inserted or error termination
ON	ON	Normal completion

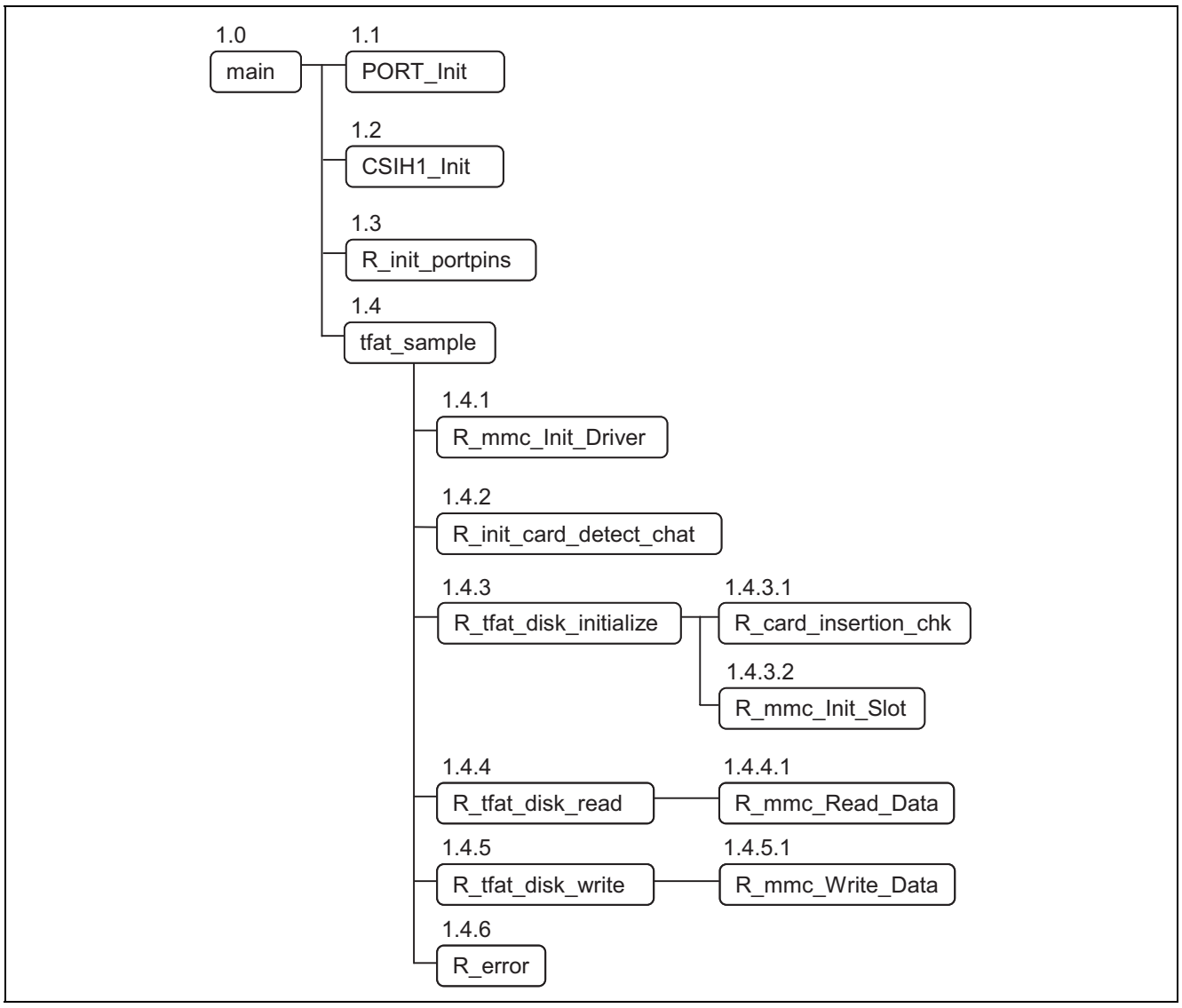
8.2 Processing Flow



8.3 Functions

No	Function Name	Overview
1.0	main	Initializes the microcontroller clock and CPU port LEDs and runs the sample program.
1.1	PORT_Init	Initializes the CSIH ports and DETECT pin.
1.2	CSIH1_Init	Initializes the CSIH module.
1.3	R_init_portpins	Initializes the LED ports.
1.4	tfat_sample	Runs the TFAT file system sample program.
1.4.1	R_mmc_Init_Driver	Initializes MMC driver processing. • This is an API function.
1.4.2	R_init_card_detect_chat	Initializes card insertion/removal detection memory
1.4.3	R_tfat_disk_initialize	Initializes the card. • This is an API function.
1.4.3.1	R_card_insertion_chk	Detects card insertion.
1.4.3.2	R_mmc_Init_Slot	Initializes card control RAM and ports, initializes the card, and switches the card state (from MMC mode to SPI mode).
1.4.4	R_tfat_disk_read	Reads data from a card. • This is an API function.
1.4.4.1	R_mmc_Read_Data	Reads data in 512-byte units from a card.
1.4.5	R_tfat_disk_write	Writes data to a card. • This is an API function.
1.4.5.1	R_mmc_Write_Data	Writes data in 512-byte units to a card.
1.4.6	R_error	Function that is run when an error is detected

8.4 Function Chart



9. Reference Documents

- Hardware Manual
  V850E2/ML4 User's Manual: Hardware [R01UH0262EJ]
  (The latest version can be downloaded from the Renesas Electronics Web site.)

- Software manual
  V850E2M User's Manual: Architecture [R01US0001EJ]
  (Download the latest version from the Renesas Electronics Corporation web site.)

- Related application notes
  [1] V850E2M M3S-TFAT-Tiny: FAT File System Software Adoption Guide [R01AN1028EJ0100]

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Mar.27.12	—	First edition issued
1.01	Jul.23.12	—	Changed source project setting, port initial setting and card clock operations.

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 manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

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 one with a different type number, confirm that the change will not lead to problems.
   — The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

Notice

1. 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.
2. 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.
3. 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.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "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 etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). 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 for which it is not intended. 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 which the product is not intended by Renesas Electronics.
6. 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.
7. 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 systems manufactured by you.
8. 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.
9. 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. You should not use Renesas Electronics products or 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. When exporting the Renesas Electronics 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.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
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.

RENESAS

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: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, 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.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

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

Colophon 2.2