APPLICATION NOTE

I²C COMMUNICATION BETWEEN ST7 AND M24Cxx EEPROM

by 8-Bit Micro Application

1 INTRODUCTION

The goal of this application note is to present an practical example of communication using the I²C peripheral of the ST7. It shows a basic single master communication between a ST7 microcontroller and an M24Cxx I²C bus EEPROM. The purpose is to implement, from the ST7 through the I²C interface, a write and a read to the external EEPROM without error management.

2 ST7 I²C INTERFACE

The ST7 I²C peripheral allows multi master and slave communication with bus error management. In this application, only the single master mode is used without error management. As the polling mode is the most difficult one to implement, the application is based on this mode, but it can be easily adapted for interrupt management.

The I²C synchronous communication needs only two signals: SCL (Serial clock line) and SDA (Serial data line). The corresponding port pins have to be configured as floating inputs.

Please refer to the ST7 datasheet for more details.

2.1 COMMUNICATION SPEED

The ST7 I²C peripheral allows a large range of communication speeds. It is able to work in standard and fast I²C modes.

In master mode the communication speed is given by the Clock Control Register (CCR). An example is given in Table 1.

Table 1. Example of Possible I²C Communication Speed (f_{CPU}=8 MHz)

	Standard Mode					Fast Mode		
Speed [KHz]	**15.5**	**25.00**	**50.00**	**70.00**	**100.00**	**167.00**	**190.00**	**333.00**
CCR [hex]	EC	9E	4E	37	26	8E	8C	86

2.2 START, STOP CONDITION AND ACKNOWLEDGE GENERATION

In master mode, the Start and Stop conditions can be generated setting the START and STOP bits in the Control Register (CR).

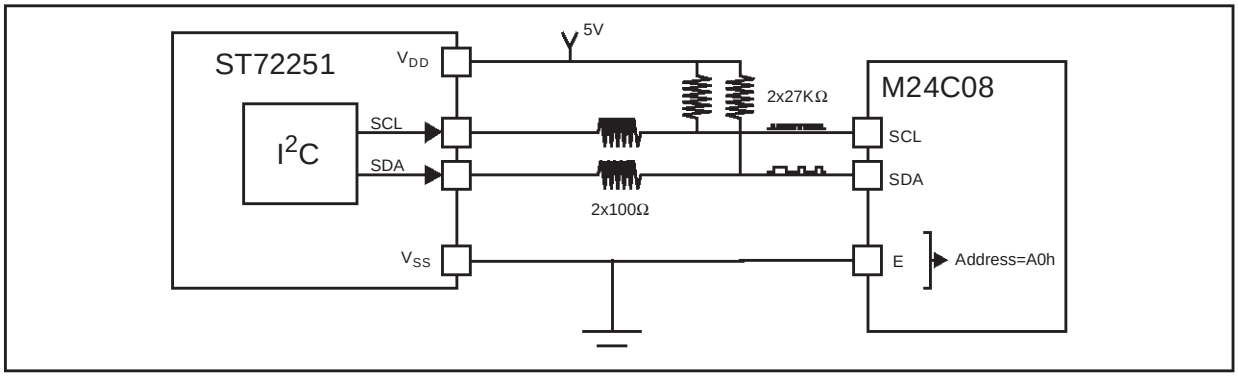
An Acknowledge is sent after an address or a data byte is received when the ACK bit is set in the Control Register (CR).

3 ST7 / M24CXX I²C COMMUNICATION APPLICATION

3.1 HARDWARE CONFIGURATION

The ST7 / M24Cxx I²C communication application hardware consists of a ST72251 microcontroller which communicates with an external M24C08 EEPROM through an I²C bus interface.

Figure 1. ST7 / EEPROM I2C Communications Application



3.2 ST7 I²C PERIPHERAL BASIC DRIVERS

In this chapter all registers refer to the ST7 I²C peripheral (unless otherwise specified).

3.3 INITIALIZE THE I²C PERIPHERAL

In this application the initialization of the ST7 I²C peripheral is done completely by software without taking into account the hardware reset status.

First the Control Register (CR) is cleared and the Data (DR) and Status (SR1,SR2) registers are touched to clear any pending events.

Then, the peripheral is enabled through the Control Register (CR). This action needs to write twice in the register due to the fact that the Control Register (CR) bits can be set only when the PE enable bit is already set. To allow the peripheral to acknowledge the received data the ACK bit of the Control Register (CR) is set.

As the ST24C08 EEPROM is specified with a maximum I²C clock speed at 100-KHz, the ST7 I²C peripheral is set to this speed (CCR=26h) in the application.

3.4 INITIATING A COMMUNICATION ON THE I²C BUS

To initiate an I²C communication, first a start condition has to be generated and then the selected slave address has to be sent, both by the master.

In the ST7 I²C peripheral this action is done by setting the START bit of the Control Register (CR) followed by the write of the slave address in the Data Register (DR) with the least significant bit correctly set (0 = transmission, 1 = reception).

3.5 SENDING A DATA BYTE ON THE I²C BUS

To transmit a new data byte from the ST7 I²C peripheral on the I²C bus, the address or data byte previously transmitted has to be completed correctly. This previous byte transmission check is done with a polling waiting loop for the BTF flag of the Status Register 1 (SR1). If during this delay an error is detected in the Status Registers (SR1,SR2) then the application goes in an infinite loop (no error management).

When the previous data transmission is over, the application writes the new data byte to be transmitted in the Data Register (DR).

Note: If the data byte to transmitted is the first one after the slave address, a dummy write in the Control Register (CR) has to be performed to allow the setting of the BTF bit (see ST7 datasheet for more details). In this application, so this dummy write generates no wrong effect if it is done systematically, the write is done by setting the PE bit for each data byte transmission.

3.6 RECEIVING A DATA BYTE ON THE I²C BUS

To receive a new data byte in the ST7 I²C peripheral from the I²C bus, the data byte to receive has to be completed correctly. This byte reception check is done with a polling waiting loop for the BTF flag of the Status Register 1 (SR1). If during this loop an error is detected in the Status Registers (SR1, SR2) then the application goes in an infinite loop (no error management).

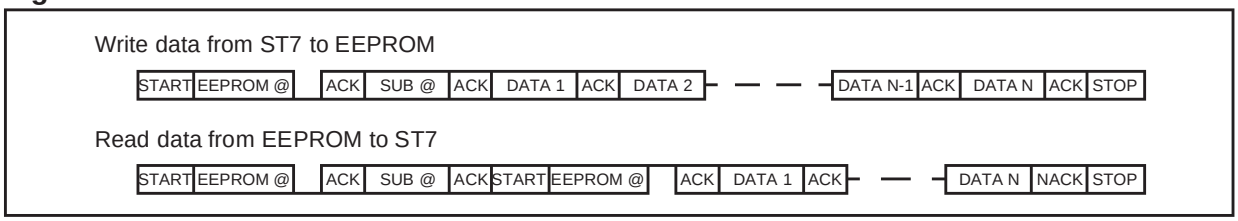
When the data reception is finalized, the application reads the new data byte received in the Data Register (DR).

Note: if the data byte to receive is the first one after the slave address, a dummy write in the Control Register (CR) has to be performed to allow the setting of the BTF bit (see ST7 datasheet for more details). In this application, so this dummy write generates no wrong effect if it is done systematically, the write is done by setting the PE bit for each data byte transmission.

3.7 COMMUNICATE WITH THE I^2C EEPROM

The communication protocol between the ST7 and the external M24Cxx EEPROM is given in Figure 2. For more details, please refer to the ST24C08 datasheet.

Figure 2. I^2C Communication Protocol



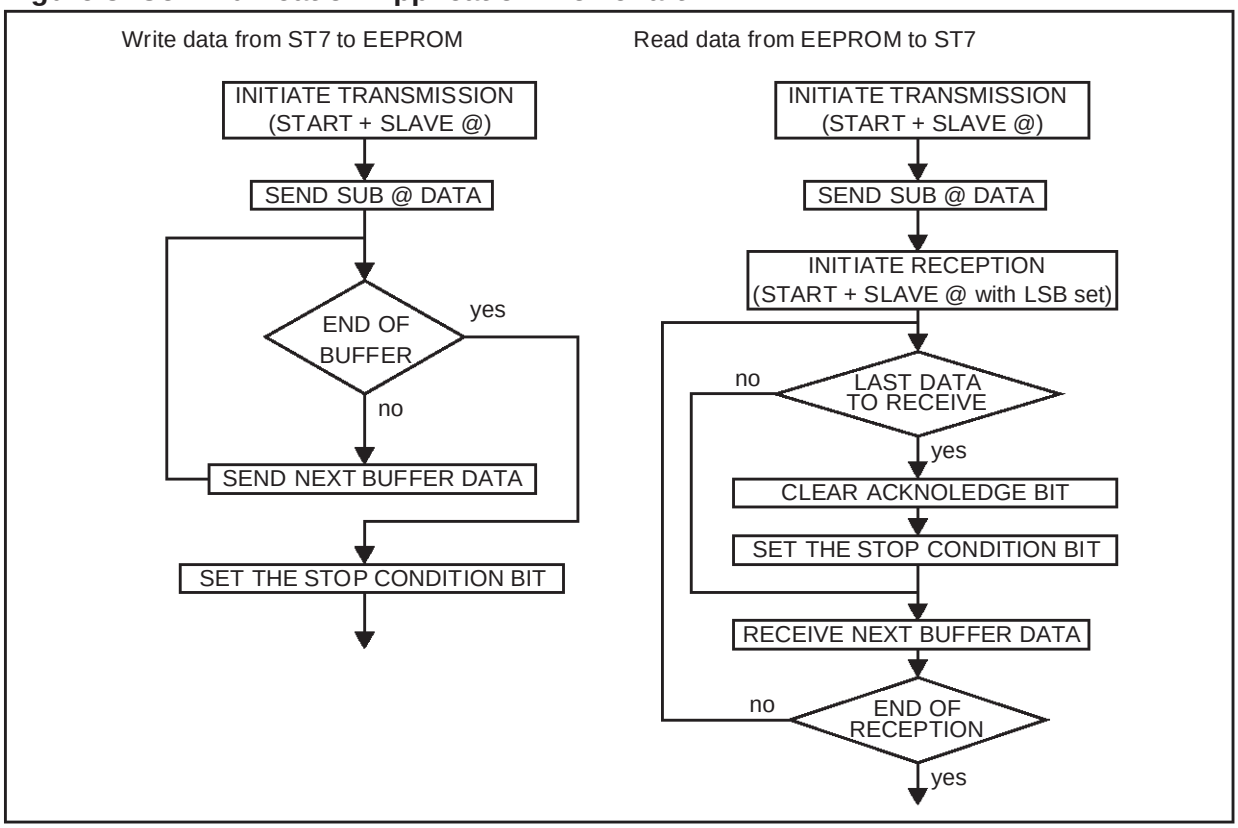
The ST7 / M24Cxx communication application is based on two steps:

– a write of an output buffer content (from the ST7 ROM) in the M24Cxx EEPROM

– a read of this written buffer from the M24Cxx EEPROM to the ST7 RAM.

The Figure 3. shows the flowchart of these two steps.

Figure 3. Communication Application Flowchart



4 SOFTWARE

The assembly code given below is for guidance only. For the missing label declarations please refer to the register label description of the datasheet or the ST web software library (“ST72251.inc” file...).

To adapt this polling software to interrupt management, replace the polling waiting loop by an interrupt event.

```
st7/
;******************** (c) 1997 ST Microelectronics ***************************
; PROJECT:        ST7 I2C COMMUNICATION WITH AN EEPROM 24C08 DEMO SYSTEM
; COMPILER:       ST7 ASSEMBLY CHAIN
; MODULE:         i2c_eepr.asm
; CREATION DATE:  16/03/98
; AUTHOR:         8-Bit Micro Application / STMicroelectronics Rousset
;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
;  THE SOFTWARE INCLUDED IN THIS FILE IS FOR GUIDANCE ONLY. STMicroelectronics
;  SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL
;  DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM USE OF THIS SOFTWARE.
;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
; DESCRIPTION:    ST7 I2C single master peripheral software driver for a
;                 communication between a ST7 and a ST24C08 EEPROM.
;                 Polling software strategy without error management.
;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
; MODIFICATIONS:
; 08/07/97 - V1.0 - C Version.
; 16/03/98 - V3.0 - Assembler version.
;*****************************************************************************

       TITLE  “I2C_EEPR.ASM”
       MOTOROLA
       #INCLUDE “ST72251.inc”    ; ST72251 registers and memory mapping file


;*****************************************************************************
;    Macro definitions
;*****************************************************************************

; I2C CCR possible speeds ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       #define I2C_SPEED$26                                    ; 100.00 KHz.
;      #define I2C_SPEED$37                                     ; 70.00 KHz.
;      #define I2C_SPEED$4E                                     ; 50.00 KHz.
;      #define I2C_SPEED$9E                                     ; 25.00 KHz.
;      #define I2C_SPEED$EC                                     ; 15.75 KHz.
;      #define I2C_SPEED$86                                    ; 333.00 KHz.
;      #define I2C_SPEED$8C                                    ; 190.00 KHz.
;      #define I2C_SPEED$8E                                    ; 167.00 KHz.
```

```
; I2C CR Bit definitions ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       #define PE       5       ; Peripheral enable.
       #define ENGC     4       ; Enable general call.
       #define START    3       ; Start condition generation.
       #define ACK      2       ; Acknowledge level.
       #define STOP     1       ; Stop condition generation.
       #define ITE      0       ; Interrupt enable.
; I2C SR1 Bit definitions ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       #define SR2F     7       ; Status register 2 flag.
       #define ADD10    6       ; 10 bit master addressing mode.
       #define TRA      5       ; Transmitter / receiver.
       #define BUSY     4       ; Bus busy (between start and stop condition.
       #define BTF      3       ; Byte transfer finished.
       #define ADSL     2       ; Addressed as slave.
       #define MSL      1       ; Master / slave.
       #define SB       0       ; Start bit generated (master mode).
; I2C SR2 Bit definitions ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       #define AF       4       ; Acknowledge failure.
       #define STOPF    3       ; Stop detection flag (slave mode).
       #define ARLO     2       ; Arbitration lost.
       #define BERR     1       ; Bus error.
       #define GCAL     0       ; General call (slave mode).

; I2C register initial values ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; Control register: I2C CR                  --- --- PE ENGC START ACK STOP ITE
       #define CR_INIT_VALUE    $24    0;    0   1   0     0    1   0    0


;*****************************************************************************
;    RAM SEGMENT
;*****************************************************************************

       BYTES                    ; following addresses are 8 bit length
       segment byte at 80-FF 'ram0'

.buff_in    DS.B 8              ; Input buffer to read the external EEPROM
.buff_out   DS.B 8              ; Output buffer to write the external EEPROM

;*****************************************************************************
;    ROM SEGMENT
;*****************************************************************************

       WORDS
       segment 'rom'

.buff_data  DC.B 0,1,2,3,4,5,6,7              ; Constant data value buffer.
```

```
;       ***********************************
;       *    SUB-ROUTINES LIBRARY SECTION*
;       ***********************************

; ------------------------------------------------------------------------
; ROUTINE NAME: I2Cm_Init
; INPUT/OUTPUT: None.
; DESCRIPTION: I2C peripheral initialisation routine.
; COMMENTS: Contains inline assembler instructions in C like mode!
; ------------------------------------------------------------------------
.I2Cm_Init
       CLR  I2CCR               ;Force reset status of the control register.
       LD   A,#I2C_SPEED        ;Set the selected I2C-bus speed.
       LD   I2CCCR,A
       TNZ  I2CDR               ;Touch registers to remove pending interrupt.
       TNZ  I2CSR1
       TNZ  I2CSR2
       LD   A,#CR_INIT_VALUE    ;Set initial control register value.
       LD   I2CCR,A
       LD   I2CCR,A             ;Write 2 times: PE=1 then other flag setting.
       RET

; ------------------------------------------------------------------------
; ROUTINE NAME: I2Cm_Start
; INPUT/OUTPUT: None.
; DESCRIPTION: Generates I2C-Bus Start Condition.
; ------------------------------------------------------------------------
.I2Cm_Start
       BSET I2CCR,#START           ; Generate start condition.
.StwaitBTJF I2CSR1,#SB,Stwait      ; Wait for the Start bit generation (EV5).
       RET
; ------------------------------------------------------------------------
; ROUTINE NAME: I2Cm_Stop
; INPUT/OUTPUT: None.
; DESCRIPTION: Generates I2C-Bus Stop Condition.
; ------------------------------------------------------------------------
.I2Cm_Stop
       BSET I2CCR,#STOP            ; Generate stop condition.
       RET
; ------------------------------------------------------------------------
; ROUTINE NAME: I2Cm_SetAddr
; INPUT/OUTPUT: External I2C device address / None.
; DESCRIPTION: Generates Start-bit and transmits the address byte.
; COMMENTS: Transfer sequence = START, EV5, ADD, ACK...
; ------------------------------------------------------------------------
.I2Cm_SetAddr ; IN: A=i2c_addr / OUT: None
       CALL I2Cm_Start          ; Generates a start condition.
       LD   I2CDR,A             ; Write address to be transmitted.
       RET
```

```
; ------------------------------------------------------------------------
; ROUTINE NAME: I2Cm_TxData
; INPUT/OUTPUT: data byte to be transferred / None.
; DESCRIPTION: Transmits a data byte.
; COMMENTS: Transfer sequence = DATA, ACK, EV8...
; ------------------------------------------------------------------------
.I2Cm_TxData; IN: A=i2c_data /OUT: None
       LD   Y,I2CSR2                ; Check the communication error status.
.Txerr JRNE Txerr                   ; Communication error check: infinite loop.
       BSET I2CCR,#PE               ; Touch the control register to pass “EV6”.
       BTJF I2CSR1,#BTF,I2Cm_TxData ; Wait BTF (“EV8”).
.Tx    LD   I2CDR,A                 ; Write data byte to be transmitted.
       RET

; ------------------------------------------------------------------------
; ROUTINE NAME:    I2Cm_RxData
; INPUT/OUTPUT:    Last byte to receive flag (active high) / Received data byte.
; DESCRIPTION:     Receive a data byte.
; COMMENTS:        Transfer sequence = DATA, ACK, EV7...
; ------------------------------------------------------------------------
.I2Cm_RxData ; IN: A=last / OUT: A=Data
       LD   Y,I2CSR2                ; Check the communication error status.
.Rxerr  JRNE Rxerr                  ; Communication error check: infinite loop.
       BSET I2CCR,#PE               ;Touch the control register to pass “EV6”.
       BTJF I2CSR1,#BTF,I2Cm_RxData ; Wait BTF (“EV7”).
       TNZ  A                       ; Check if it is the last byte to receive.
       JREQ Rx
       CALL I2Cm_Stop               ; End of communication: stop condition generation.
.Rx    LD   A,I2CDR                 ; Read data byte received.
       RET

; ------------------------------------------------------------------------
; ROUTINE NAME: I2Cm_Tx
; INPUT/OUTPUT: I2c dest @, sub @, Nb data byte to transmit / None.
; DESCRIPTION: Transmit output data buffer via I2C.
; COMMENTS: Low significant bytes first.
; ------------------------------------------------------------------------
.I2Cm_Tx    ; IN: Y=sub_add, X=nb, A=dest_add / OUT: None.
       CALL I2Cm_SetAddr            ; Slave address selection on I2C bus.
       LD   A, Y
       CALL I2Cm_TxData             ; Send sub-address through I2C bus.
.Txcont DEC  X                      ; X reg contains the number of data byte to transmit.
       JRMI Txend                   ; End of output data buffer reached.
       LD   A, (buff_out,X)         ; Next output buffer data byte selected.
       CALL I2Cm_TxData             ; Send data byte through I2C bus.
       JRA  Txcont
.Txend JRA  I2Cm_Stop               ; Communication End: stop generation and return.
```

```
; ------------------------------------------------------------------------
; ROUTINE NAME: I2Cm_Rx
; INPUT/OUTPUT: Buffer @, sub @, Nb data byte to receive, I2c dest @ / None.
; DESCRIPTION: Receive in data buffer via I2C.
; COMMENTS:    Low significant bytes first.
; ------------------------------------------------------------------------
.I2Cm_Rx    ; IN: Y=sub_add, X=nb, A=dest_add / OUT: None.
       PUSH A                       ; Store the dest_add in stack.
       CALL I2Cm_SetAddr            ; Slave address selection on I2C bus.
       LD   A, Y
       CALL I2Cm_TxData             ; Send sub-address through I2C bus.
       POP  A                       ; Restore the dest_add in stack.
       OR   A,#$01                  ; Force the LSB device address to be 1 (read mode).
       CALL I2Cm_SetAddr            ; Slave address selection on I2C bus.
.RxcontCLR  A                       ; Not yet the end of the communication.
       DEC  X                       ; X reg contains the number of data byte to receive.
       JRMI Rxend                   ; End of input data buffer reached.
       JRNE Rxb
       BRES I2CCR,#ACK              ; Non acknowledge after last reception.
       LD   A,#$01                  ; End communication request.
.Rxb   CALL I2Cm_RxData             ; Receive data byte from I2C bus.
       LD   (buff_in,X),A           ; Next input buffer data byte stored.
       JRA  Rxcont
.Rxend BSET I2CCR,#ACK             ; Acknowledge after reception and return.
       RET


;      ****************************
;      *    MAIN-ROUTINES SECTION *
;      ****************************

.main
       ; Input buffer initialization ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       LD   X,#7
.InibufCLR  (buff_in,X)
       DEC  X
       JRPL Inibuf

       ; Copy Constant data buffer in buff_out ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       LD   X,#7
.CpybufLD   A,(buff_data,X)
       LD   (buff_out,X),A
       DEC  X
       JRPL Cpybuf

       ; Init I2C peripheral ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       CALL I2Cm_Init
```

```
       ; Write data from buff_out to the EEPROM ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       LD   A,#$A0                  ; EEPROM address parameter setting.
       LD   X,#8                    ; Number of byte to write in the EEPROM.
       LD   Y,#$50                  ; EEPROM internal data address.
       CALL I2Cm_Tx                 ; IN: Y=sub_add, X=nb, A=dest_add / OUT: None.
       ; Waiting loop ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       CLR  A
.Wtloop JREQ Wtloop                 ; To exit from this loop: break point and set A<>0:
                                    ;first click on the Z flag; then change A value and
                                    ; press enter.
       ; Read data from the EEPROM to the buff_in~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       LD   A,#$A0                  ; EEPROM address parameter setting.
       LD   X,#8                    ; Number of byte to write in the EEPROM.
       LD   Y,#$50                  ; EEPROM internal data address.
       CALL I2Cm_Rx                 ; IN: Y=sub_add, X=nb, A=dest_add / OUT: None.
       ; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
.end   JRA  end                     ; Infinite main loop.
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~



;      ******************************************
;      *    INTERRUPT SUB-ROUTINES LIBRARY SECTION*
;      ******************************************

.dummy_rt    iret                   ; Empty subroutine. Go back to main (iret instruction)
.i2c_rt      iret                   ; I2C Interrupt

       segment 'vectit'
          DC.W  dummy_rt   ;FFE0-FFE1h location
          DC.W  dummy_rt   ;FFE2-FFE3h location
i2c_it:   DC.W  dummy_rt   ;FFE4-FFE5h location
          DC.W  i2c_rt     ;FFE6-FFE7h location
          DC.W  dummy_rt   ;FFE8-FFE9h location
          DC.W  dummy_rt   ;FFEA-FFEBh location
          DC.W  dummy_rt   ;FFEC-FFEDh location
timb_it:  DC.W  dummy_rt   ;FFEE-FFEFh location
          DC.W  dummy_rt   ;FFF0-FFF1h location
tima_it   DC.W  dummy_rt   ;FFF2-FFF3h location
spi_it:   DC.W  dummy_rt   ;FFF4-FFF5h location
          DC.W  dummy_rt   ;FFF6-FFF7h location
io_bc_it: DC.W  dummy_rt   ;FFF8-FFF9h location
io_a_it:  DC.W  dummy_rt   ;FFFA-FFFBh location
softit:   DC.W  dummy_rt   ;FFFC-FFFDh location
reset:    DC.W  main       ;FFFE-FFFFh location

       END
;********* (c) 1997 STMicroelectronics ******************** END OF FILE ****
```

THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNEXION WITH THEIR PRODUCTS.