H8S/2215 Group

0.35-µm F-ZTAT Software ECC Programming

Contents

1. Preface

The F-ZTAT* microcomputer incorporates flash memory which is programmable after the microcomputer has been mounted on a board. With the 0.35-µm F-ZTAT microcomputer, reprogramming of the on-chip flash memory up to 100 times is normally guaranteed. However, we can increase the limit to 10,000 times by adding an error correction code (ECC) to the programmed data and using this to detect and correct bit errors during programming.

This application note describes how to add the ECC to the data which is programmed to the flash memory and how to realize the detection and correction of bit errors during programming in software.

This information will be useful to those users who are developing systems which use some of the 0.35-µm F-ZTAT microcomputer's on-chip flash memory as a data area and expect this area to be reprogrammed more than 100 times.

Note: * F-ZTAT (Flexible Zero Turn Around Time) is a trademark of Renesas Technology Corp.

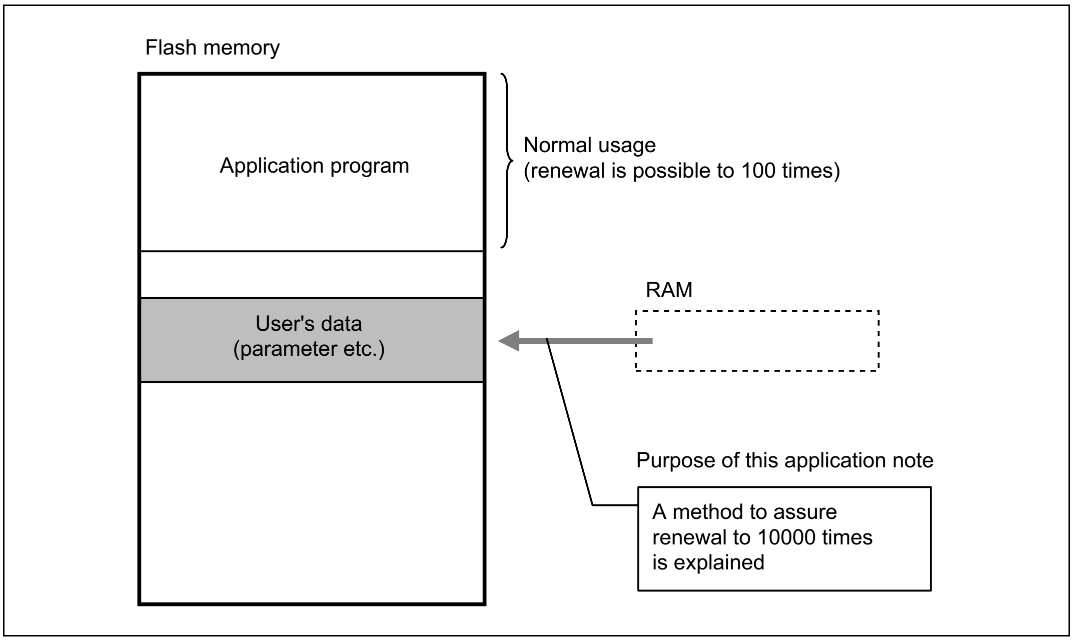


Figure 1.1 Use example of the flash memory

2. Overview of Software ECC

2.1 Objective of Software ECC

The flash memory which is incorporated in the 0.35-µm F-ZTAT microcomputer can only be reprogrammed up to 100 times. This limitation is due to the characteristics of the flash memory in retaining data (the so-called retention characteristic). In general, each programming/erasing operation (W/E cycle) worsens the retention characteristic of a flash-memory unit, but this degradation does not apply evenly to all memory cells of the flash memory. The failure rate of an individual flash-memory cell due to poor retention is very small.

Therefore, we can greatly increase the limit on the number of times the flash memory can be reprogrammed by adding the error correction code (ECC) to the data when it is programmed to the flash memory and using it to detect and correct bit errors in the programmed data. The addition of ECC and the detection and correction of bit errors are both accomplished by software. This software ECC lets us guarantee reprogramming of the flash memory up to 10,000 times. Note, however, that this 10,000-time guarantee only applies to an area no larger than eight kbytes. Also, if you want to apply this method, please carefully read the descriptions in this application note.

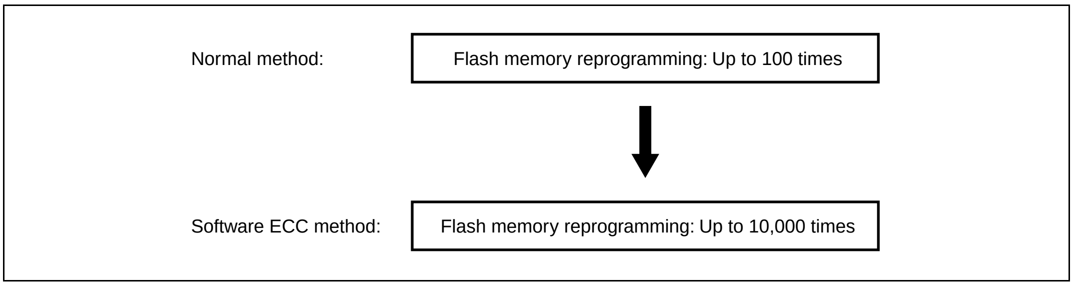

Normal method:
Flash memory reprogramming: Up to 100 times
Software ECC method:
Flash memory reprogramming: Up to 10,000 times

Figure 2.1 Number of Guaranteed Reprogramming with Software ECC

2.2 Function of Software ECC

The rate of error correction by software ECC is 1 bit per 32 bits of user data. Six bits of ECC are required for 32 bits of user data, but if we take the byte boundaries into consideration, one byte (only the lower 6 bits are valid) is used for the ECC of a 32-bit unit of user data. This means that, when the possibility of a bit error in the ECC itself is included, the rate is one bit of error detection and correction per 38 bits. The format in which the user data and ECC are allocated is given in figure 2.2, since we need to consider physical allocation within the flash memory's mats. If another format is used, the defined number of reprogramming operations is not guaranteed.

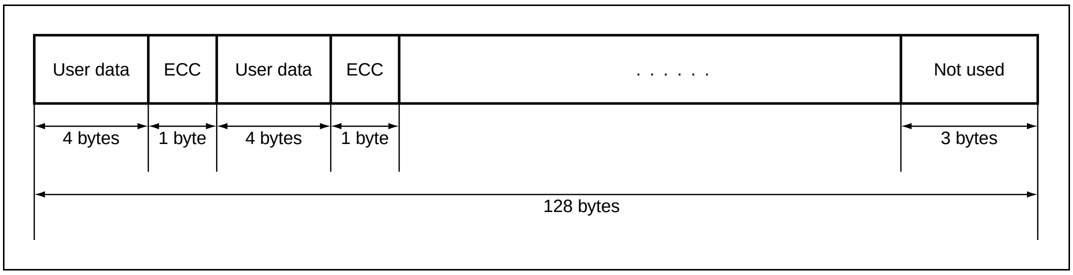


Figure 2.2 Data Format for Programming of Flash Memory

As shown in figure 2.2, each 4 bytes (32 bits) of user data to which error correction is applied and one byte of ECC must be allocated in sequence. The ECC is generated by software. Programming of the flash memory is in 128-byte units, and there are no more than 100 bytes of user data in this unit. The data must be programmed to the flash memory in the format given in figure 2.2. Either of the following forms of errors can be detected and corrected in the data which is read from the flash memory (in the same format as was used for programming):

- One bit of error detection and the correction of a detected bit error in every four bytes of user data
- One bit of error detection and the correction of a detected bit error in every byte of ECC

The position of the software ECC function is shown in figure 2.3. The programming and erasing of flash memory are not covered by this application note, so refer to the descriptions of procedures in the hardware manuals of individual products when actually programming or erasing flash memory.

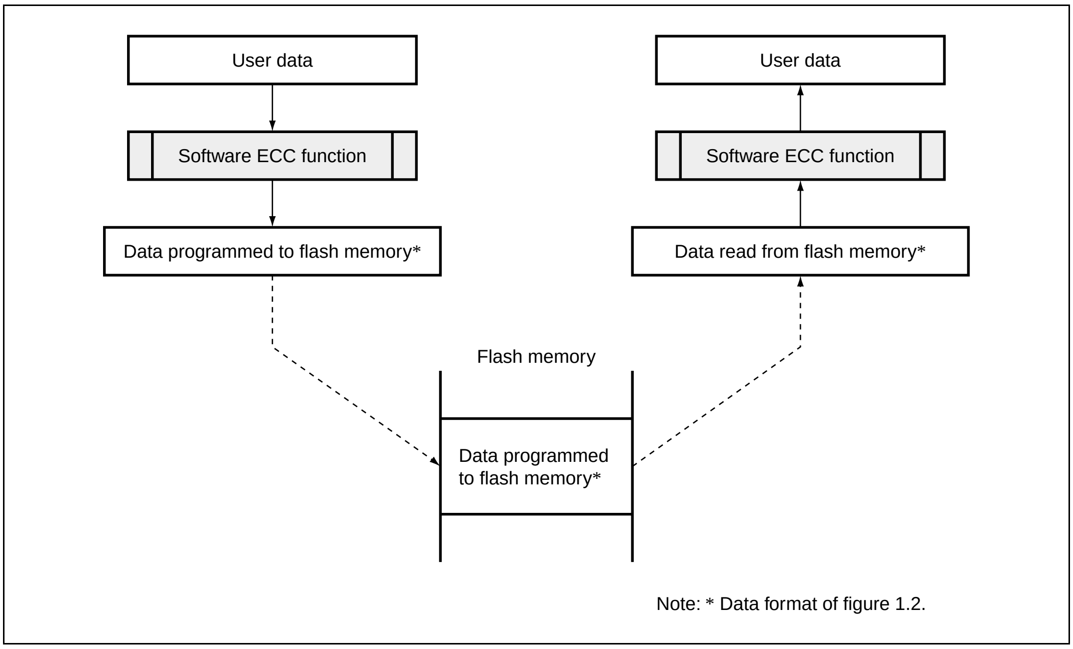

User data
Software ECC function
Data programmed to flash memory*
User data
Software ECC function
Data read from flash memory*
Flash memory
Data programmed to flash memory*
Note: * Data format of figure 1.2.

Figure 2.3 Position of Software ECC Function

3. Algorithm for Software ECC

In software ECC, a Hamming code, which is a single-error correction code, is used. The Hamming code includes the information bits and check bits shown in figure 3.1.

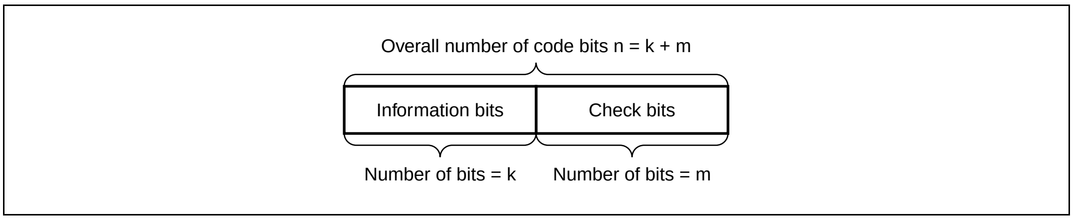


Figure 3.1 Configuration of (n, k) Hamming Code

A Hamming code with 32 information bits and m = 6 satisfies $2^m - m \geq k + 1$. By adding 6 check bits to every 32 bits of information, the correction of a single error becomes possible. Accordingly, we apply a (38, 32) Hamming code. The information bits correspond to 32 bits of user data and the check bits correspond to a 6-bit ECC. The ECC consists of 6 bits, but is applied as a byte (only the lower 6 bits are valid) to take the byte boundaries into consideration.

To simplify the explanation, we give an example of a (7, 4) Hamming code. The Hamming code is shown in figure 3.2. The definition (parity) for generating the check bits is given in table 3.1.

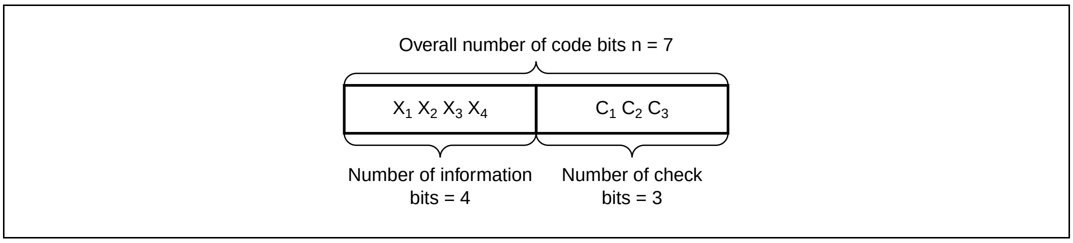


Figure 3.2 Configuration of (7, 4) Hamming Code

Table 3.1 Parity Definition

Position of Error	S_1	S_2	S_3	Hexadecimal Notation
x_1	1	1	1	0x07
x_2	1	1	0	0x06
x_3	1	0	1	0x05
x_4	0	1	1	0x03
c_1	1	0	0	0x04
c_2	0	1	0	0x02
c_3	0	0	1	0x01

Let the pattern of information bits be B′1101. The parity which corresponds to each bit position of the information bits is XORed (exclusively logically added). Since $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, and $x_4 = 1$, we take the XOR of 0x07, 0x06, and 0x03, to obtain 0x02. This result is the value of the check bits, so $c_1 = 0$, $c_2 = 1$, and $c_3 = 0$ (collectively referred to as C).

- When Information Bits Change to B'1111 (x_3 has changed from 0 to 1)

  The check bits for B'1111 are $c_1' = 1$, $c_2' = 1$, and $c_3' = 1$ (collectively referred to as C'). C is not equal to C', so we can see that the value of a bit has changed. The XOR of C and C' is value s where $s_1 = 1$, $s_2 = 0$, and $s_3 = 1$. Table 4.1 tells us that this value indicates an error in bit x_3, so the information bits are corrected to B'1101.

- When Information Bits Change to B'1001 (x_2 has changed from 1 to 0)

  The check bits (C') for B'1001 are $c_1' = 1$, $c_2' = 0$, and $c_3' = 0$. C is not equal to C', so we can see that the value of a bit has changed. The XOR of C and C' is value s, where $s_1 = 1$, $s_2 = 1$, and $s_3 = 0$. Table 4.1 tells us that this value indicates an error in bit x_2, so the information bits are corrected to B'1101.

- When Check Bits Change to B'110 (c_1 has changed from 0 to 1)

  The check bits (C') for B′1101 are $c_1' = 0$, $c_2' = 1$, and $c_3' = 0$. C is not equal to C', so we can see that the value of a bit has changed. The XOR of C and C' is value s, where $s_1 = 1$, $s_2 = 0$, and $s_3 = 0$. Table 4.1 tells us that this value indicates an error in bit c_1, so the check bits are corrected to B'010.

- When Check Bits Change to B'000 (c_2 has changed from 1 to 0)

  The check bits (C') for B'1101 are $c_1' = 0$, $c_2' = 1$, and $c_3' = 0$. C is not equal to C', so we can see that the value of a bit has changed. The XOR of C and C' is value s, where $s_1 = 0$, $s_2 = 1$, and $s_3 = 0$. Table 4.1 tells us that this value indicates an error in bit c_2, so the check bits are corrected to B'010.

- When Information Bits and Check Bits have not Changed

  When C' is obtained by the same procedure as was used to obtain C, the two values are equal, so we can see that no bit error has occurred.

4. Sample Program

A sample program that realizes software ECC is described in this section. The operation of this sample program has been checked. However, when the program is embedded in a user system, its operation must be reconfirmed because of possible differences in the operating environment.

4.1 Configuration of Sample Program and Specifications of Functions

The sample program consists of two files as shown in table 4.1.

Table 4.1 Configuration Files of Sample Program

No.	File Name	Contents
1	FlashECC.h	Header file in which the function prototypes are declared and the constants which are return values of the function are defined.
2	FlashECC.c	Function that realizes software ECC. Consists of four functions, two of which are for the interface with the user system.

Each function is outlined in table 4.2. For further details, refer to the specifications of functions below. The hierarchical structure of the functions is shown in figure 4.1.

Table 4.2 Outline of Functions

No.	Function Name	Description
1	CreateUserToFlash( )*1	Creates data*2 for programming to the flash memory from user data (100 bytes max.), adding one byte of ECC for every four bytes of data.
2	CreateFlashToUser( )*1	Creates user data (100 bytes max.) from the data*2 which is read from the flash memory. One-bit errors detected during the reading of data are corrected.
3	GenerateECC( )	Calculates one byte of ECC for four bytes of user data.
4	CheckECC( )	Performs one-bit error detection for four bytes of user data and one byte of ECC. When a one-bit error is detected, the bit is corrected. An error in two or more bits is detected but cannot be corrected.

Notes: 1. Function for the interface with the user system
2. Data in the format given in figure 3.2.

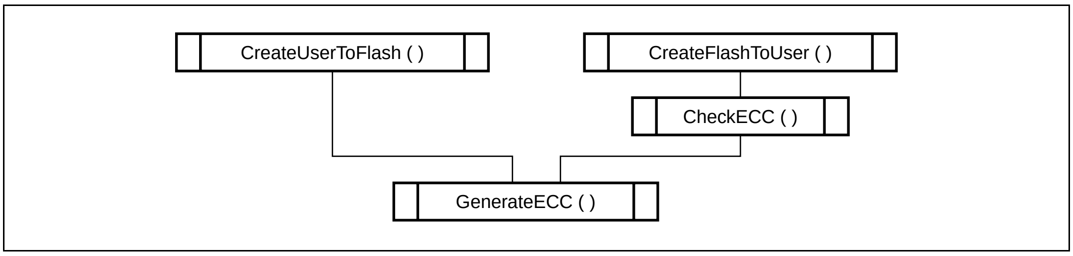
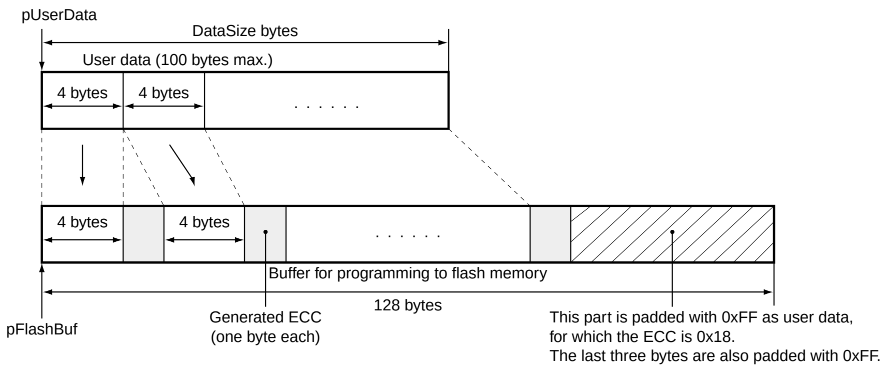
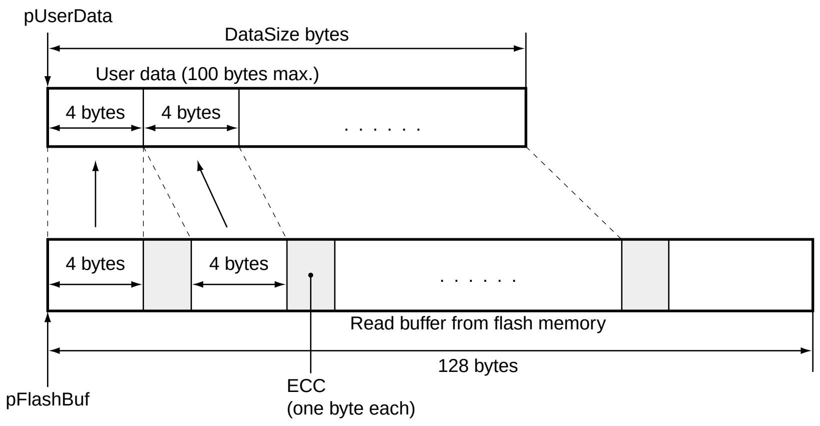


Figure 4.1 Hierarchy of Functions

The specifications of the individual functions are given below.

Creation of data for programming to flash memory	
void CreateUserToFlash (unsigned char *pUserData, unsigned char *pFlashBuf, unsigned char DataSize)	
Function	From user data (100 bytes max.), creates data in the format shown in figure 2.2 for programming to the flash memory by adding one byte of ECC for every four bytes of data.
Parameter	pUserData Pointer to the user data. The user data must be in an area of DataSize consecutive bytes. pFlashBuf Pointer to the buffer for use in programming to the flash memory. The buffer must be a 128-byte area. DataSize Number of bytes of user data: 1 ≤ DataSize ≤ 100.
Return value	None
Description	 pUserData DataSize bytes User data (100 bytes max.) 4 bytes 4 bytes . . . . . . 4 bytes 4 bytes . . . . . . Buffer for programming to flash memory 128 bytes pFlashBuf Generated ECC (one byte each) This part is padded with 0xFF as user data, for which the ECC is 0x18. The last three bytes are also padded with 0xFF. When the length of the user data is not a multiple of four bytes, the last part of the data, i.e. the shortfall from four bytes, is padded with 0xFF. Since reprogramming 10,000 times is guaranteed for up to 8 kbytes, the software ECC can be applied to handle up to 64 x 128-byte-unit program buffers (= 8 kbytes).

Creation of user data from data which is read from flash memory	
unsigned char	CreateFlashToUser (unsigned char *pUserData, unsigned char *pFlashBuf, unsigned char DataSize)
Function	Creates user data from the data in the format shown in figure 2.2, which is read from the flash memory. The error-correction processing (one-bit error detection and correction) is applied to every four-byte unit of user data and corresponding ECC which are read. The user data is created after the errors have been corrected.
Parameter	pUserData Pointer to the user data. The user data must be in an area of DataSize consecutive bytes. pFlashBuf Pointer to the read buffer from the flash memory. The buffer must be a 128-byte area. DataSize Number of bytes for user data: 1 ≤ DataSize ≤ 100.
Return value	When a bit error is not detected: ECC_NOERROR When a bit error is detected and corrected: ECC_REPAIRED When more than one bit error is detected: ECC_FAILED
Description	 pUserData DataSize bytes User data (100 bytes max.) 4 bytes 4 bytes . . . . . . 4 bytes 4 bytes . . . . . . Read buffer from flash memory 128 bytes pFlashBuf ECC (one byte each) One-bit errors in four-byte units of user data (an error in one bit per four bytes) or errors in single bits of one byte of ECC are detected in the read buffer to which the data is read from the flash memory. The user data is then created by applying error correction to the read buffer. Accordingly, the data in the read buffer and the user data are both corrected. In this case, the return value is ECC_REPAIRED. When more than one bit of error is detected in one of the five-byte units in the read buffer, that is, the combination of a four-byte unit of user data and its ECC, the processing is terminated. In this case, the return value is ECC_FAILED and correct user data is not created. However, this will not normally occur within the limit on the number of times the flash memory can be reprogrammed. When no bit error is found in the read buffer, the return value is ECC_NOERROR. The detection of bit errors is performed in five-byte units, that is, the four-byte unit of user data and one-byte ECC (the upper two bits of the ECC are invalid; only 38 bits are valid). One-bit error detection and correction is possible.

Calculation of ECC for four-byte unit of user data	
unsigned char	GenerateECC (unsigned long *pDataItem)
Function	Calculates one byte of ECC (error correction code) for four bytes of user data.
Parameter	pDataItem Pointer to the user data. There must be four bytes of user data.
Return value	ECC
Description	Calculates the ECC for the four bytes of user data which are passed by the pointer. The return value is the calculated ECC.

Bit correction for four-byte unit of user data	
unsigned char	CheckECC (unsigned long *pDataItem, unsigned char *pECC)
Function	Tests a four-byte unit of user data and corresponding one-byte ECC (error correction code) for a bit error and corrects any detected single-bit error.
Parameter	pDataItem Pointer to the user data. The data must be four bytes. pECC Pointer to the ECC. The ECC must be one byte.
Return value	When no bit error is detected: ECC_NOERROR When a bit error is detected and corrected: ECC_REPAIRED When more than one bit error is detected: ECC_FAILED
Description	Determines whether or not there is a bit error in the four-byte user data passed by the pointer or in the corresponding ECC. When an error in one bit of either the user data or the ECC is detected, the error is corrected. The correction is applied to the area passed by the pointers, and thus updates the data in the area indicated by pDataItem and pECC. In this case, the return value is ECC_REPAIRED. When an error in two or more bits is detected in the overall five bytes, i.e. the user data and ECC (the upper two bits of the ECC are invalid; only 38 bits are valid), the errors are not corrected. Here, the return value is ECC_FAILED. However, such an error will not normally occur within the limit on the number of times the flash memory can be reprogrammed. When no error is detected in any bit of the user data and ECC, the return value is ECC_NOERROR.

4.2 Examples of Usage

The interface with the user system is provided by the functions CreateFlashToUser( ) and CreateUserToFlash( ). Simple examples of the usage of these functions are given below.

4.2.1 Usage Example: CreateFlashToUser( )

```
#include  <string.h>
#include  “FlashECC.h”

#define Param_Adrs  0x0030000         //Address of user data in flash
memory
#define Param_Size  (unsigned char)92 //Number of bytes of user data in
flash memory
unsigned char FlashBuf[128];          //Read buffer from flash memory

unsigned char   bUserParam1[11];      //These variables are user data
in RAM
unsigned char   bReserved;            //luded as dummy variable for
boundary adjustment
unsigned long   lUserParam2[20];      //Total area of 92 bytes is used

void  sample1(void)
{
   unsigned char    *pUserData;       //Pointer to user data
   unsigned char    bRtnCode;         //Return value of function

                                      //Read user data from flash
memory
   memcpy((const char *)FlashBuf,(const char *)
Param_Adrs,(size_t)128);
                                      //Create user data from read data
   bRtnCode = CreateFlashToUser(bUserParam1, FlashBuf, (unsigned
char)Param_Size);
   if(bRtnCode == ECC_NOERROR)
   {
         //No bit error detected
   }
   else if(bRtnCode == ECC_REPAIRED)
   {
```

```
            //Bit error detected and corrected
    }
    else
    {
            //More than one bit error detected
    }
       .
       .
       .
    }
```

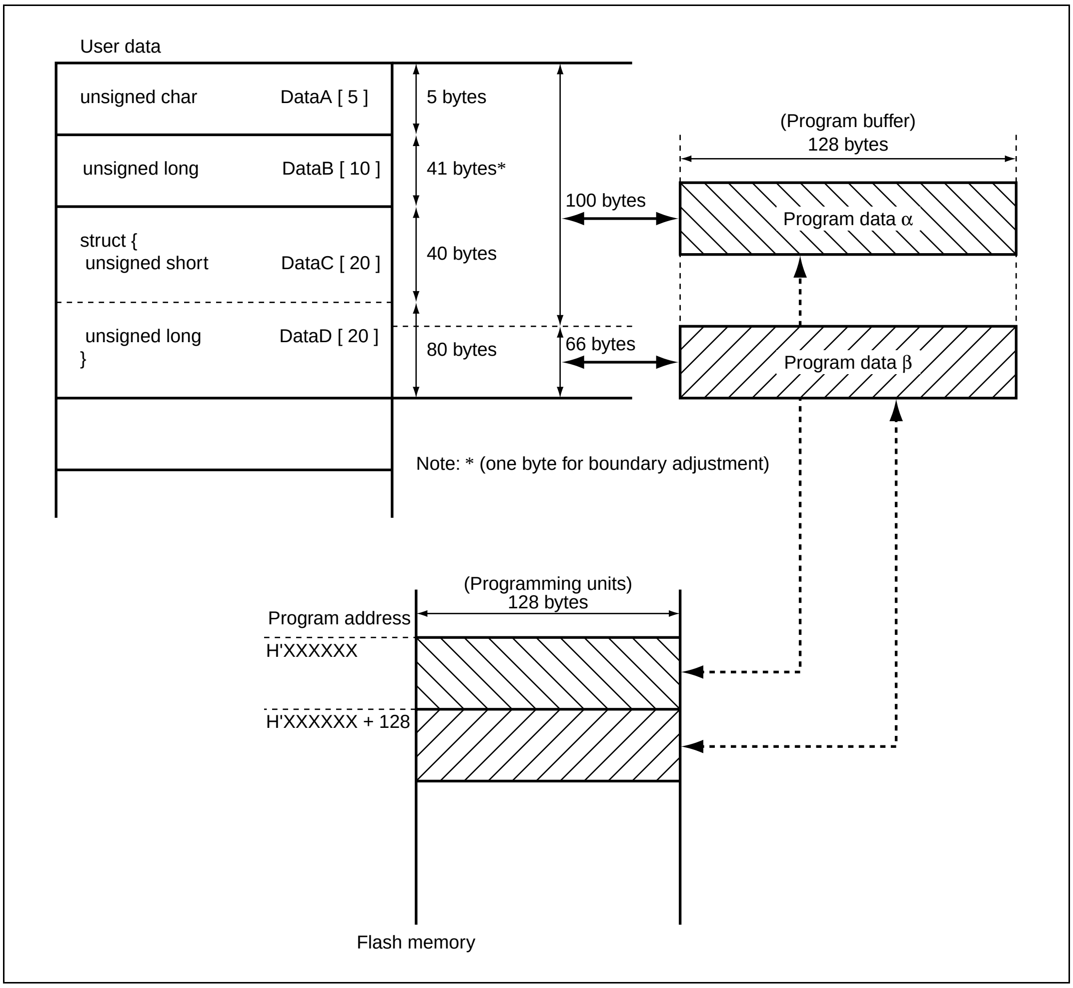
4.2.2 Usage Example: CreateUserToFlash( )

```
#include  “FlashECC.h”


#define Param_Size  (unsigned char)92        //Number of bytes of user
data in flash memory
unsigned char       FlashBuf[128];           //Program buffer to flash
memory


unsigned char       bUserParam1[11];         //These variables are user
data in RAM
unsigned char       bReserved;               //Included as dummy
variable for boundary adjustment
unsigned long       lUserParam2[20];         //Total area of 92 bytes
is used


void   sample2(void)
{
   unsigned char    *pUserData;              //Pointer to user data


      .
      .   (Set user data in RAM)
      .
                                             //Create program data from
user data
CreateUserToFlash (bUserParam1, FlashBuf, (unsigned char)Param_Size);


      .
      .   (Program 128 bytes of FlashBuf to flash memory)
      .
}
```

Note: The actual flash-memory programming processing must be executed in RAM.

4.3 Handling User Data

The unit for programming of the flash memory is 128 bytes, 100 bytes of which can be used to hold user data. However, more than 100 bytes of user data must be divided into 100-byte units. An example is given below.



4.4 Source Program List

```
/***********************************************************************/
/*                                                                     */
/*FILE :FlashECC.h                                                     */
/*DATE :Sep 1, 2002                                                    */
/*DESCRIPTION :Header file of software ECC functions for Flash memory*/
/*CPU TYPE :H8S/2215                                                   */
/*                                                                     */
/***********************************************************************/
#ifndef         _FLASHECC_H_
#define         _FLASHECC_H_

/***********************************************************************/
/*                                                                     */
/*Function prototypes                                                  */
/*                                                                     */
/***********************************************************************/
void                CreateUserToFlash(unsigned char *pUserData,
                    unsigned char *pFlashBuf,
                    unsigned char  DataSize);

unsigned char       CreateFlashToUser(unsigned char *pUserData,
                    unsigned char *pFlashBuf,
                    unsigned char  DataSize);

unsigned char       GenerateECC(unsigned long *pDataItem);

unsigned char       CheckECC (unsigned long *pDataItem, unsigned char
*pECC);
```

```
/*********************************************************************/
/*                                                                   */
/*Return-values of functions                                         */
/*                                                                   */
/*********************************************************************/
enum
{
   ECC_NOERROR ,       /* No error was detected.                     */
   ECC_REPAIRED,       /* One bit repair was completed.              */
   ECC_FAILED          /* Multiple bits error was detected.          */
};

#endif    /*_FLASHECC_H_*/
```

```
/**********************************************************************/
/*                                                                    */
/*FILE :FlashECC.c                                                    */
/*DATE :Sep 1, 2002                                                   */
/*DESCRIPTION :Software ECC functions for Flash memory                */
/*CPU TYPE    :H8S/2215                                               */
/*                                                                    */
/*                                                                    */
/**********************************************************************/
#include     "FlashECC.h"

/**********************************************************************/
/*                                                                    */
/*The T-table for ECC generation                                      */
/*                                                                    */
/**********************************************************************/
static const unsigned char     T[38] =
{                      /* values for the 32-bit data item             */
   0x03, 0x05, 0x06, 0x07, 0x09, 0x0A, 0x0B, 0x0C,
   0x0D, 0x0E, 0x0F, 0x11, 0x12, 0x13, 0x14, 0x15,
   0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D,
   0x1E, 0x1F, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26,
                       /* values for the 6-bit ECC item               */
   0x01, 0x02, 0x04, 0x08, 0x10, 0x20
};
```

```
/*******************************************************************/
/*                                                                 */
/*C Prototype: void  CreateUserToFlash(unsigned char *pUserData,   */
/*                                    unsigned char *pFlashBuf,    */
/*                                    unsigned char  DataSize);    */
/*                                                                 */
/*Function: This routine makes the data to be write into Flash memory*/
/*           including ECC values from user's data.                */
/*                                                                 */
/*Parameters:pUserData is a pointer to user's data. Maximal length of*/
/*           User's data must be 100 bytes.                        */
/*           pFlahBuf is a pointer to the data area to be write into*/
/*           Flash memory. The data area must be in RAM and needs 128*/
/*           bytes area.                                           */
/*           DataSize is a byte-length of user's data. Its value may*/
/*           not exceed 100.                                       */
/*                                                                 */
/* Return : None.                                                  */
/*                                                                 */
/*******************************************************************/
void  CreateUserToFlash(unsigned char *pUserData,
                        unsigned char *pFlashBuf,
                        unsigned char  DataSize)
{
   unsigned char    i;           /* a loop counter                 */
   unsigned char    ecc;         /* an ECC value                   */
   unsigned char    len;         /* a temporary length             */
   unsigned char    fil;         /* byte-length for fill up        */
   union                         /* a data with zero padding       */
   {
      unsigned long   lword;
      unsigned char   bytes[4];
   }  upad;
   /*------------------------------------------*/
   /* Initialize local variables               */
   /*------------------------------------------*/
   upad.lword = 0xFFFFFFFF;
```

```
    fil = 128 - (((DataSize + 3) >> 2) * 5);
    /*------------------------------------------*/
    /* Generate an ECC and a Flash data buffer  */
    /*------------------------------------------*/
    while(0 < DataSize)
    {
        if(4 <= DataSize)         len = 4;
        else                      len = DataSize;
        for(i=0; i<len; i++)      upad.bytes[i] = *(pUserData++);
        ecc = GenerateECC((unsigned long *)(&upad));
        for(i=0; i<4; i++)        *(pFlashBuf++) = upad.bytes[i];
        *(pFlashBuf++) = ecc;
        DataSize -= len;
    }
    /*------------------------------------------*/
    /* Fill up a Flash data buffer              */
    /*------------------------------------------*/
    for(i=0; i<fil;i++)
    {
        if((i % 5) == 4)   *(pFlashBuf++) = 0x18;
        else               *(pFlashBuf++) = 0xFF;
    }
    return;
}
```

```
/*********************************************************************/
/*                                                                 */
/*C Prototype: unsigned char CreateFlashToUser(unsigned char *pUserData,*/
/*                                         unsigned char *pFlashBuf, */
/*                                         unsigned char  DataSize); */
/*                                                                   */
/*Function : This routine makes user's data from the data including ECC */
/*           values which are read from Flash memory into RAM.        */
/*                                                                   */
/*Parameters : pUserData is a pointer to user's data. Maximal length of */
/*             User's data must be 100 bytes.                        */
/*             pFlahBuf is a pointer to the data area which are read from*/
/*             Flash memory into RAM. The data area must be 128 bytes. */
/*             DataSize is a byte-length of user's data. Its value may */
/*             not exceed 100.                                       */
/*                                                                   */
/*Returns : If no error was detected, return-value will be ECC_NOERROR. */
/*          If one bit repair for user's data or an ECC value was    */
/*          detected, return-value will be ECC_REPAIRED.             */
/*          If multiple bits error was detected, return-value will be  */
/*          ECC_FAILED.                                              */
/*                                                                   */
/*********************************************************************/
unsigned char  CreateFlashToUser(unsigned char *pUserData,
                                 unsigned char *pFlashBuf,
                                 unsigned char  DataSize)
{
   unsigned char    i;              /* a loop counter               */
   unsigned char    ecc;            /* an ECC value                 */
   unsigned char    len;            /* a temporary length           */
   unsigned char    rtn;            /* a return-value               */
   union                            /* a data with zero padding     */
   {
      unsigned long    lword;
      unsigned char    bytes[4];
   }  upad;
```

```
    /*------------------------------------------*/
    /* Initialize local variables               */
    /*------------------------------------------*/
    upad.lword = 0xFFFFFFFF;
    rtn = ECC_NOERROR;
    /*------------------------------------------*/
    /* Generate user's data from a Flash buffer */
    /*------------------------------------------*/
    while(0 < DataSize)
    {
        if(4 <= DataSize)  len = 4;
        else               len = DataSize;
        for(i=0; i<4; i++)  upad.bytes[i] = *(pFlashBuf++);
        switch(CheckECC((unsigned long *)(&upad),pFlashBuf))
        {
            case ECC_REPAIRED:
                for(i=0; i<4; i++)  *(pFlashBuf - 4 + i) = upad.bytes[i];
                rtn = ECC_REPAIRED;
                /*--- break statement no need --*/
            case ECC_NOERROR:
                for(i=0; i<len; i++)  *(pUserData++) = upad.bytes[i];
                pFlashBuf++;
                break;
            case ECC_FAILED:
                return(ECC_FAILED);
    }
    DataSize -= len;
    }
    return(rtn);
}
```

```
/**********************************************************************/
/*                                                                    */
/*C Prototype: unsigned char  GenerateECC(unsigned long *pDataItem)  */
/*                                                                    */
/*Function : This routine returns an 8-bit ECC value calculated from a  */
/*           supplied 32-bit data item.                               */
/*                                                                    */
/*Parameters : pDataItem is a pointer to 32-bit data value for which an */
/*             ECC value is required.                                 */
/*                                                                    */
/*Returns : The return value is an 8-bit ECC value. Only lower 6-bit of */
/*          an ECC are valid.                                         */
/*                                                                    */
/**********************************************************************/
unsigned char  GenerateECC(unsigned long *pDataItem)
{
   unsigned char   i;         /* a loop counter                    */
   unsigned char   ecc;       /* an ECC value                      */
   unsigned long   mask;      /* a bit mask value                  */
   /*------------------------------------------*/
   /* Initialize local variables               */
   /*------------------------------------------*/
   ecc  = 0x00;
   mask = 0x00000001;
   /*------------------------------------------*/
   /* Calculate for each bit of a 32-bit data  */
   /*------------------------------------------*/
   for(i=0; i<32; i++)
   {
      if((*pDataItem & mask) != 0)  ecc ^= T[i];
      mask <<= 1;
   }
   return(ecc);
}
```

```
/*********************************************************************/
/*                                                                   */
/*C Prototype: unsigned char  CheckECC(unsigned long *pDataItem,     */
/*                                     unsigned char *pECC);         */
/*                                                                   */
/*Function : This routine checks the validity of a 32-bit data item and */
/*           an 8-bit ECC value. If an error is detected, a repair is */
/*           performed. The repair can be in operation for only one bit */
/*           error.                                                  */
/*                                                                   */
/*Parameters : pDataItem is a pointer to the 32-bit data value.      */
/*             pECC is a pointer to the 8-bit original ECC value.    */
/*                                                                   */
/*Returns : If no error was detected, return-value will be ECC_NOERROR. */
/*           If one bit repair for user's data or an ECC value was   */
/*           detected, return-value will be ECC_REPAIRED.            */
/*          If multiple bits error was detected, return-value will be */
/*          ECC_FAILED.                                              */
/*                                                                   */
/*********************************************************************/
unsigned char  CheckECC(unsigned long *pDataItem,
                        unsigned char *pECC)
{
   unsigned char    i;          /* a loop counter                    */
   unsigned char    ecc;        /* an ECC value                      */
   unsigned long    mask;       /* a bit mask value                  */
   /*------------------------------------------*/
   /* Initialize a local variable              */
   /*------------------------------------------*/
   mask = 0x00000001;
   /*------------------------------------------*/
   /* Check an ECC and repair a data an ECC    */
   /*------------------------------------------*/
   ecc = GenerateECC(pDataItem);
   if(ecc != *pECC)
   {
      for(i=0; i<sizeof(T); i++)
```

```
        {
            if(T[i] == (ecc ^ *pECC))
            {
                if(32 <= i)  *pECC      ^= (unsigned char)mask;
                else         *pDataItem ^= mask;
                return(ECC_REPAIRED);
            }
            if(i == 31)  mask    = 0x00000001;
            else         mask <<= 1;
        }
        return(ECC_FAILED);
    }
    else
    {
        return(ECC_NOERROR);
    }
}
```

5. Precautions

The following is a list of precaution to take in applying software ECC. Some relevant descriptions have already been given in the previous sections.

1. The data format for programming of the flash memory in terms of the physical allocation of the memory mat must be as shown in figure 3.2.
2. The error correction involves the detection and correction of single-bit errors in 32-bit user-data units. If two or more bits are in error within a 32-bit user-data unit, the error is detected but cannot be corrected.
3. 100 bytes of user data is retained in each unit for programming (128 bytes) of the flash memory.
4. The reprogramming of any 8-kbyte span within flash memory 10,000 times can be guaranteed. Here, eight kbytes include the user data and the ECC. Accordingly, this holds 64 units in the data format shown in figure 3.2. For other areas, the normal guarantee of 100 reprogramming operations applies.
5. The user data is divided into four-byte units for programming to the flash memory. Since the user data is assumed to be continuous, take care to adjust the user data to correctly fill the spaces between boundaries if there are empty areas.
6. Since the ECC is inserted every four bytes, this area cannot be used to hold instructions. That is, an area of flash memory which contains ECCs is not available for use as a memory-mapped program area.
7. If the flash-memory read time due to software ECC processing becomes a problem for the user system, apply a countermeasure such as moving all of the user data from flash memory to RAM on resets.