

---

# H8S/2623F Group

## On-Board Program with CAN Bus Example

---

### Introduction

Onboard program algorithm written on this material is not guaranteed algorithm.

And, neither program time, nor other data are guaranteed value.

This material is made for customer reference for system designing.

### Contents

1. Specifications .....	2
2. Operation Overview.....	5
3. Operation Explanation .....	9
4. Software Explanation .....	13
5. Flow Chart .....	19
6. Hardware Explanation .....	33
7. Circuit Figure .....	36
8. Programming List .....	37

## 1. Specifications

### (1) Operation condition

- Device : HD64F2623FA20
- CPU operation mode : Mode 7
- Operation voltage : Vcc=3.3V, PVcc=5.0V
- Operation frequency : 20MHz

### (2) Onboard programming mode

- User program mode  
 (Provided that Write /Erase program, RAM forward program, FWE judge program are already programmed by boot mode or writer mode)

### (3) Program method

- Receive the program data from forward source and program them to Flash memory
- CAN (Controller Area Network) is used for a communication with forward source.  
 Following shows the forward format

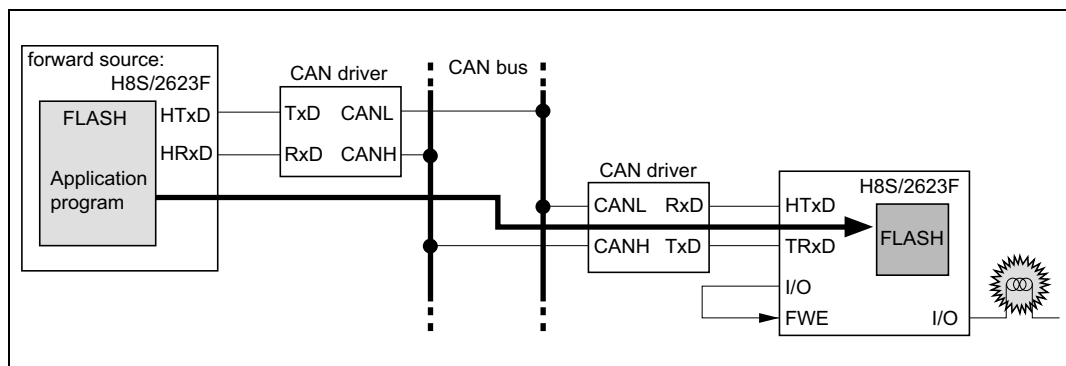
Master is shown as forward source and slave is shown as receive side.

### (4) FWE pin control

- FWE pin circuit below is an example of the user programming without using Renesas Technology onboard program tool.

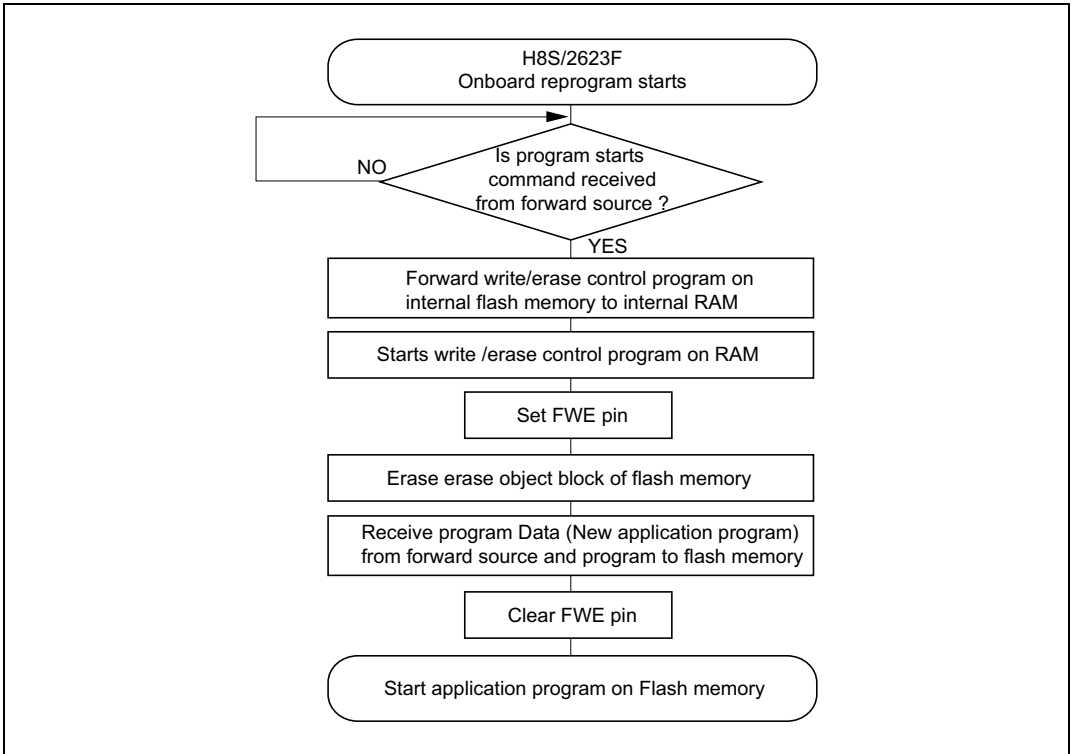
FWE pin is controlled by H8S/2623F of receive side with I/O port(PB0).

### (5) Onboard program structure



**Figure 1.1 Onboard programming structure with CAN**

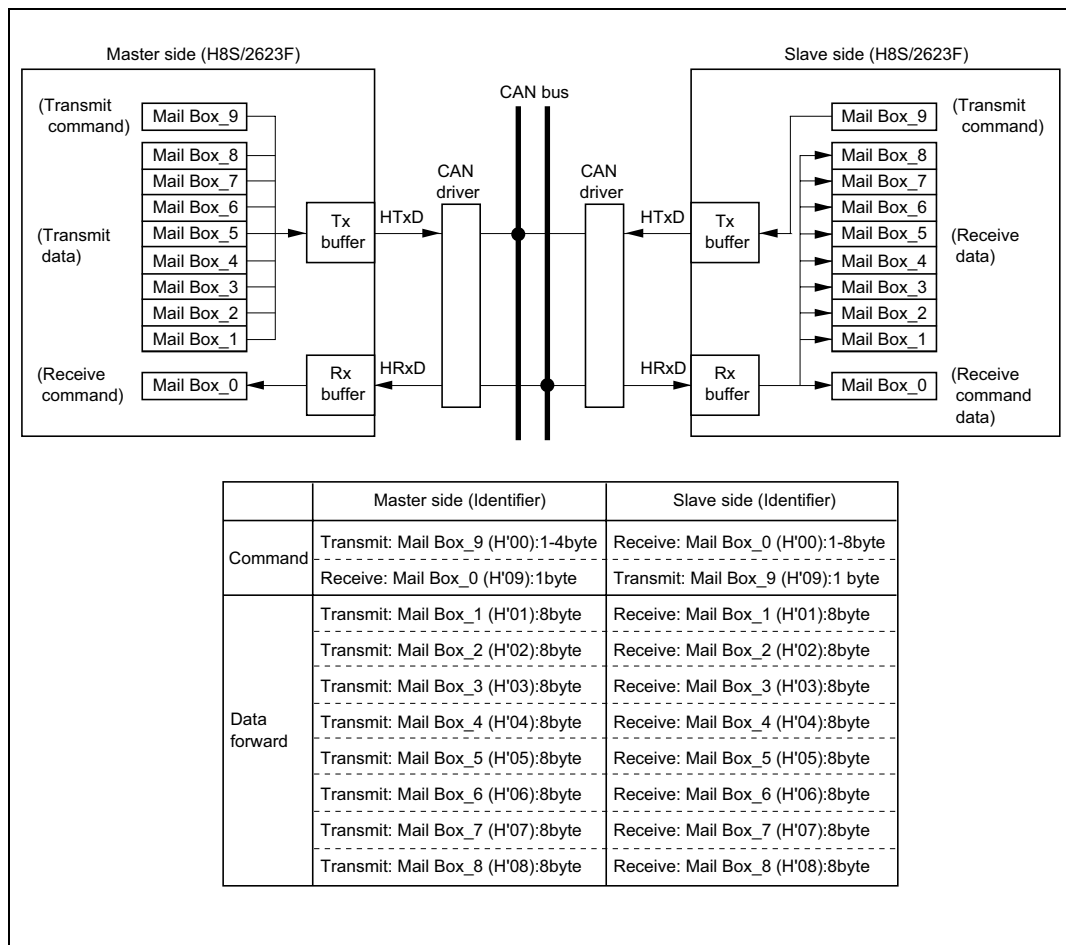
**(6) H8S/2623F Onboard reprogram execution order**



**Figure 1.2 Onboard programming order with CAN**

**(7) CAN specifications**

- Bit rate : 1 Mbps
- Number of using mail box : 10
- format : Standard format (Identifier : 11bit)
- Message priority : Mail box number order (From small number)



**Figure 1.3 Mail box allocation**

## 2. Operation Overview

Following are onboard programming operation overview with CAN

### (1) Normal operation

(1): Write/erase control program is written on a part of an application program on flash memory.

FWE pin set up method, application forward method are already set.

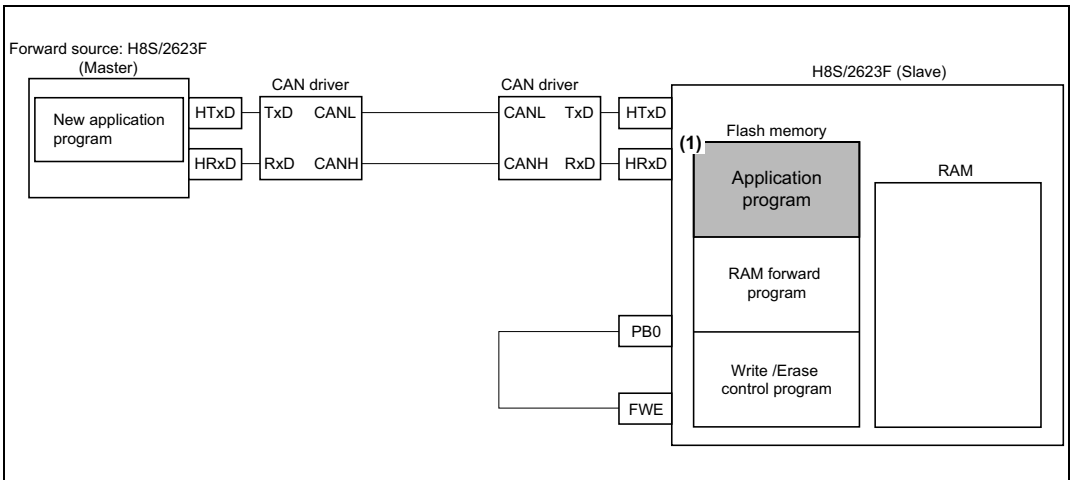


Figure 2.1 Normal operation

### (2) Onboard reprogram starts

(2): Receive programming start command from forward source.

(3): RAM forward program is started by application program and forwards write/erase control program to internal RAM on flash memory

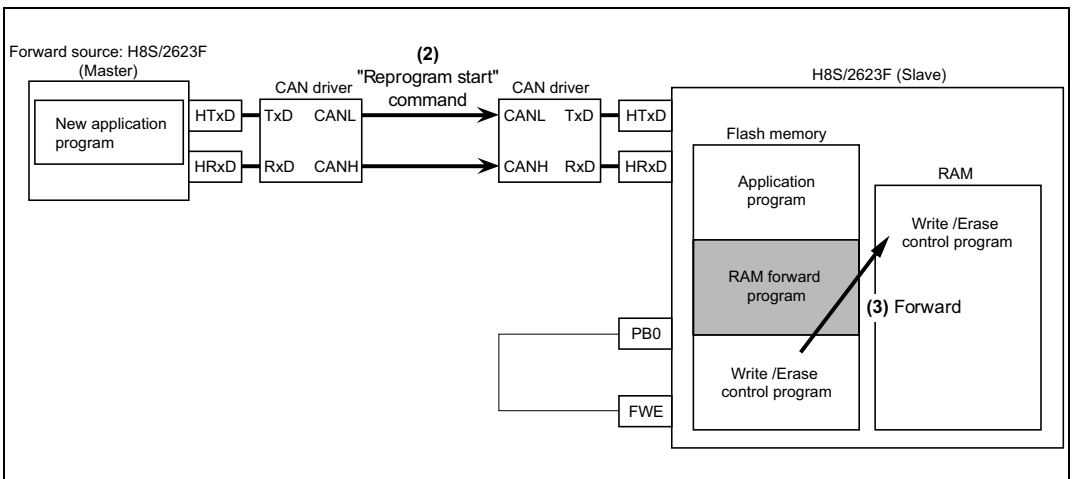
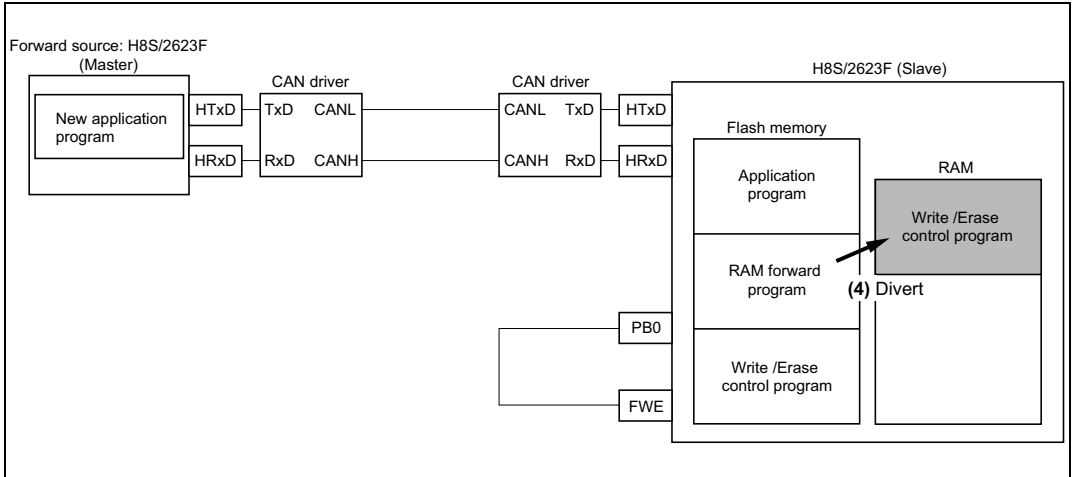


Figure 2.2 Onboard reprogram starts

**(3) Write/Erase control program starts**

**(4):** After forward, RAM forward program branches to Write / Erase control program

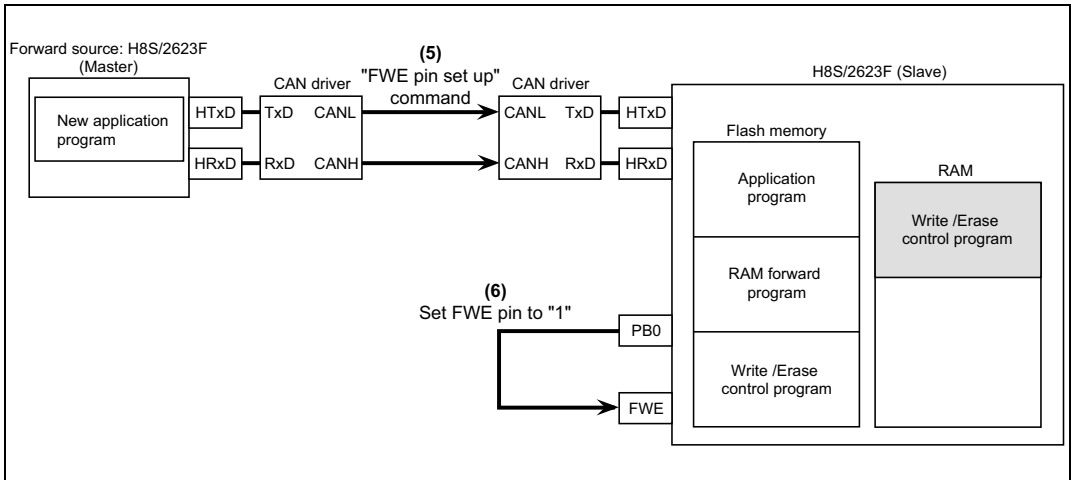


**Figure 2.3 Write/Erase control program start**

**(4) FWE pin set up**

**(5):** Receive forward source from FWE pin set up command

**(6):** Write/Erase control program controls PB0 and set FWE pin to "1"

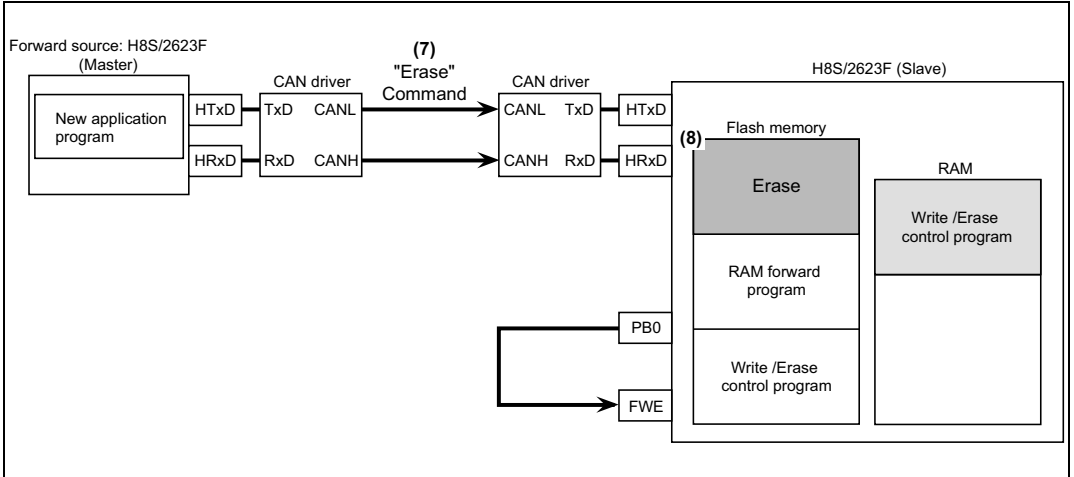


**Figure 2.4 FWE pin set up**

**(5) Erase flash memory**

**(7):** Receive erase command from forward source

**(8):** Erase object block of flash memory is erased by write /erase control program

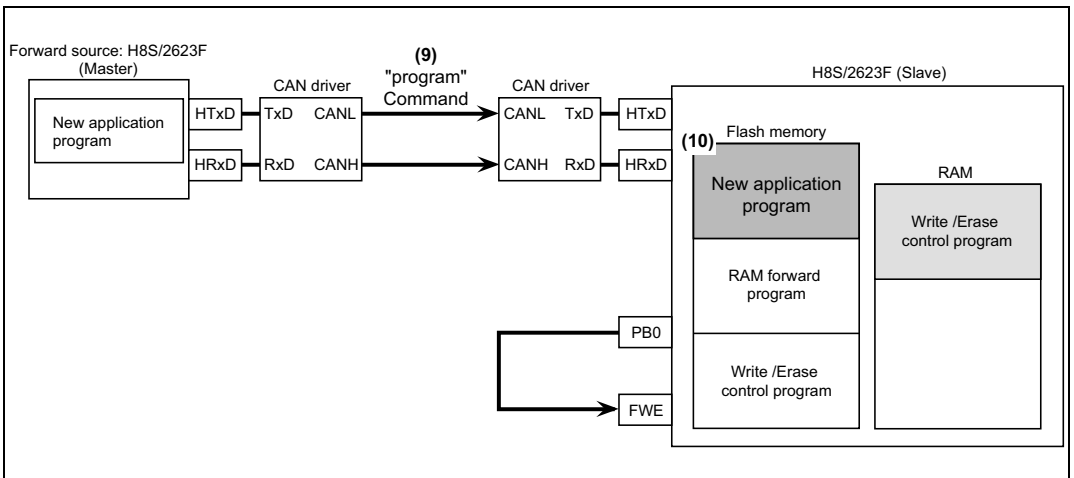


**Figure 2.5 Flash memory erase**

**(6) Flash memory program**

**(9):** Receive write command from forward source

**(10):** Write /Erase control program receives new application program from forward source and write it to flash memory

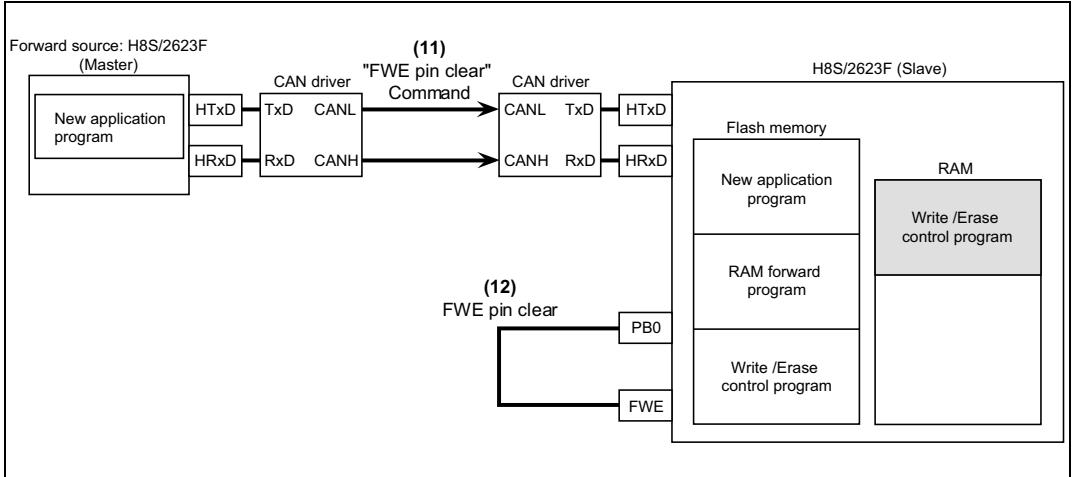


**Figure 2.6 Flash memory program**

**(7) FWE pin clear**

**(11):** Receive FWE pin erase command from forward source

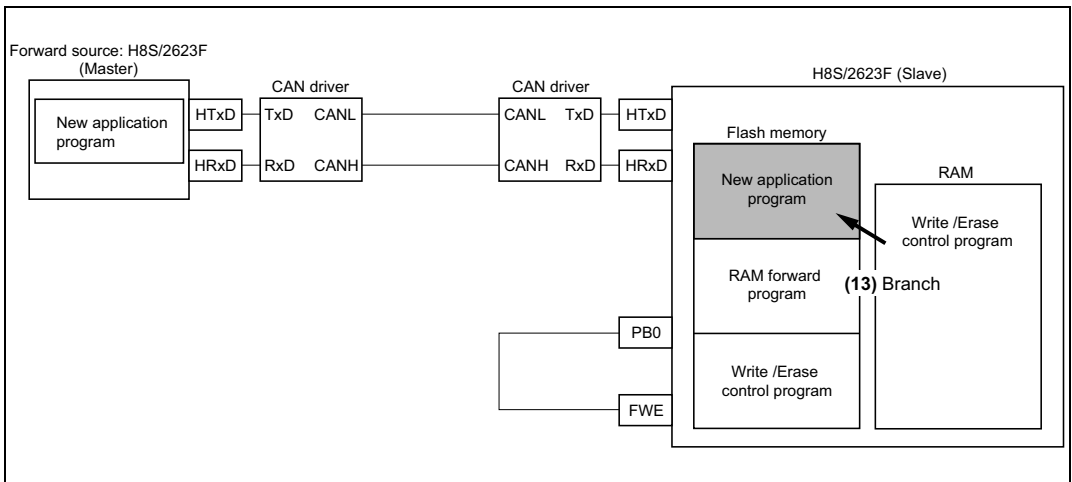
**(12):** Controls Write/ Erase control program PB0 and sets FWE pin to “0”



**Figure 2.7 FWE pin clear**

**(8) New application program start up**

**(13):** Write/Erase control program branches to new application program in flash memory

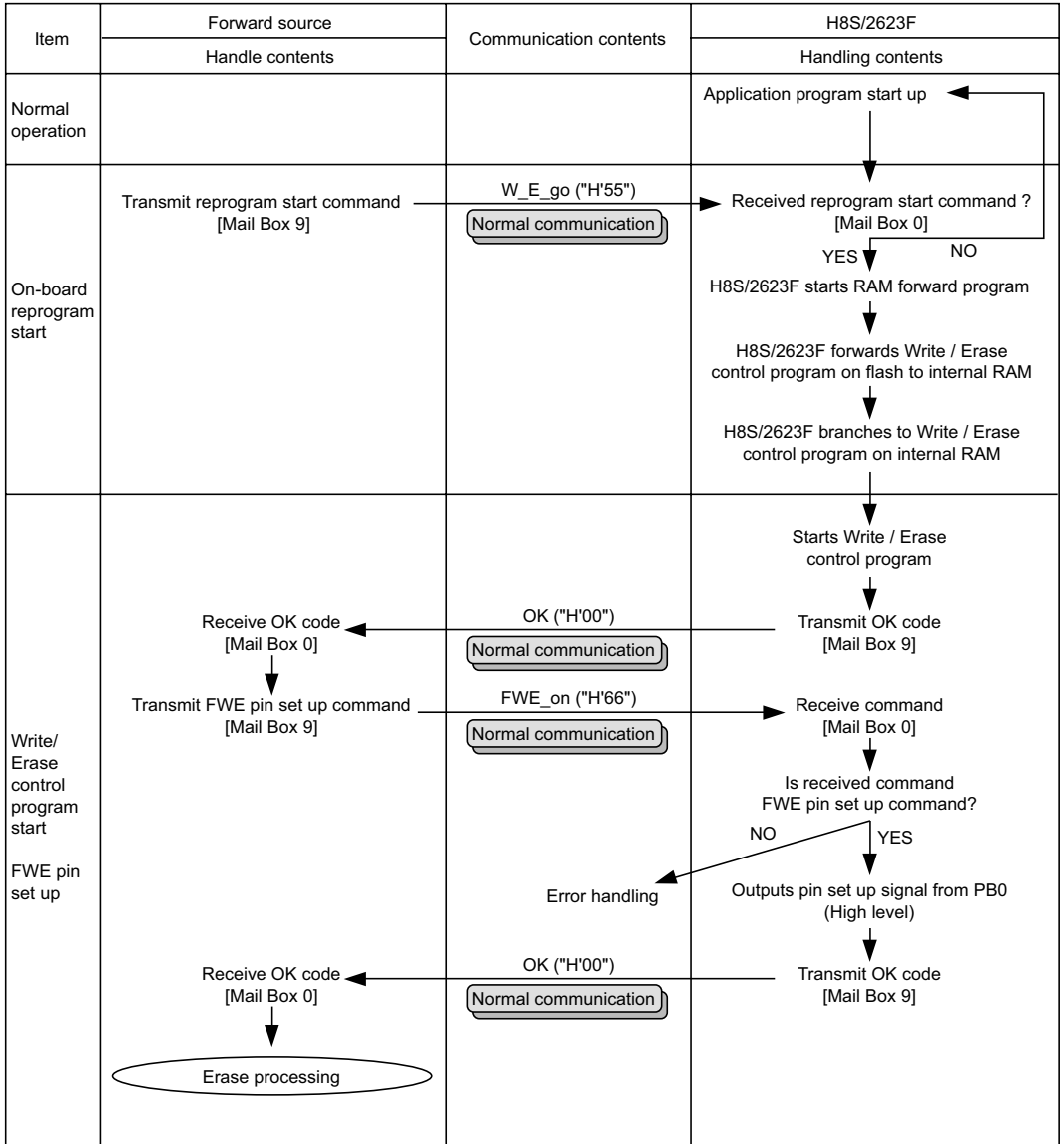


**Figure 2.8 New application program start up**

### 3. Operation Explanation

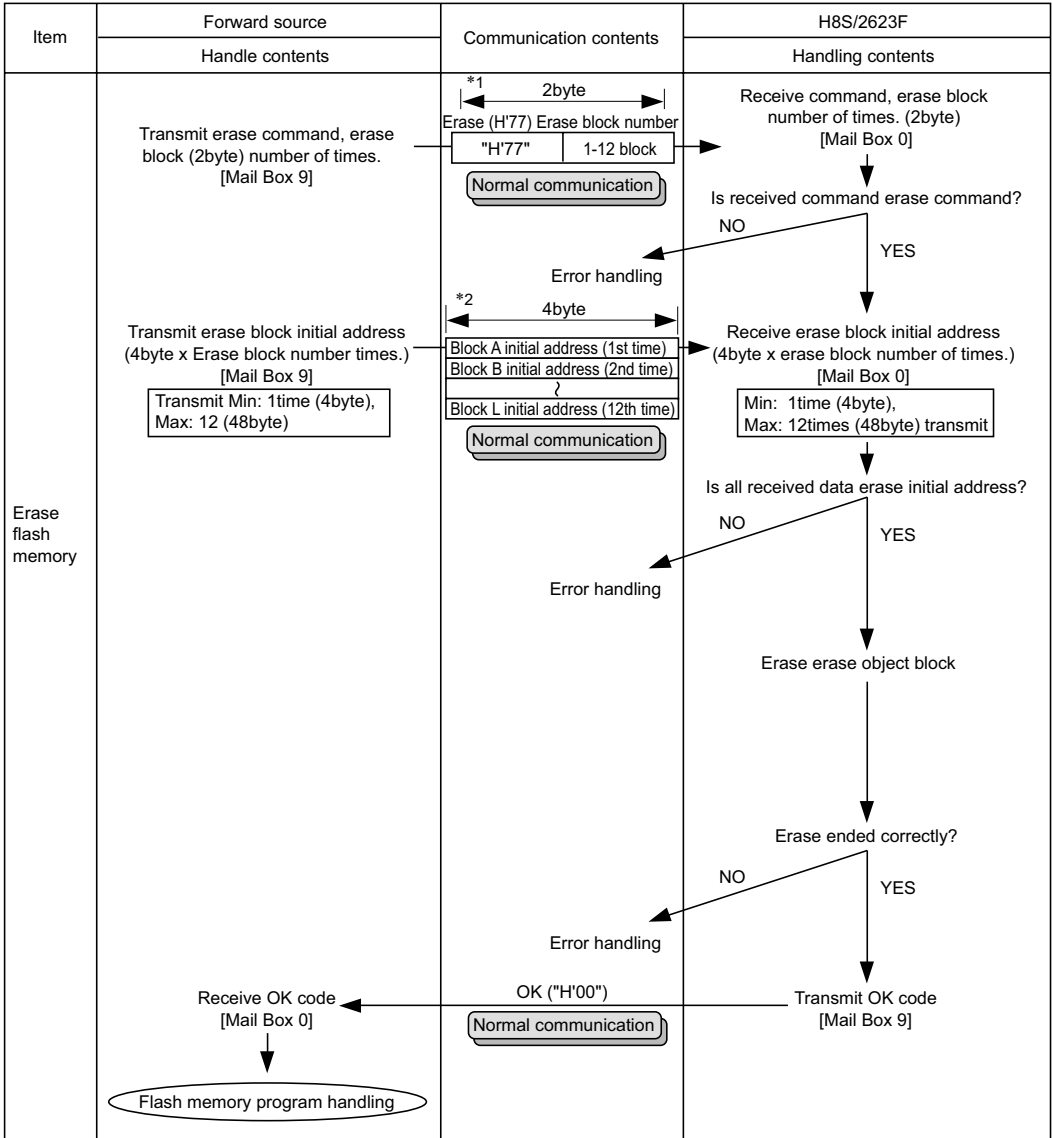
Figure 3.1 shows onboard reprogramming communication contents and CAN bus interface.

Parity which forward source transmits and acknowledge which received side transmits are abbreviated.



Follow to next page

**Figure 3.1 Communication contents for onboard programming(1)**

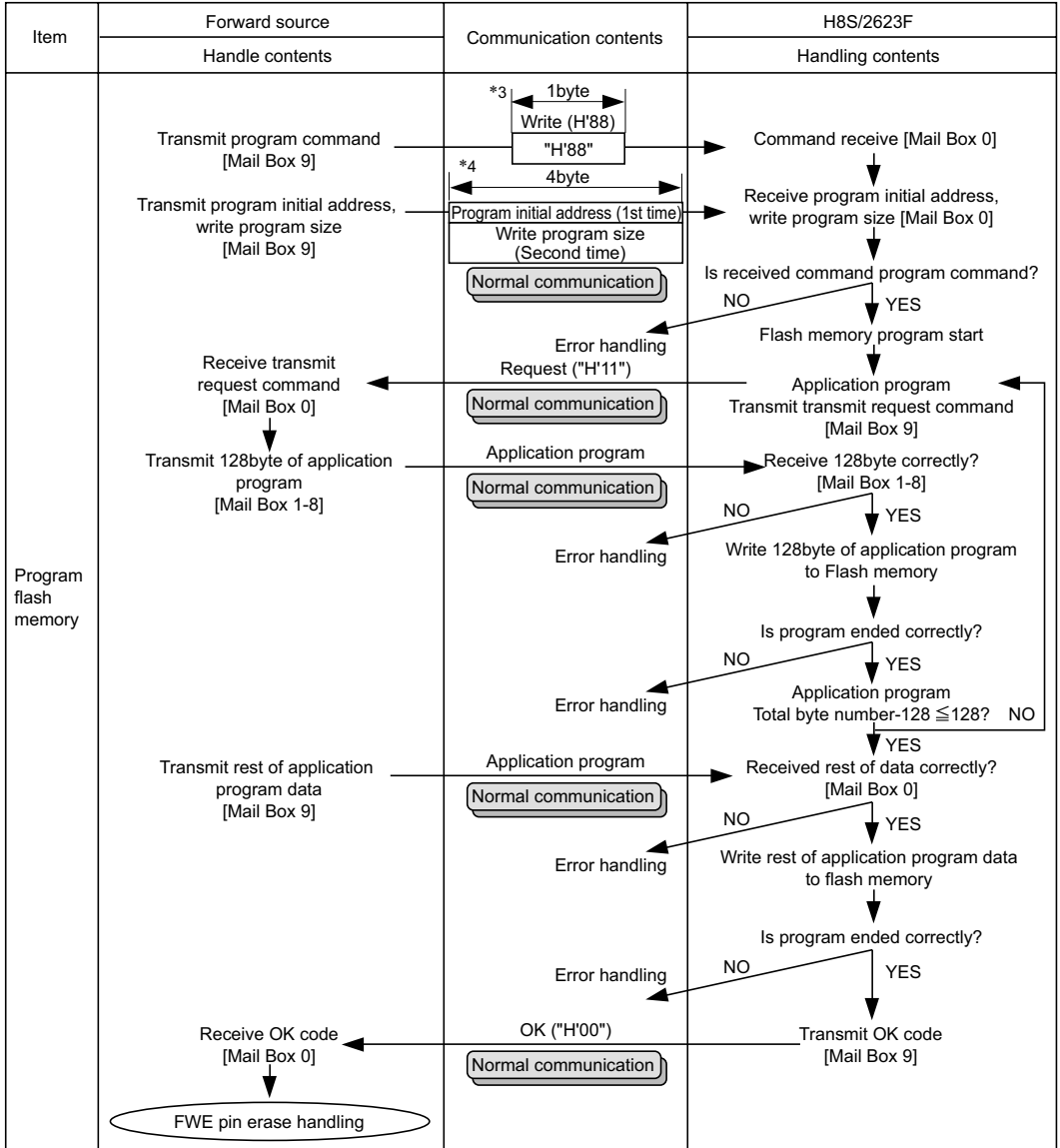


Follow to next page

\*1: Forward source transmits 2byte data of erase command (H'77) and erase block number (H'01-H'0C) to H8S/2623F.

\*2: After transmission, forward source transmits block initial address(4byte) erase block number of times.

**Figure 3.2 Communication contents for onboard programming (2)**

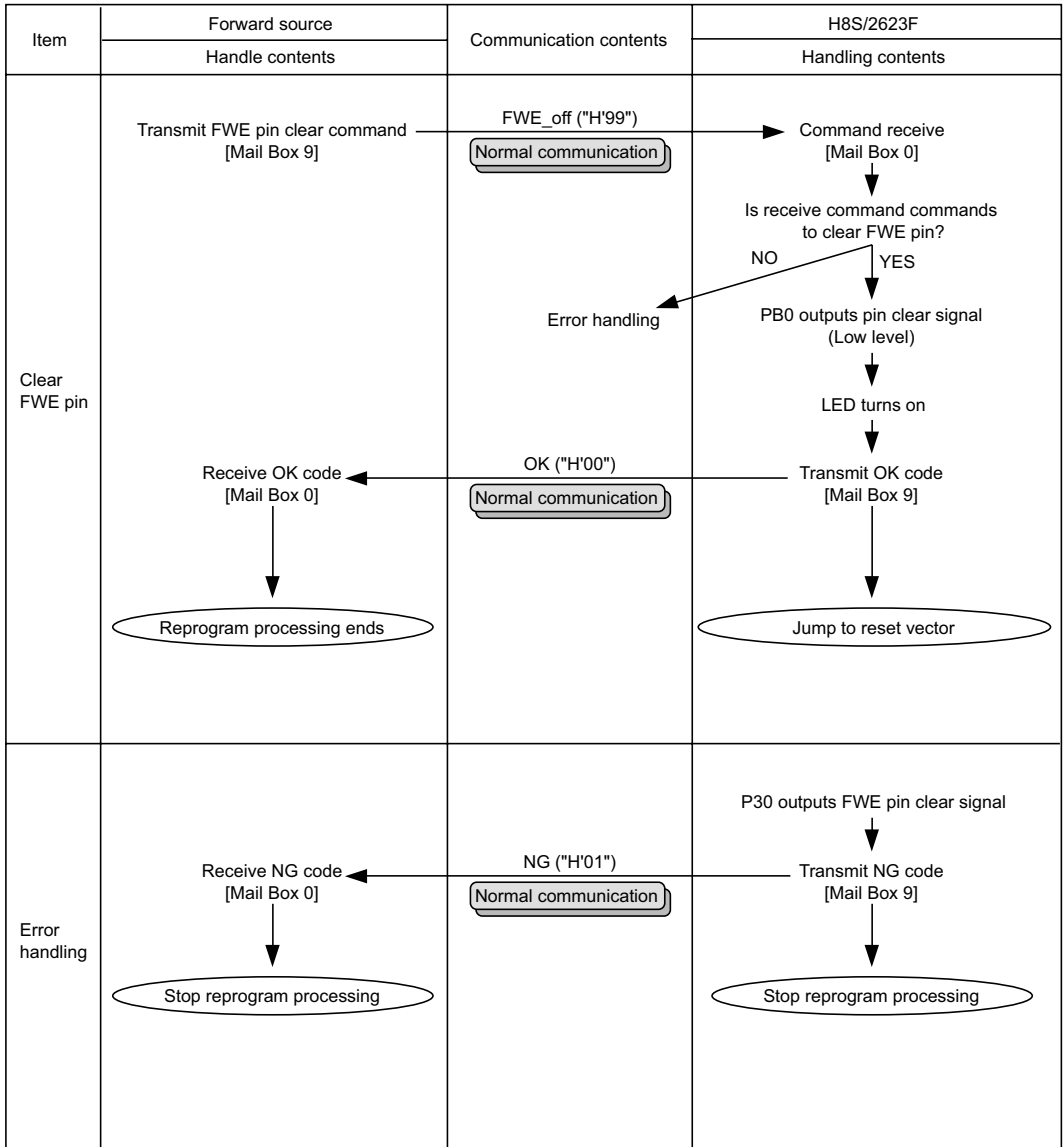


Follow to next page

\*3: Forward source transmits 1byte data of program command (H'88) to H8S/2623F.

\*4: After transmission, transmit program initial address (4byte) and byte number (4byte) of application program to transmit.

**Figure 3.3 Communication contents for onboard programming (3)**

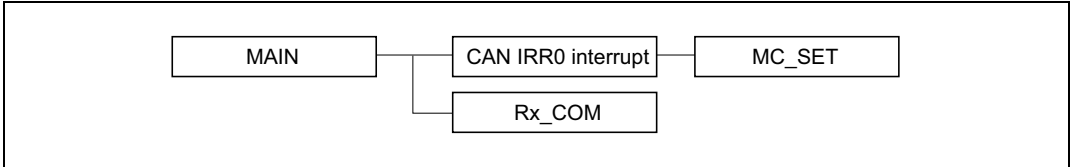


**Figure 3.4 Communication contents for onboard programming (4)**

## 4. Software Explanation

### (1) Program level structure of user program mode start up

Figure 4.1 shows level structure of user program mode start up program(reprogram start command receive, process to forward Write / Erase control program on flash memory to internal RAM)executed on flash memory.



**Figure 4.1 Program level structure of user program mode start up**

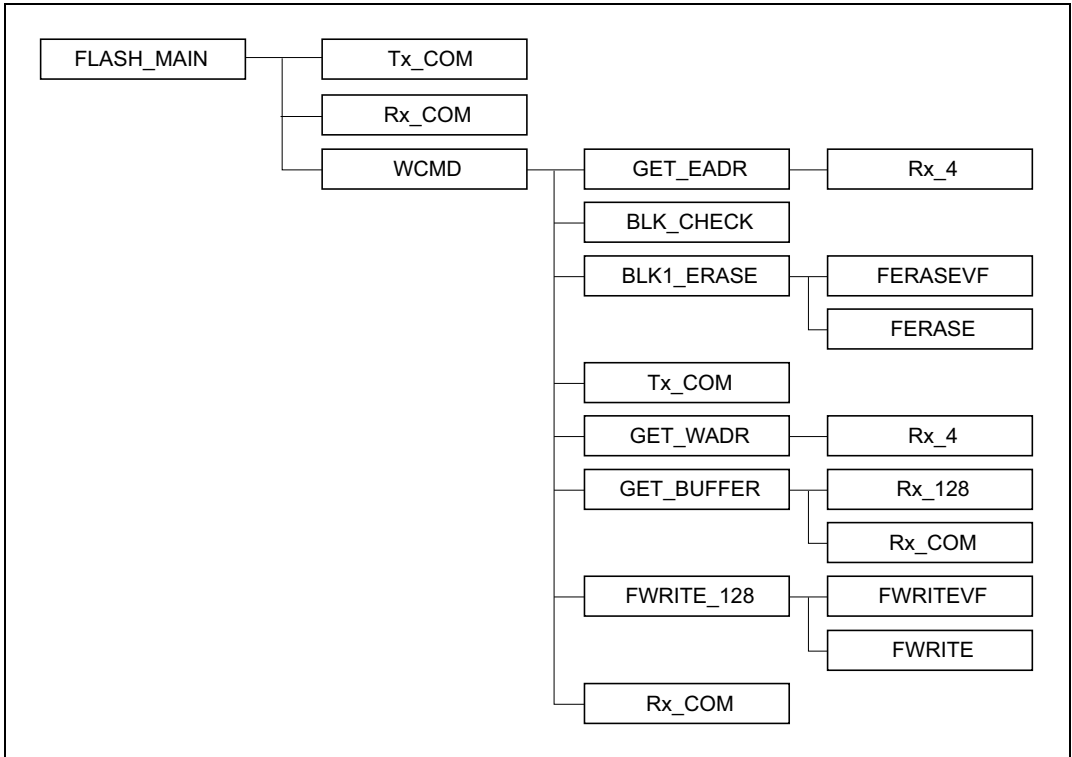
### (2) Program for user program mode start up module

**Table 4.1 Program for user program start up module**

Module	Arguments		Returned Value		Function Allocation
	Contents	Register (Data Length)	Contents	Register (Data Length)	
MAIN	—	—	—	—	Main routine
RX_COM	—	—	Data receive (1byte)	R2L (1byte)	Receive data from forward source through CAN bus
IRR0	—	—	—	—	HCAN IRR0 interrupt routine
MC_SET	MCn register address	ER5 (4byte)	MCn register	@ER5 (3byte)	Set MCn (message control)
	Data length	E6 (2byte)			
	Identifier	R6L(1byte)			

**(3) Write/Erase control program level structure**

Figure 4.2 shows level structure of Write / Erase control program executed on internal RAM



**Figure 4.2 Write/Erase control program level structure**

#### (4) Write/Erase control program module

**Table 4.2 Module of Write / erase control program**

Module	Arguments		Returned Value		Function Allocation
	Contents	Register (Data Length)	Contents	Register (Data Length)	
FLASH_MAIN	—	—	—	—	Main routine of Write / Erase control program
RX_COM	—	—	Receive data (1 byte)	R2L (1 byte)	Receive data from forward source through CAN bus
TX_COM	Transmit data	R2L (1byte)	—	—	Transmit data from forward source through CAN bus
WCMD	—	—	—	—	Write/Erase control routine Transmit/Receive command from forward
GET_EADR	—	—	Normal"0", Error"1" Erase block number Erase initial address	R0L (1 byte) ERASEBLOCK (1 byte) E_ADR (4-48 byte)	Receive erase block number and erase initial address from forward source through CAN bus
RX_4	Received data storage address	ER3 (4 byte)	Storage address	@ER3 (4 byte)	Store received data from forward source through CAN bus to @ER3
BLK_CHECK	Erase initial address	ER3 (4byte)	Normal"0", Error"1" EBR Register address FLMCR Register address Erase object block No. Erase verify initial address Erase verify final address	R0L (1 byte) ER5 (4 byte) ER6 (4 byte) BLK_NO (1byte) EVF_ST (4byte) EVF_ED (4 byte)	Check which block erase initial address applies and load applied address of FLMCR,ERB register to general register. Store each erase object block no., erase verify initial and last address to each work RAM.

Module	Arguments		Returned Value		Function Allocation
	Contents	Register (Data Length)	Contents	Register (Data Length)	
BLK_ERASE	—	—	Normal“0”, Error“1”	R0L (1 byte)	Flash erase processing
FERASEVF	FLMCR register address	ER6 (4 byte)	Normal“0”, Error“1”	R0L (1 byte)	Read last address data from erase verify initial address, and check erase is completed.
	Erase verify initial address	EVF_ST (4 byte)			
	Erase verify final address	EVF_ED (4 byte)			
FERASE	EBR register address	ER5 (4 byte)	Normal“0”, Error“1”	R0L (1 byte)	Erase erase object block number.area
	FLMCR register address	ER6 (4 byte)			
	Erase object block No.	BLK_NO (1 byte)			
GET_WADR	—	—	Normal“0”, Error“1”	R0L (1 byte)	Receive program initial address and program data size from forward source through CAN bus
			Program initial address	W_ADR (4 byte)	
			Program data size	RESTSIZE (4 byte)	
GET_BUFFER	Program data size	RESTSIZE (4 byte)	Program data size	RESTSIZE (4 byte)	Routine to store 128byte data to program data area
RX_128	Program data area initial address	ER3 (4 byte)	Program data area	W_BUFF (128 byte)	Receive 128byte application program data from forward source through CAN bus and store 128byte receive data to program data area
FWRITE_128	Program initial address	W_ADR (4 byte)	Normal“0”, Error“1”	R0L (1 byte)	Flash program processing in 128byte unit
	Program data	W_BUFF (128 byte)	Reprogram data	BUFF (128 byte)	

Module	Arguments		Returned Value		Function Allocation
	Contents	Register (Data Length)	Contents	Register (Data Length)	
FWRITEVF	FLMCR register address	ER6 (4 byte)	Normal"0", Error"1"	R0L (1 byte)	Reprogram arithmetic and additional program arithmetic. Check if 128byte program data is normally programmed.
	Reprogram data	BUFF (128 byte)	Reprogram data	BUFF (128 byte)	
	Program data	W_BUFF (128 byte)	Additional program data	OW_BUFF (128 byte)	
	Additional program data	OW_BUFF (128 byte)			
FWRITE	P bit set time	ER3 (4 byte)	—	—	Program 128byte unit (Flash voltage application)
	FLMCR register address	ER6 (4 byte)			
	Program initial address	W_ADR (4 byte)			

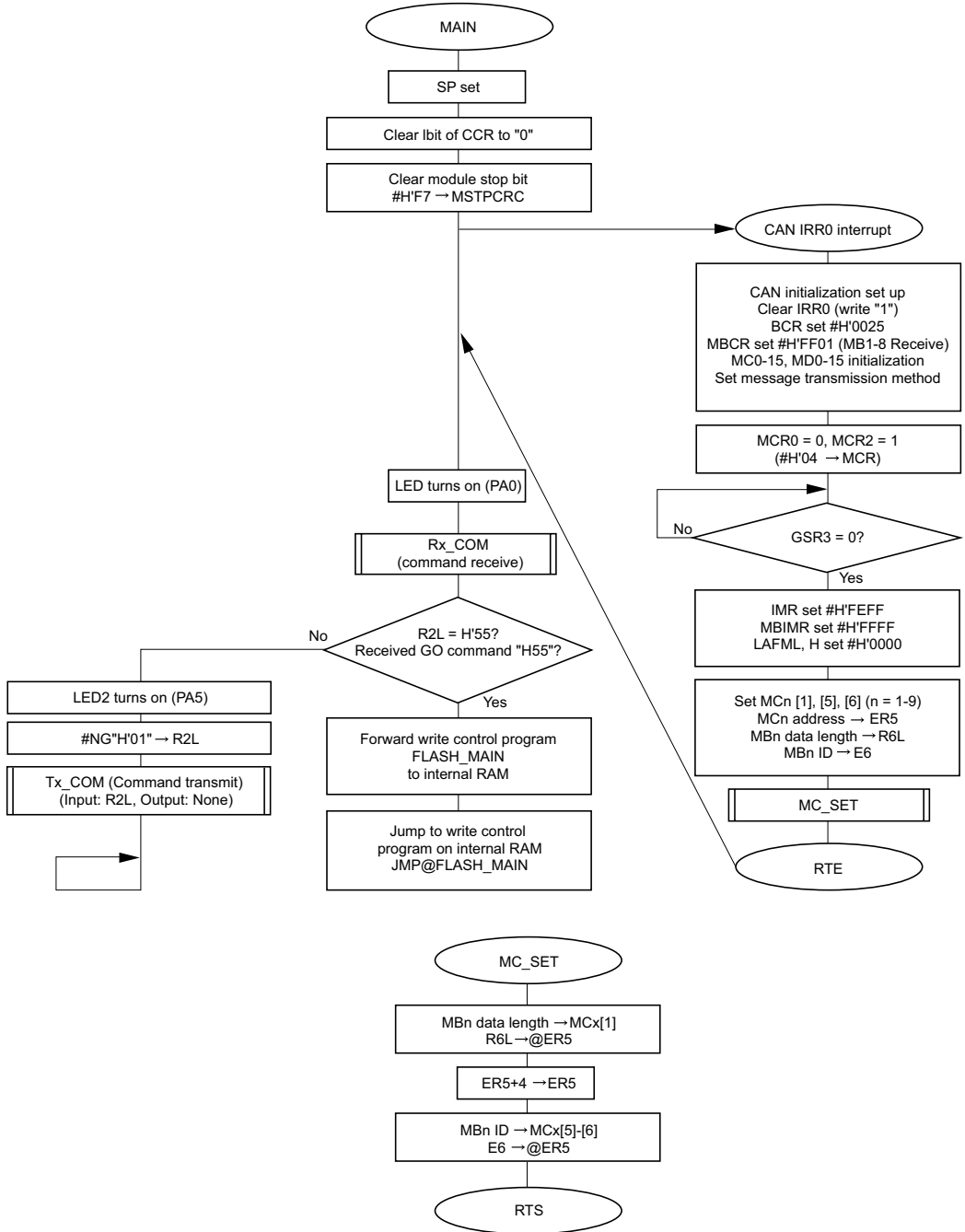
**(5) Explanation of RAM on use**

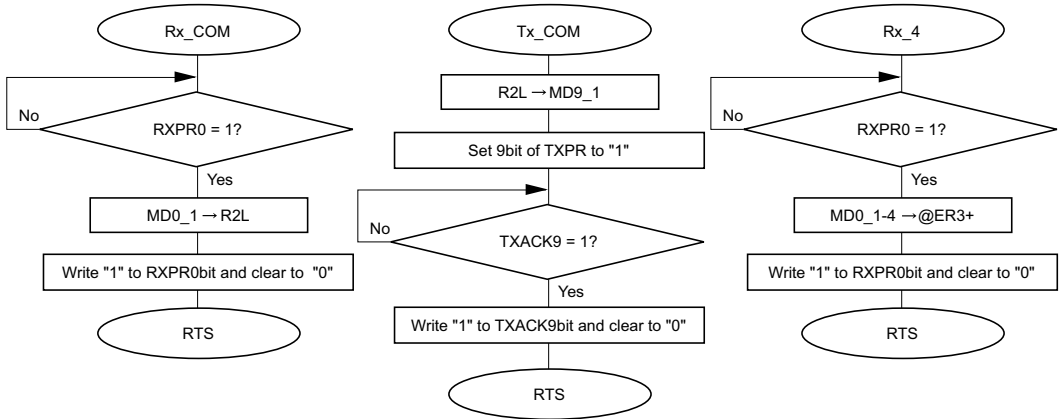
Table 4.3 shows internal RAM used for store area of write control program which is arguments or returned value and those work area

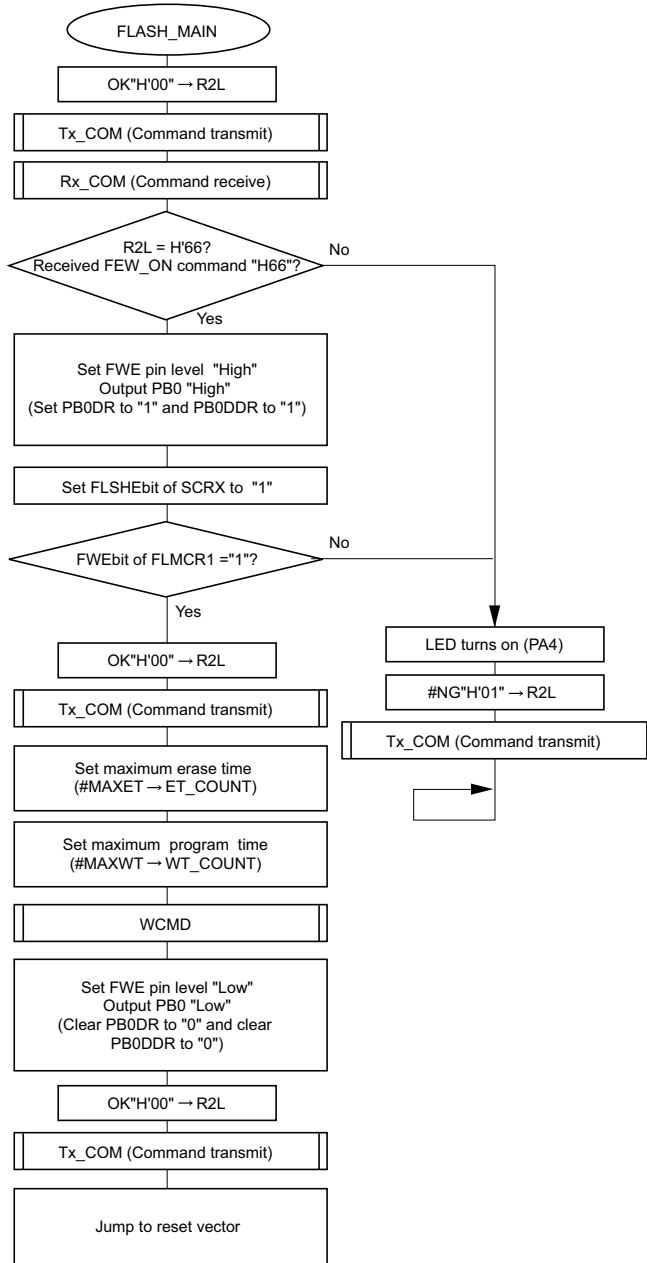
**Table 4.3 Explanation of RAM on use**

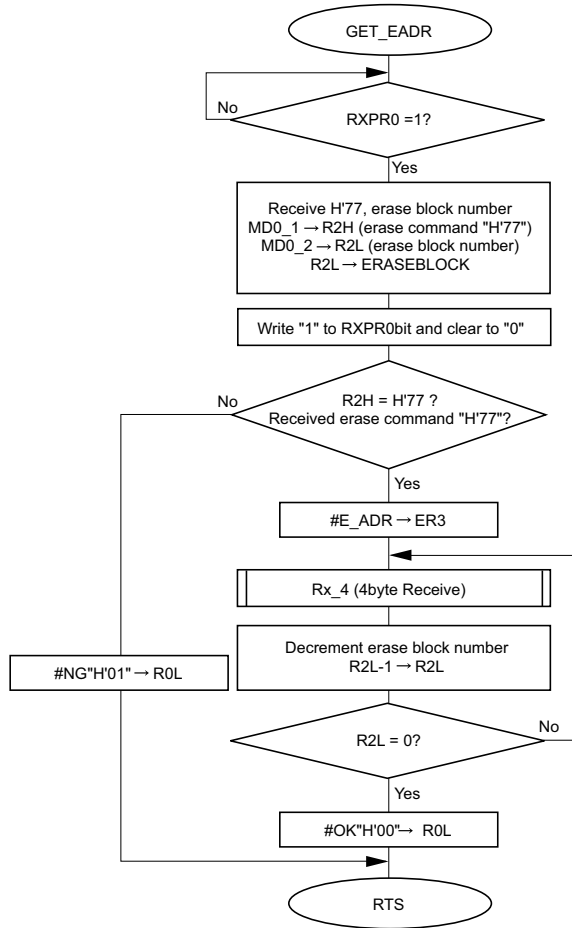
<b>Label</b>	<b>Function</b>	<b>Data Length</b>	<b>Module</b>
ERASEBLOCK	Store erase block number (Returned value)	1 byte	GET_EADR
E_ADR	Erase initial address (Returned value)	4-48 byte	GET_EADR
BLK_NO	Store erase object block number (Arguments, Returned value)	1 byte	BLK_CHECK, FERASE
EVF_ST	Store erase verify initial address (Arguments, Returned value)	4 byte	BLK_CHECK, FERASEVF
EVF_ED	Store erase verify last address (Arguments, Returned value)	4 byte	BLK_CHECK, FERASEVF
W_ADR	Store program initial address (Arguments, Returned value)	4 byte	GET_WADR, FWRITE_128, FWRITE
RESTSIZE	Store program data size (Arguments, Returned value)	4 byte	GET_WADR, GET_BUFFER
W_BUFF	Store 128byte write program data (Arguments, Returned value)	128 byte	RX_128, FWRITE_128, FWRITEVF
BUFF	Store reprogram arithmetic result (Arguments, Returned value)	128 byte	FWRITE_128, FWRITEVF
OW_BUFF	Store additional program arithmetic result (Arguments, Returned value)	128 byte	FWRITEVF

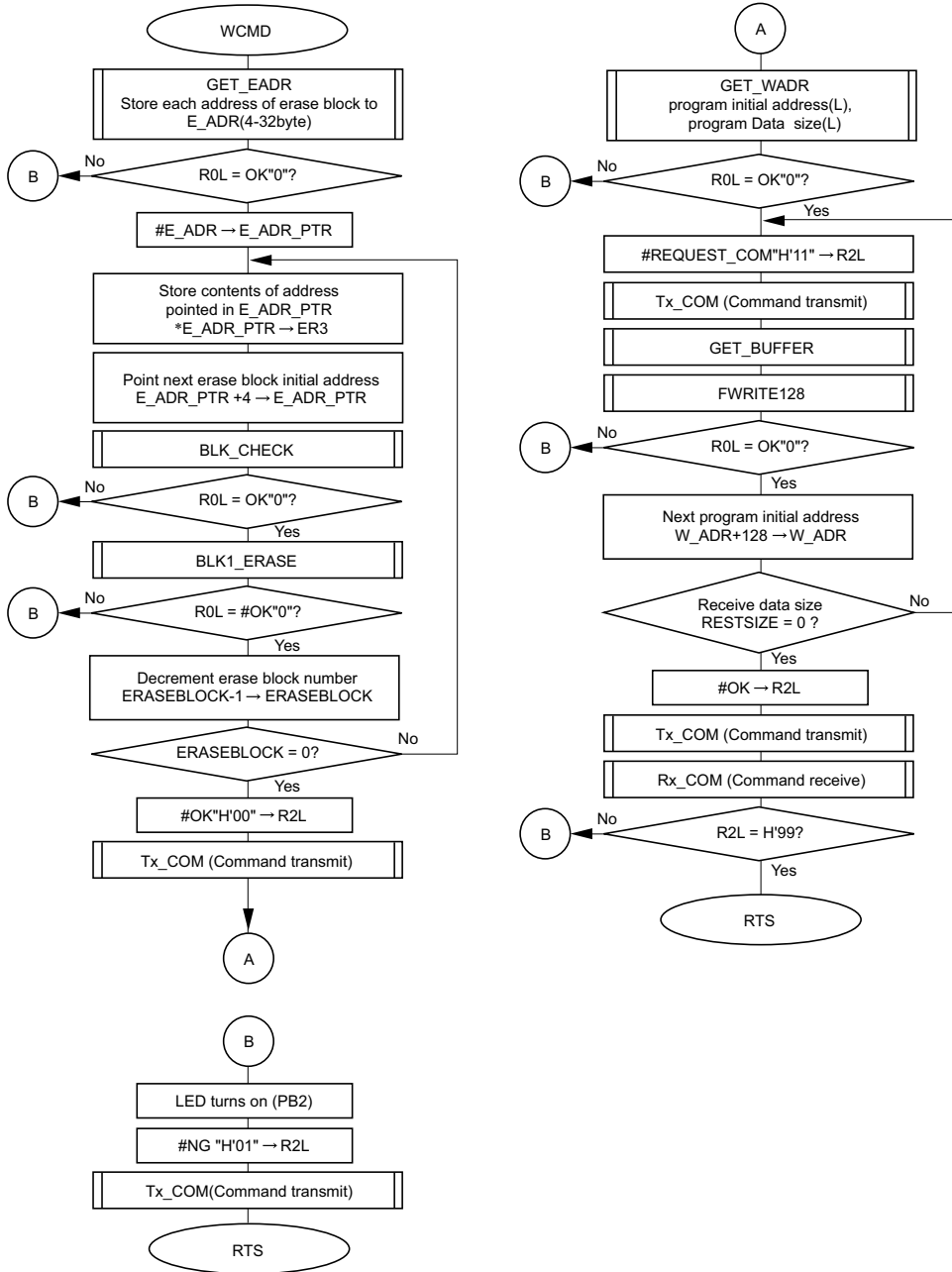
## 5. Flow Chart

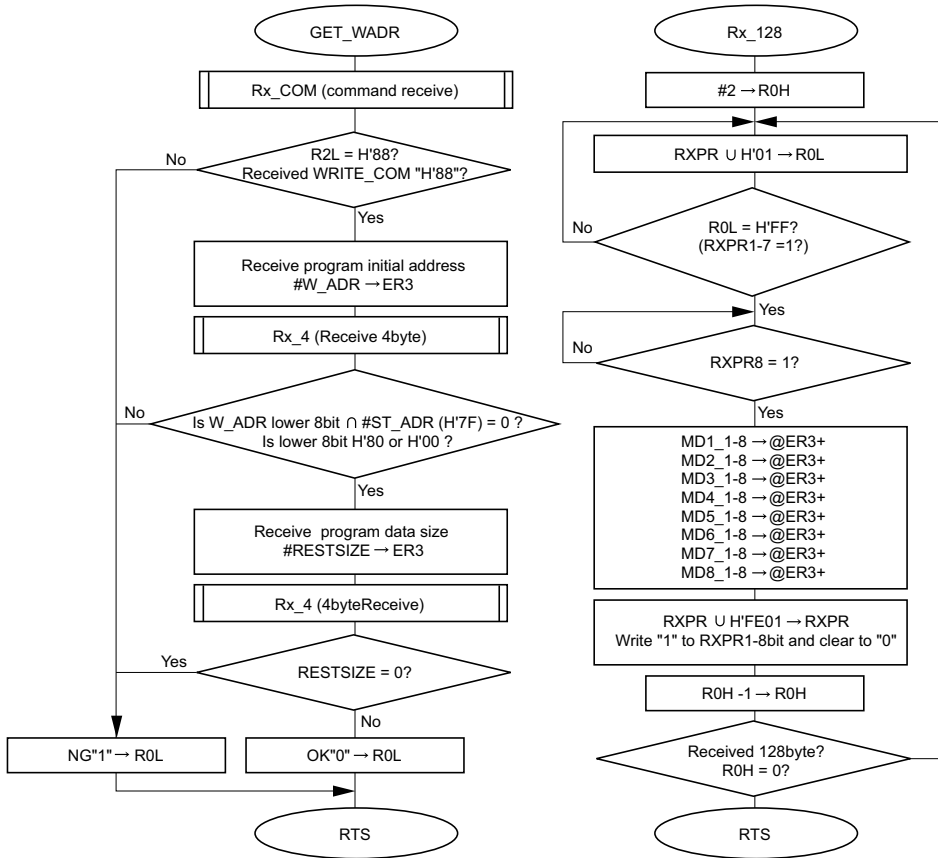


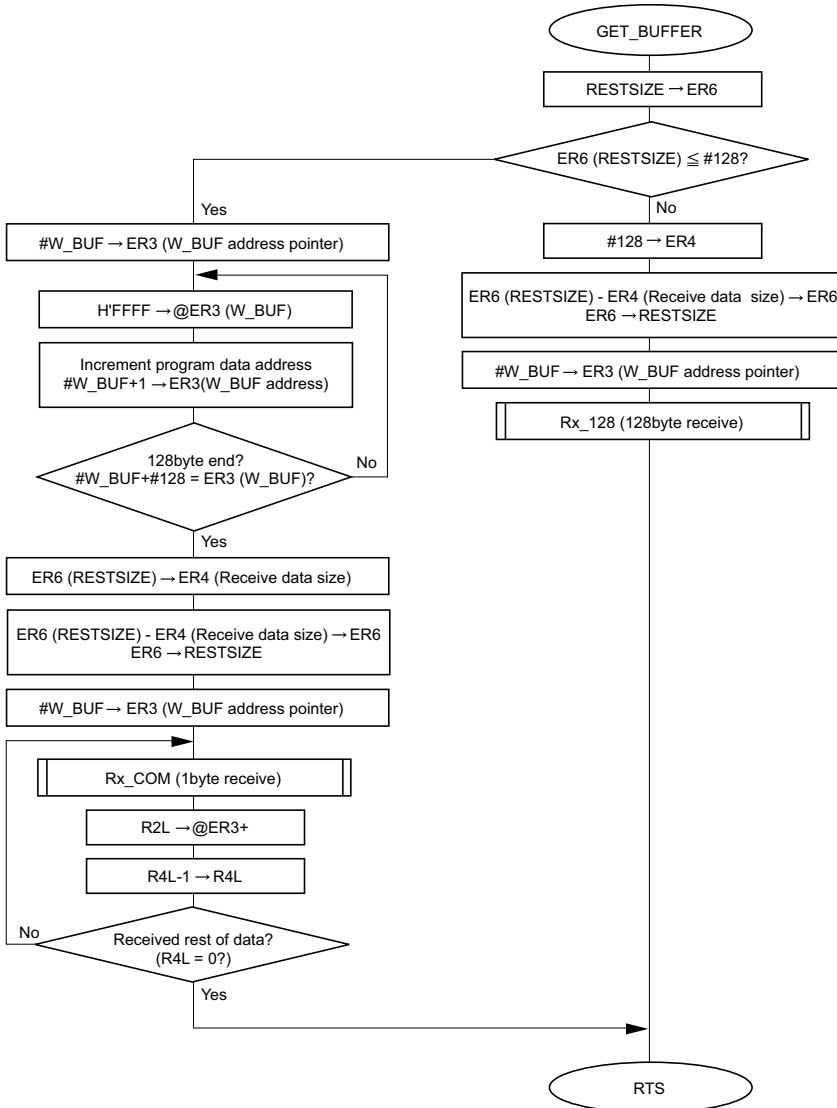


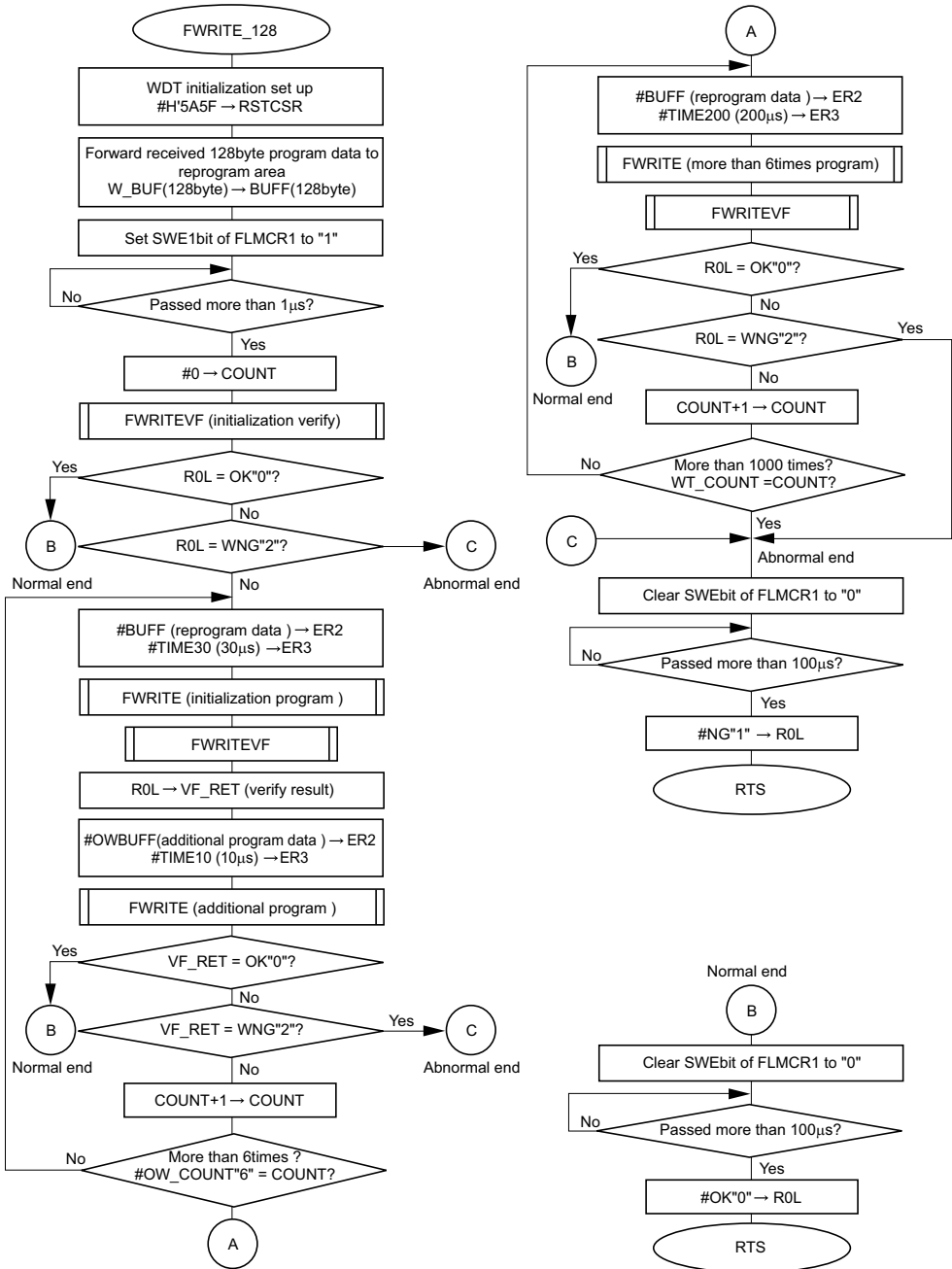


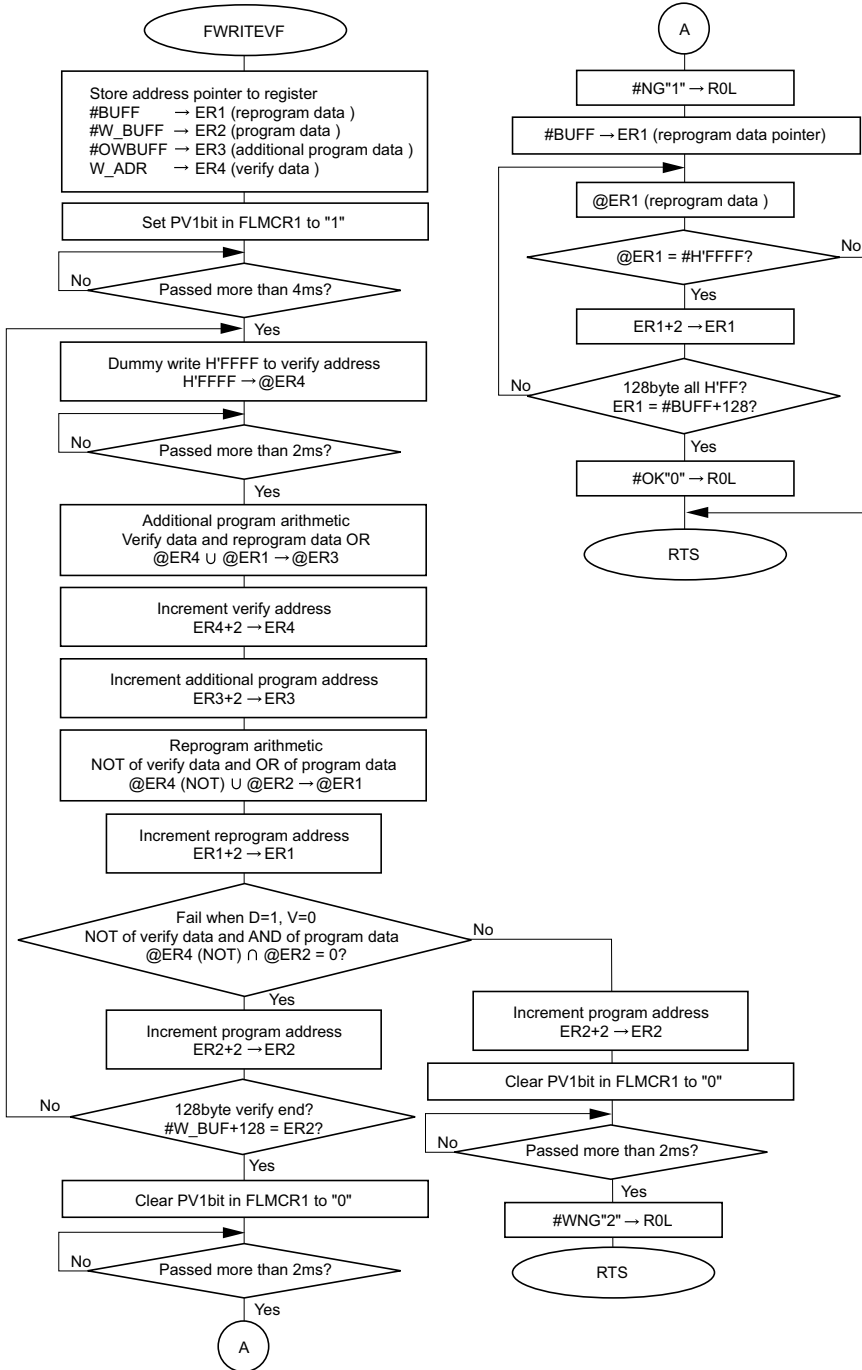


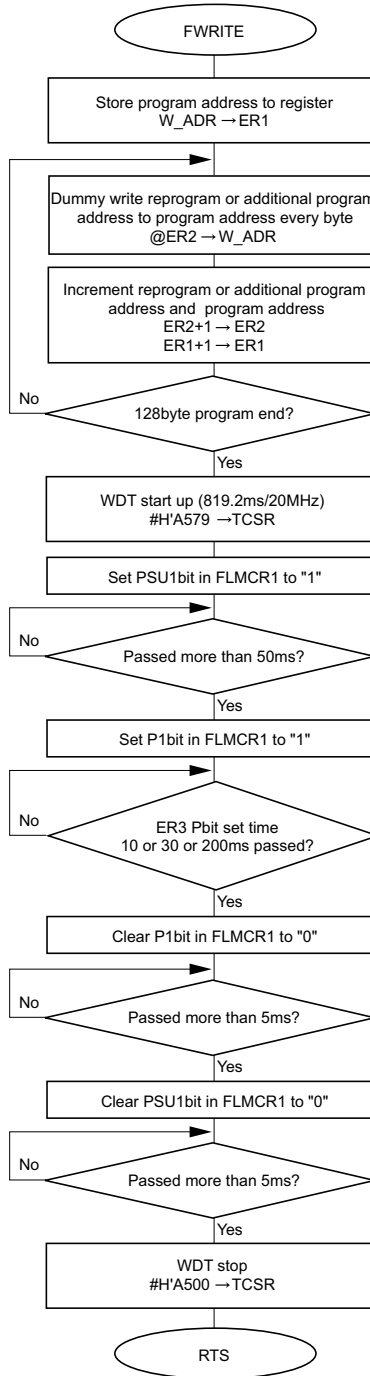


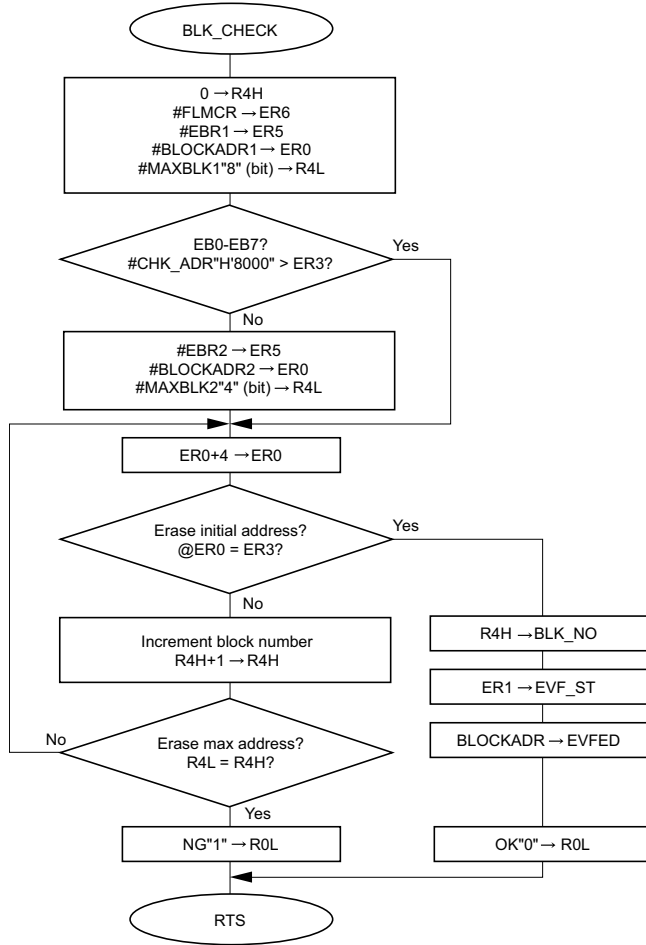


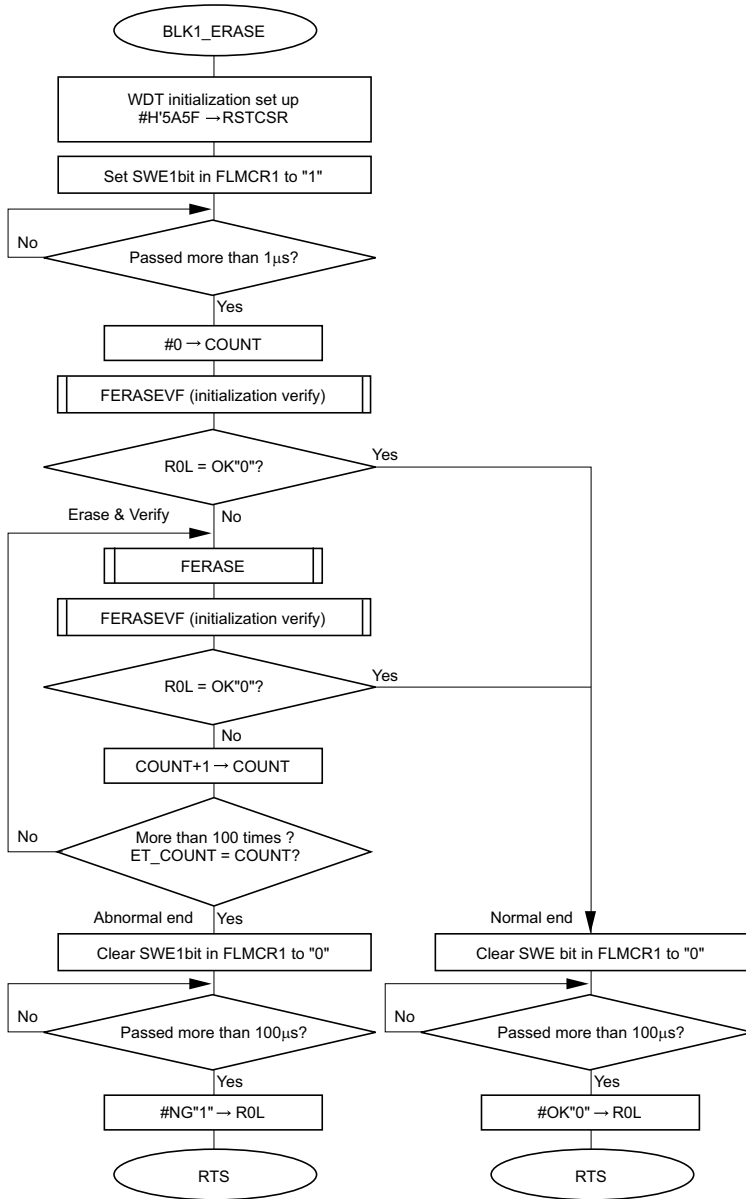


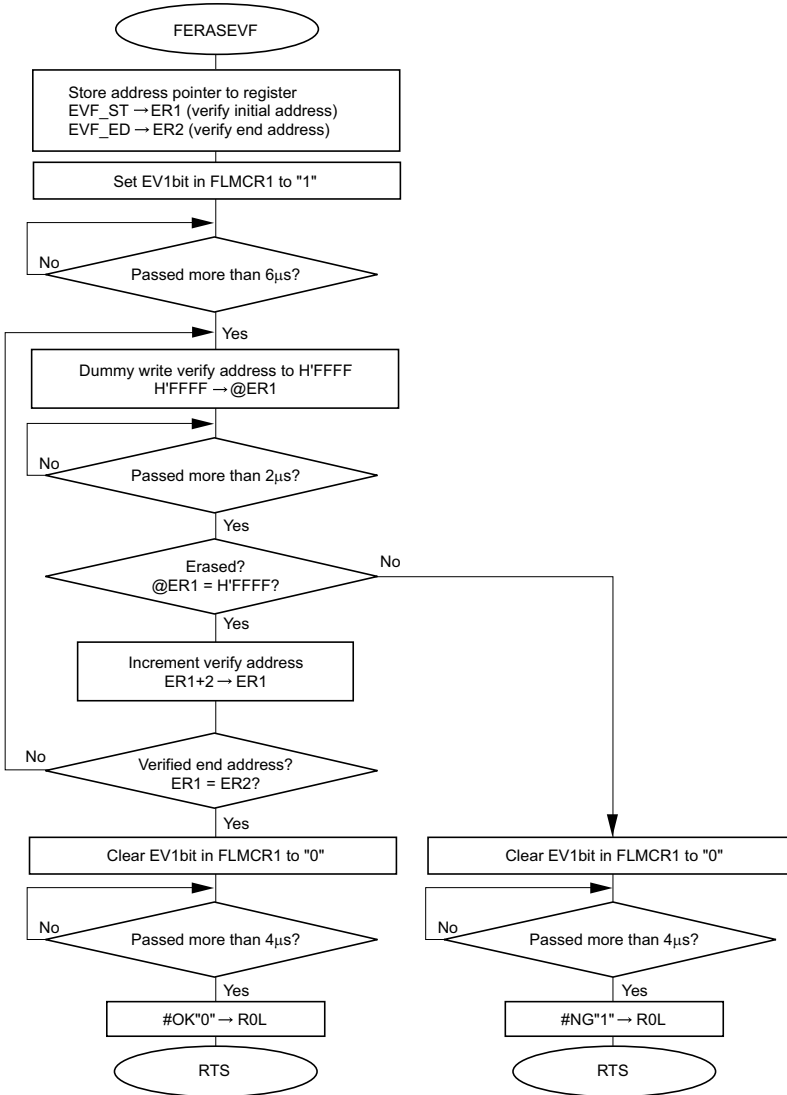


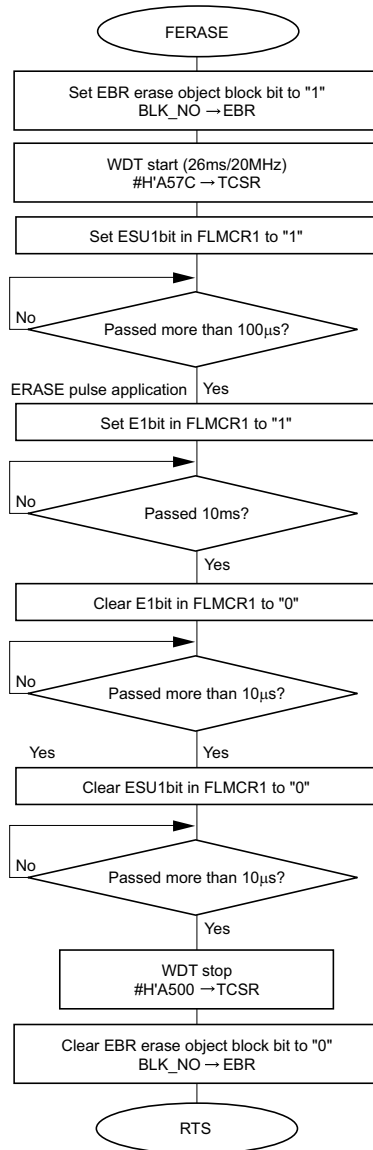












## 6. Hardware Explanation

### (1) CAN forward format

Figure15 shows data format specifications.

In this reprogramming example, standard format (Identifier:11bit)is used.

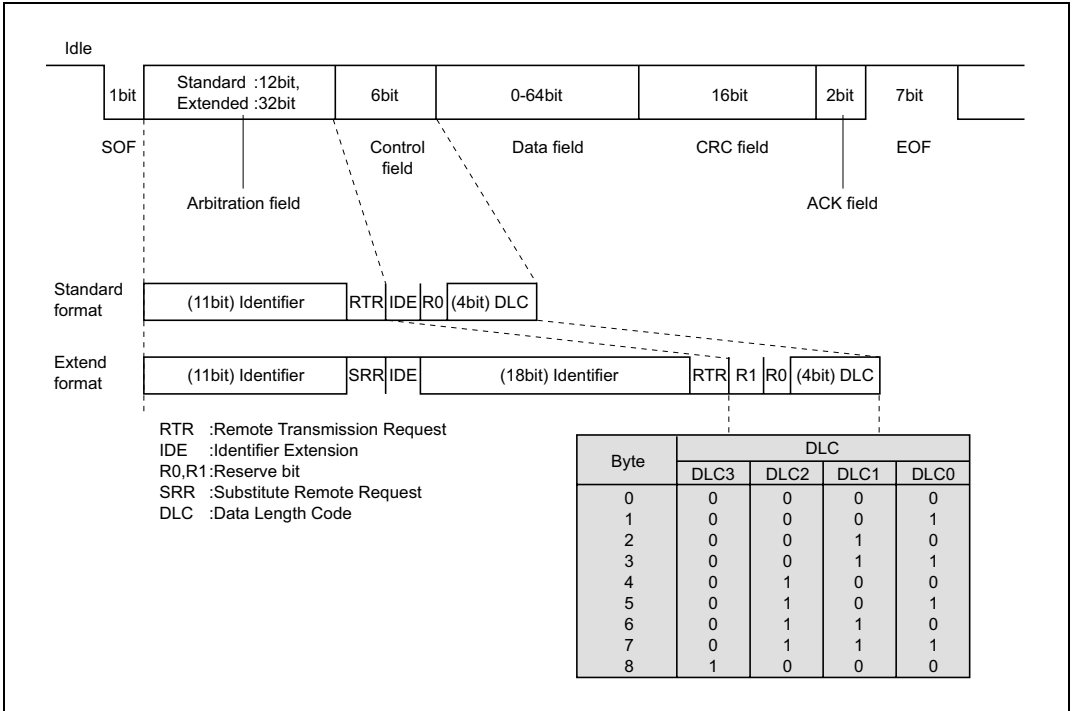
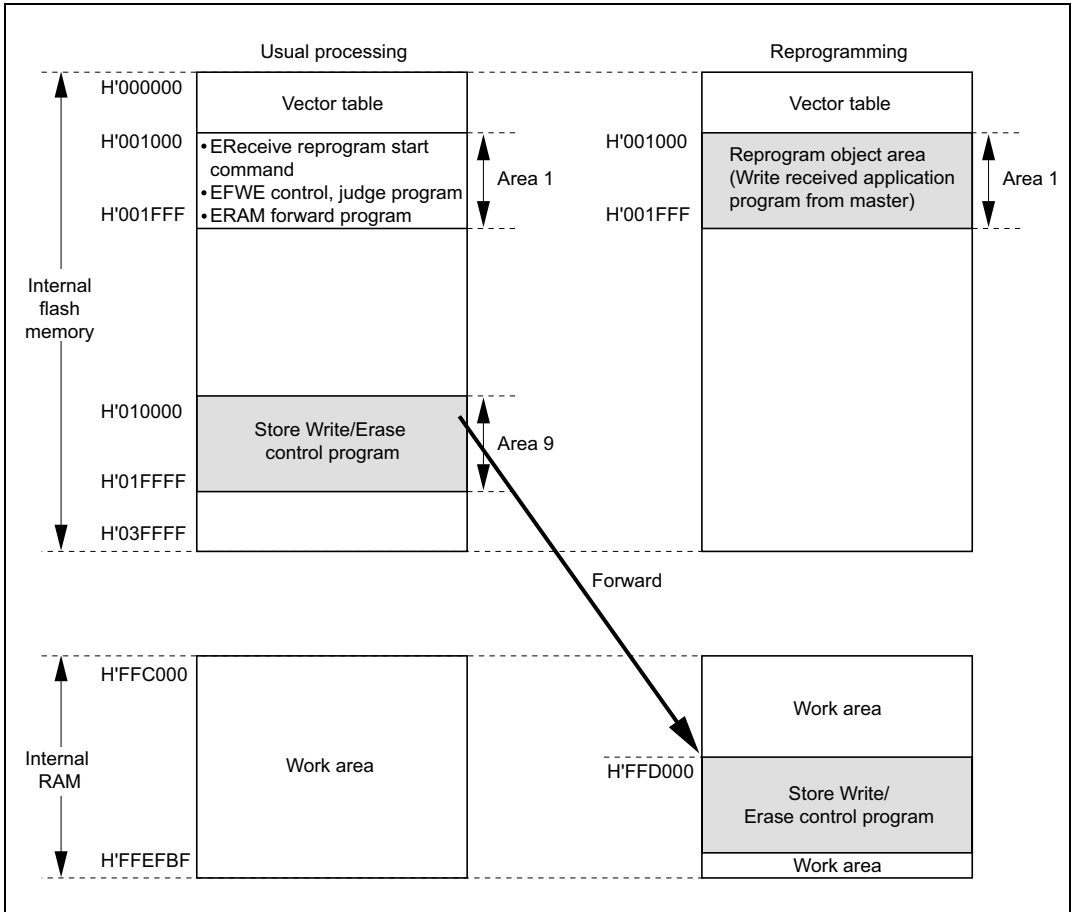


Figure 6.1 Data format

**(2) Memory map**

H8S/2623F Memory map is shown in figure6.2 (usual processing/reprogramming).

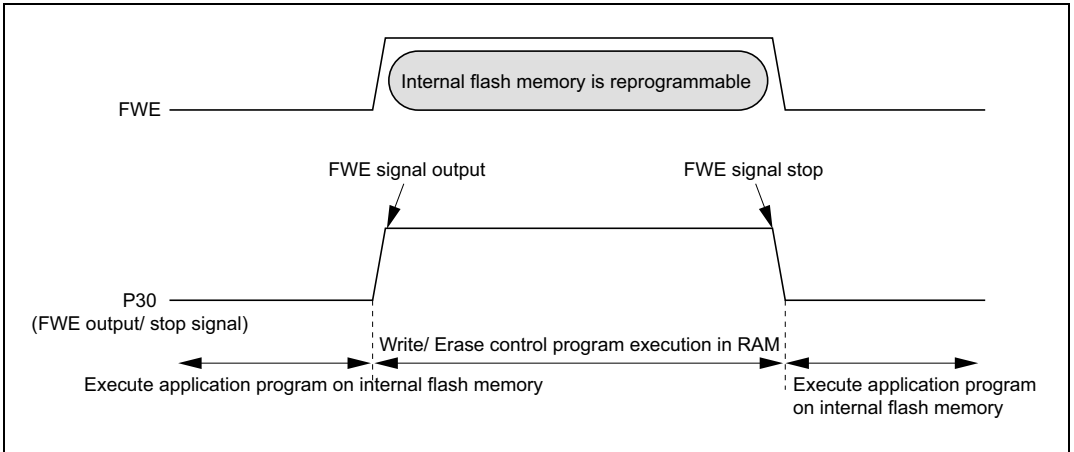


**Figure 6.2 Memory map**

**Note:** Store area, execute area of Write/Erase control program in figure 6.2 is one of example

**(3) FWE output/stop**

Write/ Erase control program is a timing shown in figure6.3 and controls output/stop of FWE pin.



**Figure 6.3 FWE output/Stop timing**

7. Circuit Figure

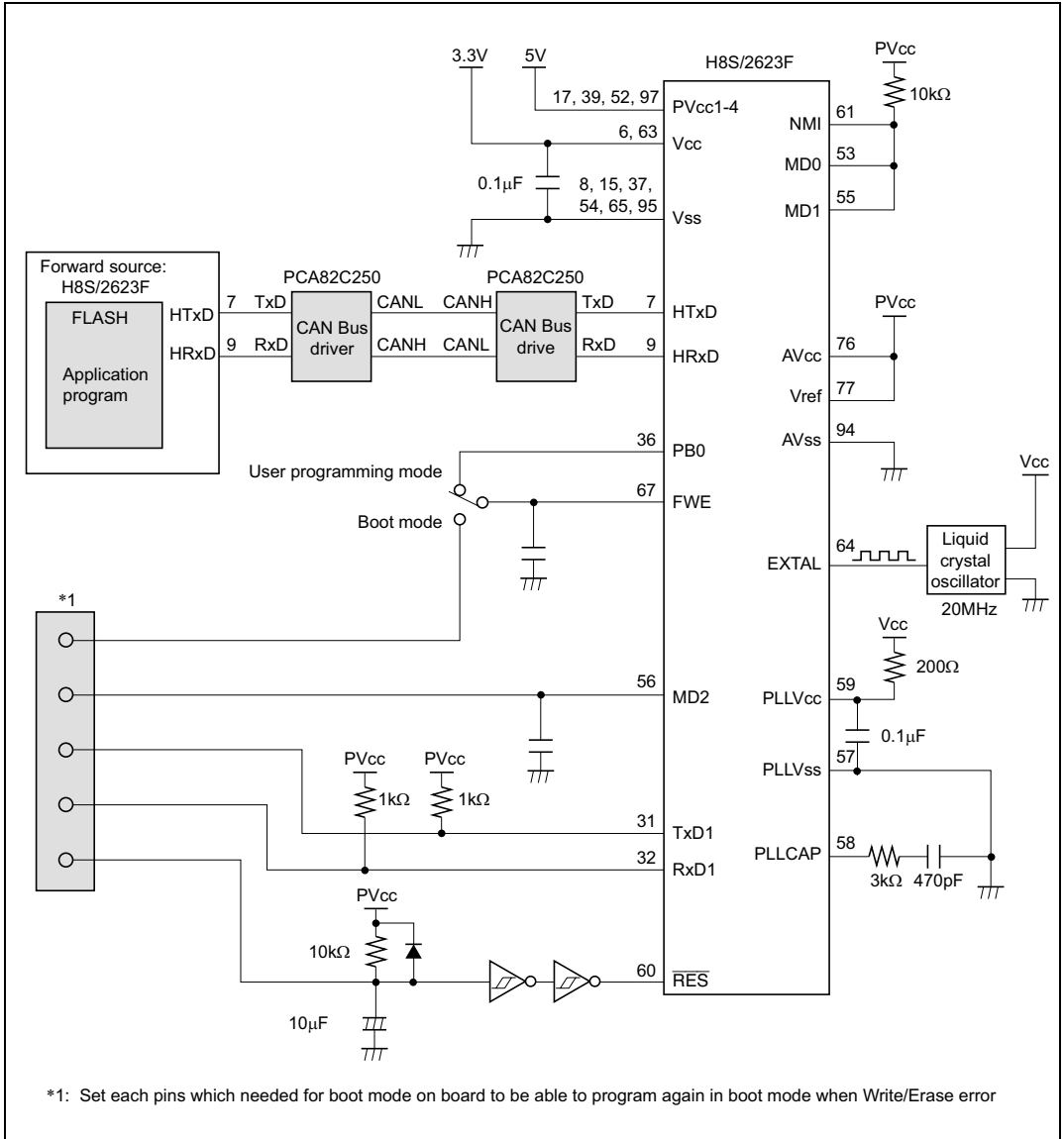


Figure 7.1 Onboard Programming circuit with CAN

## 8. Programming List

```

; *****
; * H8S/2623F *
; * EXAMPLE PROGRAM FOR "CAN F-ZTAT WRITING CONTROL PROGRAM (VER1.0)" *
; * MICROSOFT WINDOWS OPERATING SYSTEM *
; *****
.CPU          2600A   : 24MHZ
.EQU         20*100  ; 20MHZ
;
;*****  WAIT TIME  *****
WLOOP1      .EQU      1*MHZ/400+1   ; LOOP WAIT TIME
WLOOP2      .EQU      2*MHZ/400+1
WLOOP4      .EQU      4*MHZ/400+1
WLOOP5      .EQU      5*MHZ/400+1
WLOOP6      .EQU      6*MHZ/400+1
WLOOP10     .EQU      10*MHZ/400+1
WLOOP30     .EQU      30*MHZ/400+1
WLOOP50     .EQU      50*MHZ/400+1
WLOOP100    .EQU      100*MHZ/400+1
TIME10      .EQU      10*MHZ/400    ; WRITE WAIT TIME
TIME30      .EQU      30*MHZ/400    ; WRITE WAIT TIME
TIME200     .EQU      200*MHZ/400   ; WRITE WAIT TIME
TIME10000   .EQU      10000*MHZ/400 ; ERASE WAIT TIME
;*****  H8S/2623F REGISTER *****
FLMCR1      .EQU      H'FFFFA8      ; FLASH MEMORY CONTROL REGISTER 1
SWE:        .EQU      6
ESU:        .EQU      5
PSU:        .EQU      4
EV:         .EQU      3
PV:         .EQU      2
E:          .EQU      1
P:          .EQU      0
FLMCR2      .EQU      H'FFFFA9      ; FLASH MEMORY CONTROL REGISTER 2
EBR1        .EQU      H'FFFFAA      ; OBJECT BLOCK DESIGNATED REGISTER 1
EBR2        .EQU      H'FFFFAB      ; OBJECT BLOCK DESIGNATED REGISTER 2
SCRX        .EQU      H'FFFFB4      ; SERIAL CONTROL REGISTER X
FLSHE:      .EQU      3
;*****  MODULE STOP RESISUTOR *****
MSTPCRA     .EQU      H'FFFDE8      ; MODULE STOP CONTROL REGISTER A
MSTPCRB     .EQU      H'FFFDE9      ; MODULE STOP CONTROL REGISTER B
MSTPCRC     .EQU      H'FFFDEA      ; MODULE STOP CONTROL REGISTER C
;*****  WDT CN.0 *****
TCSR        .EQU      H'FFF74       ; WATCH DOG TIMER REGISTER
RSTCSR      .EQU      H'FFF76       ; RESET CONTROL REGISTER
;*****  CAN *****
MCR          .EQU      H'FFF800     ; MASTER CONTROL REGISTER
GSR          .EQU      H'FFF801     ; GENERAL STATUS REGISTER
BCR          .EQU      H'FFF802     ; BIT CONFIGURATION REGISTER
BCR_L        .EQU      H'FFF803     ; BIT CONFIGURATION REGISTER_LOW
MBCR         .EQU      H'FFF804     ; MAILBOX CONFIGURATION REGISTER
MBCR_L       .EQU      H'FFF805     ; MAILBOX CONFIGURATION REGISTER_LOW
TXPR         .EQU      H'FFF806     ; TRANSMIT WAIT REGISTER
TXPR_L       .EQU      H'FFF807     ; TRANSMIT WAIT REGISTER_LOW
TXCR         .EQU      H'FFF808     ; TRANSMIT WAIT CANCEL REGISTER
TXCR_L       .EQU      H'FFF809     ; TRANSMIT WAIT CANCEL REGISTER_LOW

```

```

TXACK          .EQU    H'FFF80A          ; TRANSMIT ACKNOWLEDGE REGISTER
TXACK_L        .EQU    H'FFF80B          ; TRANSMIT ACKNOWLEDGE REGISTER_LOW
ABACK          .EQU    H'FFF80C          ; ABORT ACKNOWLEDGE REGISTER
RXPR           .EQU    H'FFF80E          ; RECEIVE COMPLETE REGISTER
RXPR_L         .EQU    H'FFF80F          ; RECEIVE COMPLETE REGISTER_LOW
RFPR           .EQU    H'FFF810          ; REMOTE REQUEST WAIT REGISTER
RFPR_L         .EQU    H'FFF811          ; REMOTE REQUEST WAIT REGISTER_LOW
IRR            .EQU    H'FFF812          ; INTERRUPT REGISTER
IRR_L          .EQU    H'FFF813          ; INTERRUPT REGISTER_LOW
MBIMR          .EQU    H'FFF814          ; MAILBOX INTERRUPT MASK REGISTER
MBIMR_L        .EQU    H'FFF815          ; MAILBOX INTERRUPT MASK REGISTER_LOW
IMR            .EQU    H'FFF816          ; INTERRUPT MASK REGISTER
IMR_L          .EQU    H'FFF817          ; INTERRUPT MASK REGISTER
REC            .EQU    H'FFF818          ; RECEIVE ERROR COUNTER
TEC            .EQU    H'FFF819          ; TRANSMIT ERROR COUNTER
UMSR           .EQU    H'FFF81A          ; UNREAD MESSAGE STATUS REGISTER
UMSR_L         .EQU    H'FFF81B          ; UNREAD MESSAGE STATUS REGISTER_LOW
LAFML          .EQU    H'FFF81C          ; LOCAL ACCEPTANCE FILTER MASK L
LAFMH          .EQU    H'FFF81E          ; LOCAL ACCEPTANCE FILTER MASK H
;
MC0_1          .EQU    H'FFF820          ; MESSAGE CONTROL0
MC1_1          .EQU    H'FFF828          ; MESSAGE CONTROL1
MC2_1          .EQU    H'FFF830          ; MESSAGE CONTROL2
MC3_1          .EQU    H'FFF838          ; MESSAGE CONTROL3
MC4_1          .EQU    H'FFF840          ; MESSAGE CONTROL4
MC5_1          .EQU    H'FFF848          ; MESSAGE CONTROL5
MC6_1          .EQU    H'FFF850          ; MESSAGE CONTROL6
MC7_1          .EQU    H'FFF858          ; MESSAGE CONTROL7
MC8_1          .EQU    H'FFF860          ; MESSAGE CONTROL8
MC9_1          .EQU    H'FFF868          ; MESSAGE CONTROL9
MC10_1         .EQU    H'FFF870          ; MESSAGE CONTROL10
MC11_1         .EQU    H'FFF878          ; MESSAGE CONTROL11
MC12_1         .EQU    H'FFF880          ; MESSAGE CONTROL12
MC13_1         .EQU    H'FFF888          ; MESSAGE CONTROL13
MC14_1         .EQU    H'FFF890          ; MESSAGE CONTROL14
MC15_1         .EQU    H'FFF898          ; MESSAGE CONTROL15
MC_END         .EQU    H'FFF8A0          ; END OF MESSAGE CONTROL
;
MD0_1          .EQU    H'FFF8B0          ; MESSAGE DATA0
MD1_1          .EQU    H'FFF8B8          ; MESSAGE DATA1
MD2_1          .EQU    H'FFF8C0          ; MESSAGE DATA2
MD3_1          .EQU    H'FFF8C8          ; MESSAGE DATA3
MD4_1          .EQU    H'FFF8D0          ; MESSAGE DATA4
MD5_1          .EQU    H'FFF8D8          ; MESSAGE DATA5
MD6_1          .EQU    H'FFF8E0          ; MESSAGE DATA6
MD7_1          .EQU    H'FFF8E8          ; MESSAGE DATA7
MD8_1          .EQU    H'FFF8F0          ; MESSAGE DATA8
MD9_1          .EQU    H'FFF8F8          ; MESSAGE DATA9
MD10_1         .EQU    H'FFF900          ; MESSAGE DATA10
MD11_1         .EQU    H'FFF908          ; MESSAGE DATA11
MD12_1         .EQU    H'FFF910          ; MESSAGE DATA12
MD13_1         .EQU    H'FFF918          ; MESSAGE DATA13
MD14_1         .EQU    H'FFF920          ; MESSAGE DATA14
MD15_1         .EQU    H'FFF928          ; MESSAGE DATA15
MD_END         .EQU    H'FFF930          ; END OF MESSAGE DATA

```

```

;***** PORT *****
PADDR      .EQU    H'FFFE39      ; PORTA DATA DIRECTION REGISTER
PBDDR      .EQU    H'FFFE3A      ; PORTB DATA DIRECTION REGISTER
PADR       .EQU    H'FFFF09      ; PORTA DATA REGISTER
PBDR       .EQU    H'FFFF0A      ; PORTB DATA REGISTER
PORTA      .EQU    H'FFFFB9      ; PORTA REGISTER
PORTB      .EQU    H'FFFFBA      ; PORTB REGISTER
PAPCR      .EQU    H'FFFE40      ; PORTA MOS PULL-UP CONTROL REGISTER
PBPCR      .EQU    H'FFFE41      ; PORTB MOS PULL-UP CONTROL REGISTER
PAODR      .EQU    H'FFFE47      ; PORTA OPEN-DRAIN CONTROL REGISTER
PBODR      .EQU    H'FFFE48      ; PORTB OPEN-DRAIN CONTROL REGISTER
;***** RAM ADDRESS POINT *****
*****
USER_TOP    .EQU    H'FFC000      ; USER RAM TOP
USER_RAM    .EQU    H'FFD000      ; WORK RAM TOP
USER_SP     .EQU    H'FFE000      ; USER SP
;**** FLAG DEFINITIONS *****
CHK_ADR     .EQU    H'08000       ; START ADDRESS FOR EBR2
ST_ADR      .EQU    H'7F         ; WRITE START ADDRESS (128-BYTE ;WRITE)
OK          .EQU    H'0          ; OK FLAG
NG          .EQU    H'1          ; ERR FLAG
WNG         .EQU    H'2          ; WRITE ERR
;**** COMMAND DEFINITIONS*****
GO_COM      .EQU    H'55         ; START REWRITE COMMAND
OK_COM      .EQU    H'00         ; OK COMMAND
FWE_ON_COM  .EQU    H'66         ; ENABLE FWE TERMINAL COMMAND
ERASE_COM   .EQU    H'77         ; ERASE COMMAND
WRITE_COM   .EQU    H'88         ; WRITE COMMAND
REQUEST_COM .EQU    H'11         ; TRANSMIT REQUEST COMMAND
FWE_OFF_COM .EQU    H'99         ; DISABLE FWE TERMINAL COMMAND
NG_COM      .EQU    H'01         ; NG COMMAND
;***** FLASH CONSTANTS *****
MAXWT       .EQU    1000         ; MAX WRITE COUNT
MAXET       .EQU    100         ; MAX ERASE COUNT
OW_COUNT    .EQU    6           ; OVER WRITE COUNT
MAXBLK1     .EQU    8           ; MAX BLOCK 1
MAXBLK2     .EQU    4           ; MAX BLOCK 2
;
; //H8S/2623F WRITE/ERASE PROGRAM RAM AREA//
        .SECTION  RAM,DATA,LOCATE = USER_TOP
        .ALIGN  2
W_ADR      .RES.B  4             ; WRITE ADDRESS AREA
LAST_JMP   .RES.B  4             ; LAST JMP ADDRESS
W_BUF      .RES.B  128          ; WRITE DATA AREA
BUFF       .RES.B  128          ; RETRY WRITE DATA AREA
OWBUFF     .RES.B  128          ; OVER WRITE DATA ARIA
COUNT     .RES.B  2            ; W_COUNT,E_COUNT
ET_COUNT   .RES.B  2            ; MAX E_COUNT
WT_COUNT   .RES.B  2            ; MAX W_COUNT
EVF_ST     .RES.B  4             ; ERASE VERIFY START ADDRESS
EVF_ED     .RES.B  4             ; ERASE VERIFY END ADDRESS
BLK_NO     .RES.B  1             ; ERASE BIT NUMBER
VF_RET     .RES.B  1             ; VERIFY CHECK
        .ALIGN  2
RESTSIZE   .RES.B  4             ; WRITE DATA SIZE
E_ADR      .RES.B  48           ; BLOCK ERASE ADDRESS

```

```

E_ADR_PTR      .RES.B  4          ; BLOCK ERASE ADDRESS POINTER
ERASEBLOCK     .RES.B  1          ; ERASE BLOCK NUMBER
;
; *****
; * NAME / VECTOR TABLE                      *
; * FUNCTION / -                              *
; * INPUT / -                                  *
; * OUTPUT / -                                *
; *****
        .SECTION  VECT,DATA,LOCATE=H'000000
        .DATA.L  MAIN
        .ORG     H'0001A0
        .DATA.L  MAIN
        .DATA.L  IRR0
        .DATA.L  MAIN
        .DATA.L  MAIN
        .DATA.L  MAIN
; *****
; * NAME / MAIN ROUTINE                      *
; * FUNCTION / AFTER RECEIVING GO COMMAND FROM MAIN, RUNS RAM TRANSFER PROGRAM *
; * INPUT / -                                  *
; * OUTPUT / -                                *
; *****
        .SECTION  PROG1,CODE,LOCATE = H'001000
MAIN     .EQU     $
        MOV.L    #USER_SP,ER7          ; INITIALIZED STACK
        MOV.B    #H'FF,R0L             ;
        MOV.B    R0L,@PADR             ; PORT A HIGH LEVEL
        MOV.B    R0L,@PADDR            ; PORT A OUTPUT TERMINAL
        LDC.B    #H'00,CCR             ; INTERRUPT PERMISSION BY CLEARING I BIT
        MOV.B    #H'F7,R0L             ;
        MOV.B    R0L,@MSTPCRC          ; DISABLE CAN MODULE STOP
        BCLR     #0,@PADR              ; PORT A0 "LOW" OUTPUT
        JSR      @RX_COM               ; COMMAND RECEIVED FROM MASTER
        CMP.B    #H'55,R2L             ; OVERWRITE COMMAND RECEIVED FROM MASTER?
        BEQ      TRANS_RAM             ;
        BCLR     #5,@PADR              ; PORT A5 "LOW" OUTPUT
        MOV.B    #NG_COM,R2L           ;
        JSR      @TX_COM               ; TRANSMIT NG COMMAND TO MASTER
ERROR_1  NOP                          ;
        BRA     ERROR_1                ; INFINITE LOOP
;
TRANS_RAM:
MOV.L    #FLASH_MAIN,ER5              ; START ADDRESS OF TRANSFER PROGRAM
        MOV.L    #USER_RAM,ER6         ; START ADDRESS OF TRANSFER RAM
        MOV.W    #ENDFILE-FLASH_MAIN,R4 ; END ADDRESS OF TRANSFER PROGRAM
        EEPMOV.W ; TRANSFER CONTROL PROGRAM TO RAM
        JMP     @USER_RAM              ; JMP TO CONTROL PROGRAM IN RAM
;
; *****
; * NAME / CAN IRR0 INTERRUPT                *
; * FUNCTION / INITIAL SETTINGS FOR CAN      *
; *****
IRR0     .EQU     $
        BSET     #0,@IRR               ; WRITE "1" OR CLEAR "0" IRR0
        MOV.W    #H'0025,R5           ;

```

```

MOV.W   R5,@BCR           ; SET BAUD RATE TO 1Mbps (at 20MHz)
MOV.W   #H'FF01,R5       ;
MOV.W   R5,@MBCR         ; SET MAILBOX TRANSMIT RECEIVE
;
MOV.L   #MC0_1,ER5       ; MC0-15INITIALIZATION (WRITE ALL H'00)
SUB.L   ER6,ER6           ;
CLEAR_1 MOV.L   ER6,@ER5   ;
ADDS   #4,ER5            ; MC ADDRESS INCREMENT
CMP.L   #MC_END,ER5     ; FINISHED UP TO MC15_8?
BNE    CLEAR_1          ;
;
MOV.L   #MD0_1,ER5       ; MD0-15 INITIALIZATION (WRITE ALL H'00)
CLEAR_2 MOV.L   ER6,@ER5   ;
ADDS   #4,ER5            ; MD ADDRESS INCREMENT
CMP.L   #MD_END,ER5     ; FINISHED UP TO MD15_8?
BNE    CLEAR_2          ;
;
MOV.B   #H'04,R6L        ;
MOV.B   R6L,@MCR         ; TRANSMIT MODE: CHANGE TO CAN NORMAL MODE, MB ORDER
LOOP_1  BTST   #3,@GSR    ; CAN NORMAL MODE? GSR3=0?
BNE    LOOP_1           ;
;
SUB.L   ER6,ER6          ;
MOV.L   ER6,@LAFML       ; MB0 ID ALL MATCH
MOV.W   #H'FFFF,R6       ;
MOV.W   R6,@MBIMR        ; INTERRUPT REQUEST TO CPU ;DISALLOWED
MOV.W   #H'FEFF,R6       ;
MOV.W   R6,@IMR          ; INTERRUPT REQUEST TO CPU ;DISALLOWED
;
MOV.L   #MC0_1,ER5       ;
MOV.B   #H'08,R6L        ; MB0 DATA LENGTH 8 BYTES
BSR    MC_SET            ; MD0 ID "H'00"
MOV.L   #MC1_1,ER5       ;
MOV.B   #H'08,R6L        ; MB1 DATA LENGTH 8 BYTES
MOV.W   #H'2000,E6       ; MB1 ID "H'01"
BSR    MC_SET
MOV.L   #MC2_1,ER5       ;
MOV.B   #H'08,R6L        ; MB2 DATA LENGTH 8 BYTES
MOV.W   #H'4000,E6       ; MB2 ID "H'02"
BSR    MC_SET
MOV.L   #MC3_1,ER5       ;
MOV.B   #H'08,R6L        ; MB3 DATA LENGTH 8 BYTES
MOV.W   #H'6000,E6       ; MB3 ID "H'03"
BSR    MC_SET
MOV.L   #MC4_1,ER5       ;
MOV.B   #H'08,R6L        ; MB4 DATA LENGTH 8 BYTES
MOV.W   #H'8000,E6       ; MB4 ID "H'04"
BSR    MC_SET
MOV.L   #MC5_1,ER5       ;
MOV.B   #H'08,R6L        ; MB5 DATA LENGTH 8 BYTES
MOV.W   #H'A000,E6       ; MB5 ID "H'05"
BSR    MC_SET
MOV.L   #MC6_1,ER5       ;
MOV.B   #H'08,R6L        ; MB6 DATA LENGTH 8 BYTES

```

```

MOV.W    #H'C000,E6          ; MB6 ID "H'06"
BSR      MC_SET
MOV.L    #MC7_1,ER5
MOV.B    #H'08,R6L          ; MB7 DATA LENGTH 8 BYTES
MOV.W    #H'E000,E6        ; MB7 ID "H'07"
BSR      MC_SET
MOV.L    #MC8_1,ER5
MOV.B    #H'08,R6L          ; MB8 DATA LENGTH 8 BYTES
MOV.W    #H'0001,E6        ; MB8 ID "H'08"
BSR      MC_SET
MOV.L    #MC9_1,ER5
MOV.B    #H'01,R6L          ; MB9 DATA LENGTH 8 BYTES
MOV.W    #H'2001,E6        ; MB9 ID "H'09"
BSR      MC_SET
RTE

;
; *****
; * NAME / MESSAGE CONTROL REGISTER SET UP                               *
; * FUNCTION / SETS ID FOR MAILBOXES 0-9 AND DATA SIZES                *
; * INPUT / ER5 = START ADDRESS OF MESSAGE CONTROL REGISTER n          *
; *          R6L = SETTING VALUE FOR DATA LENGTHS                      *
; *          E6  = ID SETTING VALUES                                   *
; * OUTPUT / -                                                           *
; *****
MC_SET    .EQU    $
          MOV.B   R6L,@ER5          ; SET MBn DATA LENGTH
          ADDS   #4,ER5             ; MDn_5
          MOV.W  E6,@ER5           ; SET MBn ID
          RTS

;
; *****
; * NAME / MAIN FLASH WRITE ROUTINE                                     *
; * FUNCTION / CHECKS COMMANDS FROM MASTER, SETS NUMBER OF ERASE/WRITE *
; * INPUT / -                                                           *
; * OUTPUT / -                                                           *
; *****
        .SECTION  PROG2,CODE,LOCATE = H'10000
;
FLASH_MAIN .EQU    $
          MOV.B   #OK_COM,R2L      ;
          BSR    TX_COM            ; TRANSMIT OK COMMAND TO MASTER
          BSR    RX_COM            ; RECEIVE COMMAND FROM MASTER
          CMP.B  #FWE_ON_COM,R2L   ; RECEIVED ENABLE FWE TERMINAL COMMAND FROM
          ; MASTER?
          BEQ    SET_FWE_PIN       ;
          BRA    ERROR_2           ;

;
SET_FWE_PIN:
          MOV.B   #H'01,R0L        ;
          MOV.B   R0L,@PBDDR       ; PORT B0 OUTPUT
          BSET   #0,@PBDR          ; SET PORT B0 TO "HIGH" OUTPUT, FWE TERMINAL "ON"
          BSET.B #FLSHE,@SCRX     ; FLASH ENABLE
          BTST  #7,@FLMCR1        ; FWE=1?
          BNE    FWE_OK           ;

ERROR_2:
          BCLR  #4,@PADR          ; PORT A4 "LOW" OUTPUT

```

```

MOV.B #NG_COM,R2L ;
BSR TX_COM ; TRANSMIT NG COMMAND TO MASTER
ERROR_3
NOP ;
BRA ERROR_3 ; INFINITE LOOP
;
FWE_OK:
MOV.B #OK_COM,R2L ;
BSR TX_COM ; TRANSMIT OK COMMAND TO MASTER
MOV.W #MAXET,R0 ; SET MAX NUMBER OF ERASES
MOV.W R0,@ET_COUNT
MOV.W #MAXWT,R0 ; SET MAX NUMBER OF WRITES
MOV.W R0,@WT_COUNT
BSR WCMD ;
;
CLEAR_FWE_PIN:
BCLR #0,@PBDR ; SET PORT B0 TO "LOW" OUPUT AND FWE TERMINAL TO
; "OFF"
MOV.B #OK_COM,R2L ;
BSR TX_COM ; TRANSMIT OK COMMAND TO MASTER
SUB.L ERO,ER0 ;
JMP @ER0 ; JMP TO RESET
;
; *****
; * NAME / MAIN ROUTINE FOR WRITING, BLOCK ERASE *
; * FUNCTION / 128 BYTE WRITE, BLOCK ERASE SUPPORT *
; * INPUT / - *
; * OUTPUT / - *
; *****
WCMD .EQU $
;----- RECEIVE ERASE START ADDRESS AND NUMBER OF BLOCKS TO ERASE -----
BSR GET_EADR ;
CMP.B #OK,R0L ;
BNE WCMD_ERR ; ERROR
MOV.L #E_ADR,ER1 ;
MOV.L ER1,@E_ADR_PTR ;
;
NEXTBLOCK
MOV.L @E_ADR_PTR,ER1 ;
MOV.L @ER1+,ER3 ;
MOV.L ER1,@E_ADR_PTR ;
;----- CHECK BLOCKSTART ADDRESS -----
BSR BLK_CHECK ; CHECK ERASE START ADDRESS
CMP.B #OK,R0L ;
BNE WCMD_ERR ; ERASE START ADDRESS ERROR
;----- F-ZTAT ERASURE PROCESS -----
BSR BLK1_ERASE ; ERASURE
CMP.B #OK,R0L ;
BNE WCMD_ERR ; ERASE ERROR
;
MOV.B @ERASEBLOCK,R3L ;
DEC.B R3L ; DECREMENT NUMBER OF BLOCKS TO ERASE
MOV.B R3L,@ERASEBLOCK ;
BNE NEXTBLOCK ; ERASE NEXT BLOCK
;
MOV.B #OK_COM,R2L ;
BSR TX_COM ; TRANSMIT OK COMMAND TO MASTER
;

```

```

;----- GET DATA SIZE AND WRITE START ADDRESS -----
        BSR     GET_WADR           ; GET WRITE START ADDRESS, SIZE
        CMP.B   #OK,R0L           ;
        BNE     WCMD_ERR          ; ERROR
;
WCMD_W10    MOV.B   #REQUEST_COM,R2L ; REQUEST MASTER FOR 128-BYTE DATA
        BSR     TX_COM           ;
;
;----- GET 128-BYTE DATA, TRANSFER TO WRITE AREA -----
        BSR     GET_BUFFER       ;
;
;----- WRITING OF 128-BYTE UNIT -----
        BSR     FWRITE128        ; WRITING TO FLASH MEMORY (128-BYTE UNIT)
        CMP.B   #OK,R0L         ;
        BNE     WCMD_ERR        ; WRITE ERROR
;
        MOV.L   @W_ADR,ER2       ; WRITE AREA ADDRESS
        ADD.L   #128,ER2        ; ADD 128 TO ADDRESS
        MOV.L   ER2,@W_ADR      ;
;
        MOV.L   @RESTSIZE,ER0    ; NUMBER OF BYTES FOR TRANSMIT PROGRAM
        CMP.L   #0,ER0          ; RECEIVED ALL PROGRAMS?
        BNE     WCMD_W10       ; RECEIVE NEXT 128-BYTE DATA AND WRITE
;
        MOV.B   #OK_COM,R2L     ;
        BSR     TX_COM          ; TRANSMIT OK COMMAND TO MASTER
        BSR     RX_COM          ; RECEIVE COMMAND FROM MASTER
        CMP.B   #FWE_OFF_COM,R2L ; RECEIVED COMMAND TO DISABLE FWE TERMINAL FROM
                                ; MASTER?
        BNE     WCMD_ERR        ;
        RTS
;
;----- ERR END -----
WCMD_ERR:
        MOV.B   #'04,R0L        ;
        MOV.B   R0L,@PBDDR      ; PORT B2 OUTPUT
        BCLR   #2,@PBDR         ; PORT B2 "LOW" OUTPUT
        MOV.B   #NG_COM,R2L     ;
        BSR     TX_COM          ; TRANSMIT NG COMMAND TO MASTER
ERROR_5    NOP                  ;
        BRA     ERROR_5         ; INFINITE LOOP
        RTS
;
; *****
; * NAME / ROUTINE TO GET ERASE ADDRESS *
; * FUNCTION / GET WRITE START ADDRESS AND SIZE. RETURN ERROR CODE IN CASE OF ERROR *
; * INPUT / R2L = NUMBER OF ADDRESSES *
; * OUTPUT / E_ADR = EACH ERASE START ADDRESS *
; *****
GET_EADR    .EQU    $
        BTST   #0,@RXPR         ; RXPR0=1?(FINISHED RECEIVING MB0?)
        BEQ   GET_EADR         ;
        MOV.W  @MD0_1,R2       ;
        MOV.B  R2L,@ERASEBLOCK ; SAVE NUMBER OF BLOCKS TO ERASE
        BSET   #0,@RXPR        ; WRITE "1" OR CLEAR "0" IRRO
        CMP.B  #ERASE_COM,R2H  ; RECEIVED ERASE COMMAND FROM ;MASTER?

```

```

        BNE      EADR_ERR      ;
GET_LONG_EADR MOV.L   #E_ADR,ER3 ;
        BSR     RX_4          ;
        DEC.B   R2L           ; INCREMENT NUMBER OF BLOCKS TO ERASE
        BNE     GET_LONG_EADR ;
        MOV.B   #OK,R0L       ; SET OK FLAG
        RTS

;----- ERR -----
EADR_ERR     MOV.B   #NG,R0L   ; SET NG FLAG
        RTS

;
; *****
; * NAME / ERASE BLOCK CHECK ROUTINE *
; * FUNCTION / EVALUATE BLOCK BIT FROM ERASE START ADDRESS, SET RELEVANT REGISTERS *
; * INPUT / ER3 = ERASE START ADDRESS *
; * OUTPUT / R0L = RESULT FLAG (OK=H'00,NG=H'01) *
; *****
BLK_CHECK    .EQU      $
        MOV.B   #0,R4H        ; INITIALIZE BLOCK COUNTER
        MOV.L   #FLMCR1,ER6   ; SET FLMCR ADDRESS
        MOV.L   #EBR1,ER5     ; SET EBR1 ADDRESS
        MOV.L   #((BLOCKADR1-FLASH_MAIN)+USER_RAM),ER0 ; SET EBR1 TABLE
        MOV.B   #MAXBLK1,R4L  ; SET 8 BLOCKS (EBR1)
        CMP.L   #CHK_ADR,ER3  ; ADDRESS H'8000
        BCS     BLK_CHK10     ; BLOCK START ADDRESS LESS THAN ADDRESS H'8000?

;
        MOV.L   #EBR2,ER5     ; SET EBR2 ADDRESS
        MOV.L   #((BLOCKADR2-FLASH_MAIN)+USER_RAM),ER0 ; SET EBR2 TABLE
        MOV.B   #MAXBLK2,R4L  ; SET 4 BLOCKS (EBR2)
BLK_CHK10   MOV.L   @ER0+,ER1  ;
        CMP.L   ER1,ER3       ; MATCH TABLE ADDRESS?
        BEQ     BLK_CHK20     ;

;
        ADD.B   #1,R4H        ; INCREMENT BLOCK COUNTER
        CMP.B   R4L,R4H       ; NUMBER OF BLOCKS MAX?
        BEQ     BLK_ERR       ; ERROR
        BRA     BLK_CHK10     ;

;
BLK_CHK20   MOV.B   R4H,@BLK_NO ; ERASE TARGET BLOCK
        MOV.L   ER1,@EVF_ST   ; SET ERASE START ADDRESS
        MOV.L   @ER0,ER2     ; ERASE END ADDRESS
        MOV.L   ER2,@EVF_ED   ; SET ERASE FINAL ADDRESS
        MOV.B   #OK,R0L       ; NORMAL TERMINATION
        RTS

;
;----- BLK ERR -----
BLK_ERR     MOV.B   #NG,R0L   ; ERASE BLOCK ADDRESS ERROR
        RTS

;
; *****
; * NAME / GET WRITE START ADDRESS AND SIZE ROUTINE *
; * FUNCTION / GETS WRITE START ADDRESS AND SIZE. RETURNS ERROR CODE IN CASE OF ERROR *
; * INPUT / - *
; * OUPUT / W_ADR = WRITE START ADDRESS *
; *          RESTSIZE = WRITE SIZE *
; *****

```

```

; *   ROL           = RESULT FLAG (OK='00,NG='01)                               *
;*****
GET_WADR           .EQU   $
BSR               RX_COM                ; RECEIVE COMMAND FROM MASTER
CMP.B             #WRITE_COM,R2L        ; GOT WRITE COMMAND FROM MASTER?
BNE               ADR_ERR                ;
MOV.L             #W_ADR,ER3            ; GET WRITE START ADDRESS
BSR               RX_4                  ;
SUBS              #4,ER3                 ; W_ADR ADDRESS
MOV.L             @ER3,ER6              ;
MOV.L             ER6,@LAST_JMP         ; ADDRESS OF LAST JUMP (W_ADR START ADDRESS)
AND.B             #ST_ADR,R1L          ; W_ADR LOWER 8BITS H'80 OR H'00?
BNE               ADR_ERR                ; ERROR IF @W_ADR NOT H'XXXXXX80,XXXXXX00
;
;
MOV.L             #RESTSIZE,ER3         ; GET WRITE PROGRAM SIZE
BSR               RX_4                  ;
CMP.L             #0,ER1                ; WRITE BLOCK SIZE ZERO?
BEQ               ADR_ERR                ; ERROR IF @RESTSIZE = H'0000
;
;
MOV.B             #OK,R0L                ; SET OK FLAG
RTS
;
;----- ERR -----
ADR_ERR           MOV.B             #NG,R0L          ; SET NG FLAG
RTS
;
; *****
; * NAME / GET WRITE DATA                                                    *
; * FUNCTION / GETS WRITE DATA FROM HOST                                    *
; * INPUT / RESTSIZE = WRITE SIZE                                            *
; * OUTPUT / W_BUF = WRITE DATA 128 BYTES                                  *
; *****
GET_BUFFER        .EQU   $
MOV.L             @RESTSIZE,ER6         ; SIZE OF WRITE PROGRAM
CMP.L             #128,ER6              ; SIZE OF WRITE PROGRAM LESS THAN 128 BYTES?
BCC               BUFFER10              ; ER6 <= 128
;
;
MOV.L             #W_BUF,ER3            ; WRITE DATA AREA
BUFFER00          MOV.B             #H'FF,R5L        ;
MOV.B             R5L,@ER3              ; FIRST WRITE H'FF
INC.L             #1,ER3                 ;
CMP.L             #W_BUF+128,ER3        ; FINISHED SETTING WHOLE DATA WRITE AREA TO H'FF?
BNE               BUFFER00              ;
;
;
MOV.L             ER6,ER4                ; ER4 = REMAINING DATA SIZE
SUB.L             ER4,ER6                ;
MOV.L             ER6,@RESTSIZE         ; WRITE PROGRAM SIZE = 0
MOV.L             #W_BUF,ER3            ; SET START ADDRESS OF WRITE DATA AREA
BUFFER01          BSR               RX_COM          ; GET ONE BYTE EACH FROM MASTER FOR MB9 TO MB0
MOV.B             R2L,@ER3              ; SAVE RECEIVED DATA TO WRITE DATA AREA
INC.L             #1,ER3                 ; INCREMENT W_BUF ADDRESS
DEC.B             R4L                    ; RECEIVED REST OF DATA?
BNE               BUFFER01              ;
RTS
;
;
BUFFER10          MOV.L             #128,ER4        ;

```

```

SUB.L   ER4,ER6           ; WRITE PROGRAM SIZE-128
MOV.L   ER6,@RETSIZE     ;
MOV.L   #W_BUF,ER3       ; SET START ADDRESS OF DATA WRITE AREA
BSR     RX_128           ;
RTS

;
; *****
; * NAME / GET 1 BYTE ROUTINE                                     *
; * FUNCTION / GET 1 BYTE OF DATA. INFINITE LOOP IN CASE OF ERROR *
; * INPUT / -                                                    *
; * OUTPUT / R2L = RECEIVED DATA                               *
; *****

RX_COM   .EQU    $
BTST    #0,@RXPR         ; RXPR0=1?(FINISHED RECEIVING MB0?)
BEQ     RX_COM
MOV.B   @MD0_1,R2L
BSET    #0,@RXPR         ; WRITE "1" OR CLEAR "0" IRRO
RTS

;
; *****
; * NAME / GET 4 BYTES ROUTINE                                    *
; * FUNCTION / GETS 4 BYTES OF DATA                             *
; * INPUT / -                                                    *
; * OUTPUT / ER2                                                *
; *****

RX_4     .EQU    $
BTST    #0,@RXPR         ; RXPR0=1?(FINISHED RECEIVING MB0?)
BEQ     RX_4
MOV.L   @MD0_1,ER1
MOV.L   ER1,@ER3         ; SAVE START ADDRESS RECEIVED TO @ER3
ADDS    #4,ER3           ; INCREMENT ER3 ADDRESS
BSET    #0,@RXPR         ; WRITE "1" OR CLEAR "0" IRRO
RTS

;
; *****
; * NAME / TRANSMIT 1 BYTE ROUTINE                               *
; * FUNCTION / TRANSMITS 1 BYTE OF DATA FROM ARGUMENT          *
; * INPUT / R2L = DATA TO TRANSMIT                             *
; * OUTPUT / -                                                  *
; *****

TX_COM   .EQU    $
MOV.B   R2L,@MD9_1       ;
BSET    #1,@TXPR_L       ; SET TXPR9 TO "1"
TX_LOOP1 BTST    #1,@TXACK_L ; TXACK9=1?
BEQ     TX_LOOP1         ;
BSET    #1,@TXACK_L     ; WRITE "1" OR CLEAR "0" TXACK9
RTS

;
; *****
; * NAME / GET 128 BYTES ROUTINE                                 *
; * FUNCTION / GET 128 BYTES OF DATA FROM MASTER MB1-8(GETS 64 BYTES TWICE) *
; * INPUT / ER3(#W_BUF)                                         *
; * OUTPUT / @ER3(W_BUF)                                        *
; *****

RX_128   .EQU    $

```

```

MOV.B #2,R0H ; COUNTER=2
RX_128_LOOP1 MOV.B @RXPR,R0L ;
OR.B #H'01,R0L ; RXPR1-7=ALL 1?
CMP.B #H'FF,R0L ;
BNE RX_128_LOOP1 ;
RX_128_LOOP2 BTST #0,@RXPR_L ;
BEQ RX_128_LOOP2 ;
;
MOV.W #64,R4 ;
MOV.L #MD1_1,ER5 ; TRANSFER SOURCE : MD1_1
MOV.L ER3,ER6 ; TRANSFER DESTINATION : W_BUF
EEPMOV.W ; MD1_1-MD8_8 -> W_BUF BLOCK TRANSFER
;
MOV.W @RXPR,E0 ;
OR.W #H'FE01,E0 ;
MOV.W E0,@RXPR ; WRITE "1" OR CLEAR "0" RXPR1-8
MOV.L ER6,ER3 ; SAVE W_BUF+128 TO ER3
DEC.B R0H ; DECREMENT COUNTER
BNE RX_128_LOOP1 ; 128 BYTES RECEIVED? (TWO FETCHES)
RTS
;
; *****
; * NAME / FLASH NOMINAL 128-BYTES WRITE ROUTINE *
; * FUNCTION / WRITES 128 BYTES, VERIFIES *
; * INPUT / @W_ADR = WRITE ADDRESS *
; * @W_BUF = 128-BYTE WRITE DATA *
; * @WT_COUNT = MAX WRITES *
; * OUTPUT / R0L = RESULT FLAG (OK=H'00,NG=H'01) *
; *****
FWRITE128 .EQU $
MOV.W #H'5A5F,R0 ; WDT INITIAL SETTINGS
MOV.W R0,@RSTCSR
;
MOV.W #128,R4
MOV.L #W_BUF,ER5
MOV.L #BUFF,ER6
EEPMOV.W ; W_BUF -> BUFF BLOCK TRANSFER
;
MOV.L #FLMCR1,ER6 ; FLAG CONTROL REGISTER POINTER
;
MOV.W #WLOOP1,R0
BSET.B #SWE,@ER6 ; SET SWE BIT
FWRITE10 DEC.W #1,R0 ; WAIT AFTER SWE SETTINGS (OVER 1µS)
BNE FWRITE10:16
;
XOR.W R0,R0 ; CLEAR WRITE COUNTER
MOV.W R0,@COUNT
;
;===== INITIAL VERIFICATION =====
BSR FWRITEVF ; INITIAL PROGRAM VERIFY
CMP.B #OK,R0L
BEQ FWRITE40 ; INITIAL VERIFY COMPLETED
;
CMP.B #WNG,R0L
BEQ FWRITE30 ; WRITE COMPLETION ERROR

```

```

;
;===== INITIAL WRITE(THERE ARE SUPPLEMENTAL WRITES) =====
FWRTE15      MOV.L   #BUFF,ER2           ; REWRITE DATA
             MOV.L   #TIME30,ER3        ; APPLY P PULSE (30µS)
             BSR     FWRITE              ; PROGRAM
             BSR     FWRITEVF           ; PROGRAM VERIFY
             MOV.B   R0L,@VF_RET
             MOV.L   #OWBUFF,ER2        ; SUPPLEMENTAL WRITE DATA
             MOV.L   #TIME10,ER3        ; APPLY P PULSE (10µS)
             BSR     FWRITE              ; SUPPLEMENTAL PROGRAM
             MOV.B   @VF_RET,R0L
             CMP.B   #OK,R0L
             BEQ     FWRTE40             ; PROGRAM COMPLETE
;
             CMP.B   #WNG,R0L
             BEQ     FWRTE30             ; WRITE COMPLETION ERROR
;
             MOV.W   @COUNT,R0         ; WRITE COUNTER @COUNT+1
             INC.W   #1,R0
             MOV.W   R0,@COUNT
             CMP.W   #OW_COUNT,R0
             BNE     FWRTE15             ; DETERMINE NUMBER OF TIMES(NUMBER OF SUPPLEMENTAL
             ; WRITES)
;
;===== NORMAL WRITE (NO SUPPLEMENTAL WRITE)=====
FWRTE20      MOV.L   #BUFF,ER2           ; REWRITE DATA
             MOV.L   #TIME200,ER3       ; APPLY P PULSE(200µS)
             BSR     FWRITE              ; PROGRAM
             BSR     FWRITEVF           ; PROGRAM VERIFY
             CMP.B   #OK,R0L
             BEQ     FWRTE40             ; PROGRAM COMPLETE
;
             CMP.B   #WNG,R0L
             BEQ     FWRTE30             ; WRITE COMPLETION ERROR
;
             MOV.W   @COUNT,R0         ; WRITE COUNTER @COUNT+ 1
             INC.W   #1,R0
             MOV.W   R0,@COUNT
             MOV.W   @WT_COUNT,E0
             CMP.W   E0,R0
             BNE     FWRTE20             ; DETERMINE NUMBER OF TIMES (MAX NUMBER OF WRITES)
;
;----- ABNORMAL TERMINATION -----
FWRTE30      MOV.W   #WLOOP100,R0
             BCLR.B  #SWE,@ER6           ; CLEAR SWE BIT
FWRTE35      DEC.W   #1,R0                ; WAIT AFTER DISABLING SWE (OVER 100µS)
             BNE     FWRTE35:16
;
             MOV.B   #NG,R0L             ; SET NG
             RTS
;
;----- ABNORMAL TERMINATION -----
FWRTE40      MOV.W   #WLOOP100,R0
             BCLR.B  #SWE,@ER6           ; CLEAR SWE BIT
FWRTE45      DEC.W   #1,R0                ; WAIT AFTER DISABLING SWE (OVER 100µS)
             BNE     FWRTE45:16

```

```

;
;           MOV.B   #OK,R0L           ; SET OK
;           RTS
;
; *****
; * NAME / WRITE VERIFY *
; * FUNCTION / VERIFY TARGET ADDRESS. CREATE REWRITE DATA *
; * INPUT / ER6 = FLMCR REGISTER ADDRESS *
; * @W_ADR = WRITE ADDRESS *
; * @W_BUF = 128-BYTE WRITE DATA *
; * @BUFF = 128-BYTE REWRITE DATA *
; * OUTPUT / @BUFF = 128-BYTE REWRITE DATA *
; * @OWBUFF = 128 BYTES SUPPLEMENTAL WRITE DATA *
; * R0L = RESULT OF VERIFY (OK=H'00,NG=H'01,WNG=H'02) *
; *****
FWRITEVF   .EQU   $
;           MOV.W   #H'FFFF,R5       ; DUMMY WRITE DATA FOR ADDRESS LATCH
;           MOV.L   #BUFF,ER1        ; DATA BUFFER FOR REWRITE
;           MOV.L   #W_BUF,ER2       ; DATA BUFFER FOR WRITE
;           MOV.L   @W_ADR,ER4       ; WRITE ADDRESS FOR FLASH ROM
;===== DATA BUFFER FOR SUPPLEMENTAL WRITE =====
;           MOV.L   #OWBUFF,ER3      ; DATA BUFFER FOR SUPPLEMENTAL WRITE
;=====
;
;           MOV.W   #WLOOP4,E0
;           BSET.B  #PV,@ER6         ; SET PV BIT
FWVF10     DEC.W   #1,E0             ; WAIT AFTER ENABLING PV (OVER 4μS)
;           BNE    FWVF10:16
;
;           MOV.W   #WLOOP2,E0
;           MOV.W   R5,@ER4          ; DUMMY WRITE(LATCH)
FWVF20     DEC.W   #1,E0             ; LATCH WAIT(OVER 2μS)
;           BNE    FWVF20:16
;
;           MOV.W   @ER4+,R0         ; FLASH ROM DATA
;===== CREATE SUPPLEMENTAL WRITE DATA =====
;           MOV.W   @ER1,E0          ; INITIAL WRITE (ONLY USED 6 TIMES)
;           OR.W    R0,E0            ; @COUNT = 0,1,2,3,4,5 VALID DATA
;           MOV.W   E0,@ER3          ; @COUNT = 6-999 INVALID DATA
;           ADDS   #2,ER3
;=====
;           NOT.W   R0
;           MOV.W   @ER2,E0
;           OR.W    R0,E0            ; REVERSE DATA OR WRITE DATA
;           MOV.W   E0,@ER1          ; SET REWRITE DATA
;           ADDS   #2,ER1
;           MOV.W   @ER2+,E0
;           AND.W   E0,R0            ; WHEN READ DATA IS 0 AND WRITE DATA IS 1
;           BNE    FWVF70            ; TO UNABLE TO WRITE ERROR
;
;           CMP.L   #W_BUF+128,ER2
;           BNE    FWVF20            ; VERIFY 128 BYTES
;
;           MOV.W   #WLOOP2,E0
;           BCLR.B  #PV,@ER6         ; CLEAR PV BIT
FWVF40     DEC.W   #1,E0             ; WAIT AFTER DISABLING PV (OVER 2μS)

```



```

FWRT50      DEC.W   #1,E0           ; WAIT AFTER DISABLING P(5μS)
            BNE     FWRT50:16
;
            MOV.W   #WLOOP5,E0
            BCLR.B  #PSU,@ER6      ; CLEAR PSU BIT
FWRT60      DEC.W   #1,E0           ; WAIT AFTER DISABLING PSU(OVER 5μS)
            BNE     FWRT60:16
;
            MOV.W   #H'A500,R0
            MOV.W   R0,@TCSR       ; STOP WATCHDOG TIMER
            RTS
;
; *****
; * NAME / ERASE 1 FLASH BLOCK ROUTINE *
; * FUNCTION / ERASES DESIGNATED BLOCK OF FLASH *
; * INPUT / ER6 = ADDRESS OF FLMCR REGISTER *
; * ER5 = ADDRESS OF EBR REGISTER *
; * @EVF_ST = ERASE START ADDRESS *
; * @EVF_ED = ERASE FINISH ADDRESS *
; * @BLK_NO = BIT NUMBER OF ERASE TARGET BLOCK *
; * @ET_COUNT = MAX NUMBER OF ERASES *
; * OUTPUT / R0L = RESULT FLAG(OK=H'00,NG=H'01) *
; *****
BLK1_ERASE  .EQU    $
            MOV.W   #H'5A5F,R0     ; WDT INITIAL SETTINGS
            MOV.W   R0,@RSTCSR
;
            MOV.W   #WLOOP1,R0
            BSET.B  #SWE,@ER6     ; SET SWE BIT
BLK1_10     DEC.W   #1,R0         ; WAIT AFTER ENABLING SWE(OVER 1μS)
            BNE     BLK1_10:16
;
            XOR.W   R0,R0         ; CLEAR ERASE COUNTER
            MOV.W   R0,@COUNT
;
;===== INITIAL VERIFY =====
            BSR     FERASEVF       ; INITIAL ERASE VERIFY
            CMP.B  #OK,R0L
            BEQ    BLK1_40        ; INITIAL VERIFY COMPLETED
;
;===== ERASE & VERIFY =====
BLK1_20     BSR     FERASE         ; ERASE
            BSR     FERASEVF       ; ERASE VERIFY
            CMP.B  #OK,R0L
            BEQ    BLK1_40        ; ERASE COMPLETED
;
            MOV.W  @COUNT,R0     ; ERASE COUNTER @COUNT+1
            INC.W  #1,R0
            MOV.W  R0,@COUNT
            MOV.W  @ET_COUNT,E0
            CMP.W  E0,R0
            BNE    BLK1_20        ; DETERMINE NUMBER OF ERASES(MAX NUMBER OF ERASES)

```

```

;----- ABNORMAL FINISH-----
BLK1_30      MOV.W    #WLOOP100,R0

              BCLR.B  #SWE,@ER6          ; CLEAR SWE BIT
BLK1_35      DEC.W    #1,R0              ; WAIT AFTER DISABLING SWE(OVER 100μS)
              BNE     BLK1_35:16
;
              MOV.B   #NG,R0L           ; SET NG
              RTS
;
;----- NORMAL FINISH -----
BLK1_40      MOV.W    #WLOOP100,R0
              BCLR.B  #SWE,@ER6          ; CLEAR SWE BIT
BLK1_45      DEC.W    #1,R0              ; WAIT AFTER DISABLING SWE(OVER 100μS)
              BNE     BLK1_45:16
;
              MOV.B   #OK,R0L           ; SET OK
              RTS
;
; *****
; * NAME / ERASE VERIFY ROUTINE *
; * FUNCTION / VERIFIES ERASURE OF SPECIFIED BLOCK *
; * INPUT / ER6 = ADDRESS OF FLPCR REGISTER *
; * @EVF_ST = VERIFY START ADDRESS *
; * @EVF_ED = VERIFY END ADDRESS *
; * OUTPUT / ROL = VERIFY RESULT(OK=H'00,NG=H'01) *
; *****
FERASEVF     .EQU     $
              MOV.L   @EVF_ST,ER1
              MOV.L   @EVF_ED,ER2
              MOV.W   #H'FFFF,E0        ; DUMMY WRITE, ERASE VERIFY DATA
              MOV.W   #WLOOP6,R0
              BSET.B  #EV,@ER6          ; SET EV BIT
VRF10        DEC.W   #1,R0              ; WAIT AFTER ENABLING EV(OVER 6μS)
              BNE     VRF10:16
;
VRF30        MOV.W   #WLOOP2,R0
              MOV.W   E0,@ER1          ; DUMMY WRITE(ADDRESS/LATCH)
VRF40        DEC.W   #1,R0              ; WAIT AFTER LATCH(OVER 2μS)
              BNE     VRF40:16
;
              MOV.W   @ER1+,R0
              CMP.W   E0,R0            ; VERIFY
              BNE     VRF60            ; WHEN TARGET ADDRESS NOT ERASED, END
              CMP.L   ER1,ER2
              BNE     VRF30
;
              MOV.W   #WLOOP4,R0
              BCLR.B  #EV,@ER6          ; CLEAR EV BIT
VRF50        DEC.W   #1,R0              ; WAIT AFTER DISABLING EV(OVER 4μS)
              BNE     VRF50:16
              MOV.B   #OK,R0L           ; SET OK FLAG
              RTS
;
;----- FERASEVF ERR -----
VRF60        MOV.W   #WLOOP4,R0
              BCLR.B  #EV,@ER6          ; CLEAR EV BIT

```

```

VRF70      DEC.W   #1,R0           ; WAIT AFTER DISABLING EV(OVER 4μS)
           BNE    VRF70:16
;
           MOV.B  #NG,R0L         ; SET NG FLAG
           RTS
;
; *****
; * NAME / ERASE ROUTINE *
; * FUNCTION / ERASES SPECIFIED BLOCK *
; * INPUT / ER6 = ADDRESS OF FLMCR REGISTER *
; * ER5 = ADDRESS OF EBR REGISTER *
; * @BLK_NO = BIT NUMBER OF TARGET BLOCK FOR ERASURE *
; * OUTPUT / - *
; *****
FERASE     .EQU    $
           MOV.B  @BLK_NO,R0H
           BSET.B R0H,@ER5         ; SET TARGET BLOCK BIT IN EBR
           MOV.W  #H'A57C,R0       ; 25MHZ (20.8mS )
           MOV.W  R0,@TCSR         ; SET WATCHDOG TIMER
           MOV.W  #WLOOP100,R0
           BSET.B #ESU,@ER6       ; SET ESU BIT
FERS10    DEC.W   #1,R0           ; WAIT AFTER ENABLING ESU(OVER 100μS)
           BNE    FERS10:16
;
           MOV.L  #TIME10000,ERO   ; 10mS
;===== APPLY ERASE PULSE =====
           BSET.B #E,@ER6         ; SET E BIT(ERASE)
FERS20    DEC.L   #1,ERO          ; ERASE TIME : 10mS
           BNE    FERS20:16
;=====
           MOV.W  #WLOOP10,R0
           BCLR.B #E,@ER6         ; CLEAR E BIT
FERS30    DEC.W   #1,R0           ; WAIT AFTER DISABLING E(OVER 10μS)
           BNE    FERS30:16
;
           MOV.W  #WLOOP10,R0
           BCLR.B #ESU,@ER6      ; CLEAR ESU BIT
FERS40    DEC.W   #1,R0           ; WAIT AFTER DISABLING ESU BIT(OVER 10μS)
           BNE    FERS40:16
;
           MOV.W  #H'A500,R0
           MOV.W  R0,@TCSR        ; STOP WATCHDOG TIMER

           MOV.B  @BLK_NO,R0H
           BCLR.B R0H,@ER5       ; CLEAR ERASE TARGET BLOCK IN EBR
           RTS
;
;
; TABLE OF START ADDRESSES FOR ERASE TARGET BLOCKS
           .ALIGN 2
BLOCKADR1 .EQU    $
           .DATA.L H'00000000     ; EB0 4KBYTE
           .DATA.L H'00001000     ; EB1 4KBYTE
           .DATA.L H'00002000     ; EB2 4KBYTE
           .DATA.L H'00003000     ; EB3 4KBYTE

```

```
.DATA.L      H'00004000      ; EB4   4KBYTE
.DATA.L      H'00005000      ; EB5   4KBYTE
.DATA.L      H'00006000      ; EB6   4KBYTE
.DATA.L      H'00007000      ; EB7   4KBYTE
.DATA.L      H'00008000      ; END_ADDRESS1
BLOCKADR2   .EQU          $
.DATA.L      H'00008000      ; EB8   32KBYTE
.DATA.L      H'00010000      ; EB9   64KBYTE
.DATA.L      H'00020000      ; EB10  64KBYTE
.DATA.L      H'00030000      ; EB11  64KBYTE
.DATA.L      H'00040000      ; END_ADDRESS2
.DATA.L      H'00000000      ; DUMMY
.DATA.L      H'00000000      ; DUMMY
.DATA.L      H'00000000      ; DUMMY
.DATA.L      H'00000000      ; DUMMY
ENDFILE     .EQU          $
            .END□
```

Keep safety first in your circuit designs!

1. Renesas Technology Corp. puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.  
Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corp. product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corp. or a third party.
2. Renesas Technology Corp. assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor for the latest product information before purchasing a product listed herein.  
The information described here may contain technical inaccuracies or typographical errors.  
Renesas Technology Corp. assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.  
Please also pay attention to information published by Renesas Technology Corp. by various means, including the Renesas Technology Corp. Semiconductor home page (<http://www.renesas.com>).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corp. is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.  
Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corp. for further details on these materials or the products contained therein.