APPLICATION NOTE

ST7 ROUTINE FOR I2C SLAVE MODE MANAGEMENT

by Microcontroller Division Application Team

1 INTRODUCTION

The goal of this application note is to present a useful example of communication using the I2C peripheral of the ST7. The ST7 microcontroller is used as a slave and can communicate with any master. This slave, through the I2C interface, receives words from the master implementing error management and returns them. This application has been realized with a ST72E251 and a 7-bit addressing mode.

2 ST7 I2C INTERFACE

The ST7 I2C peripheral allows multi master and slave communication with bus error management. In this application, only the single slave mode is used (with error 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 (here PA4 and PA6).

Please refer to the ST7 datasheet for more details.

2.1 COMMUNICATION SPEED

The ST7 I2C peripheral allows to communicate at different speeds. It is able to work in standard and fast I2C modes up to 400kHz.

It's the master which imposes the communication speed. For more details, please refer to the AN971.

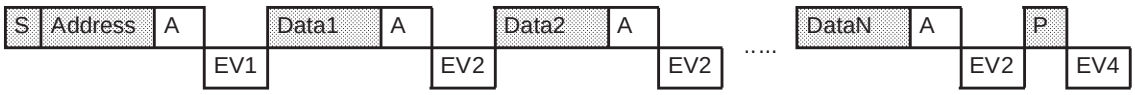
2.2 START, STOP CONDITION AND ACKNOWLEDGE GENERATION

The Start and Stop conditions are sent by the master.

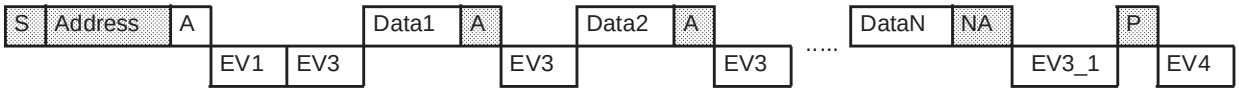
An Acknowledge is sent after an address or a data byte is received when the ACK bit is set in the Control register (CR).

2.3 TRANSFER SEQUENCING

Slave receiver:



Slave transmitter:



Legend:S=Start, P=Stop, A=Acknowledge, NA=Non-Acknowledge, EVx=Event (with interrupt if ITE=1).

EV1: EVF=1, ADSL=1, cleraed by reading SR1 register.

EV2: EVF=1, BTF=1, cleraed by reading SR1 register followed by reading DR register.

EV3: EVF=1, BTF=1, cleraed by reading SR1 register followed by writing DR register.

EV3_1: EVF=1, AF=1, cleraed by reading SR1 register.

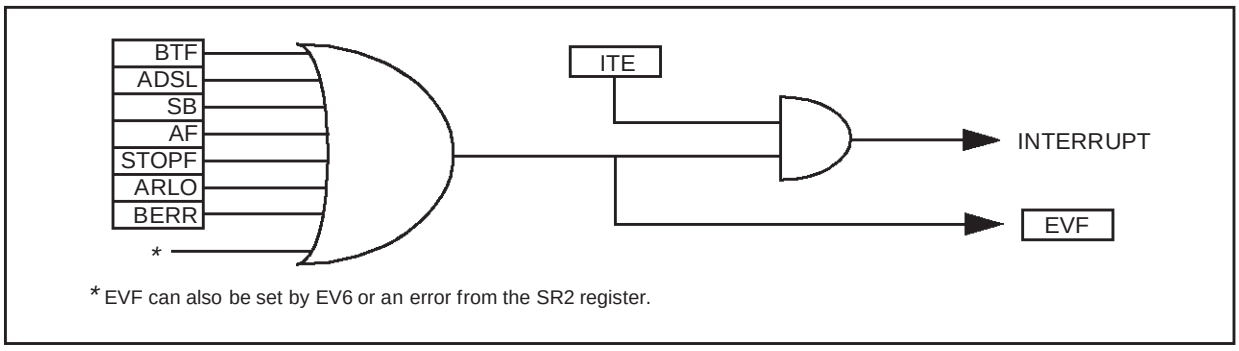
EV4: EVF=1, stopf=1, cleraed by reading SR2 register.

Grey cells are events sent by the master whereas blank ones are events sent by the slave.

These frames are sequential and as the SDA line is bidirectional, this line is held sometimes by the master and sometimes by the slave.

2.4 INTERRUPTS

Figure 1. Event Flags and Interrupt Generation

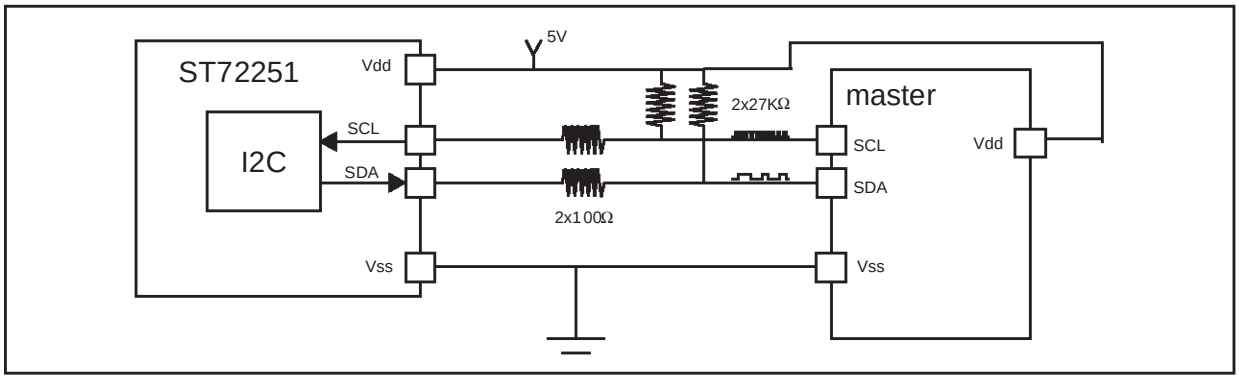


3 ST7 I2C COMMUNICATION APPLICATION

3.1 HARDWARE CONFIGURATION

The ST7 I2C communication application hardware is composed by a ST72251 microcontroller used as a slave which communicates with any master through an I2C bus interface.

Figure 2. ST7 I2C Communication Application



3.2 ST7 I2C PERIPHERAL BASIC DRIVERS

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

3.2.1 Initialize the I2C peripheral

In this application the initialization of the ST7 I2C peripheral is done completely by software.

First the Control register (CR) is cleared and the Data (DR) and Status (SR1,SR2) registers are touched to clear all possible pending event.

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.

3.2.2 Slave Communication on the I2C Bus

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

When the address received matches, the interface generates in sequence:

- Acknowledge pulse if the ACK bit is set.

- EVF and ADSL are set with an interrupt if ITE bit is set.

Then the interface waits for a read of SR1 register, holdinig the SCL line low. Next, read the DR register to determine from the least significant bit of I2COAR1 (ADD0) if the slave has to be reveiver or transmitter.

3.2.3 Receiving Data on the I2C Bus

The slave receives then bytes from the SDA line into the DR register via the internal shift interface. After each byte, the interface generates in sequence:

- Acknowledge pulse if the ACK bit is set.

- EVF and BTF bits are set with an interrupt if ITE bit is set.

Then the interface waits for a read of the SR1 register followed by a read of the DR register (see Transfer Sequencing).

After each reception, the word received is placed in a table. If an error occurs, the interface is reinitialized (I2CCR cleared and put to its initial value after having removed pending interrupts) and the slave waits next transmission.

3.2.4 Sending Data on the I2C Bus

When the slave receives all data from the master (and the stop condition), he becomes transmitter and then sends the master back data he stored into a table. He sends data from DR register to the SDA line via the internal shift register.

The slave waits for a read of the SR1 register followed by a read of the DR register(see Transfer Sequencing).

When the acknowledge pulse is received:

- the EVF and BTF bits are set by hardware with an interrupt if ITE bit is set.

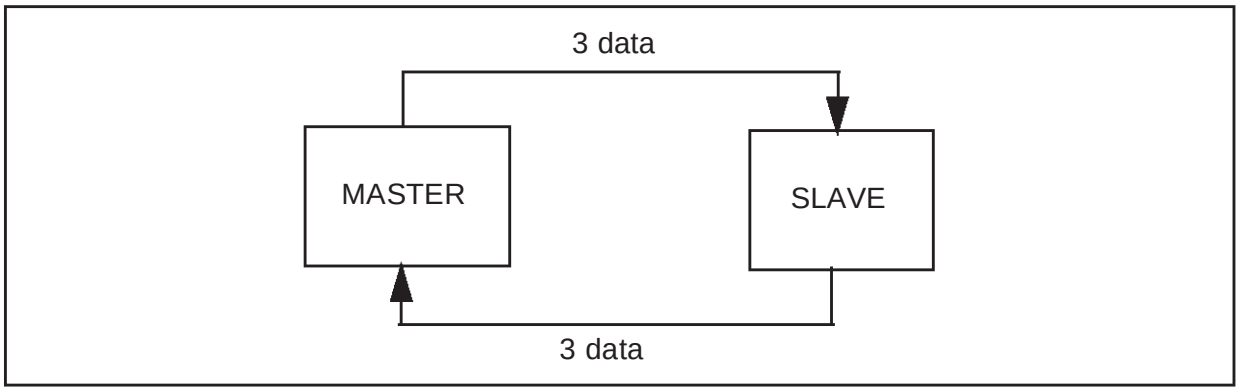
After the last data byte is transferred, a Stop condition is generated by the master. The interface detects this condition and sets:

- EVF and STOPF bits with an interrupt if ITE bit is set.

Then the interface waits for a read of the SR2 register.

In this application, the least significant bit of I2COAR1 (ADD0) is used as a flag to determine the mode (receiver or transmitter) of the microcontroller.

3.3 COMMUNICATION

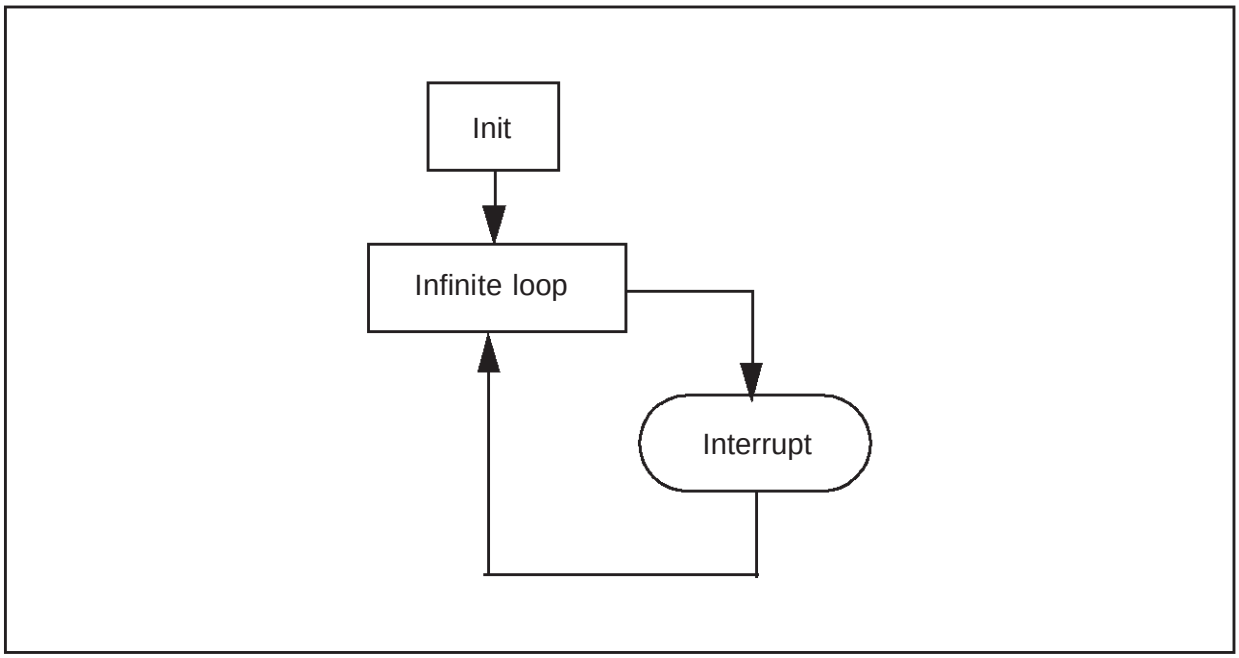


The ST7 slave mode management application is based on two steps driven by the master:

– a reception of data from any master.

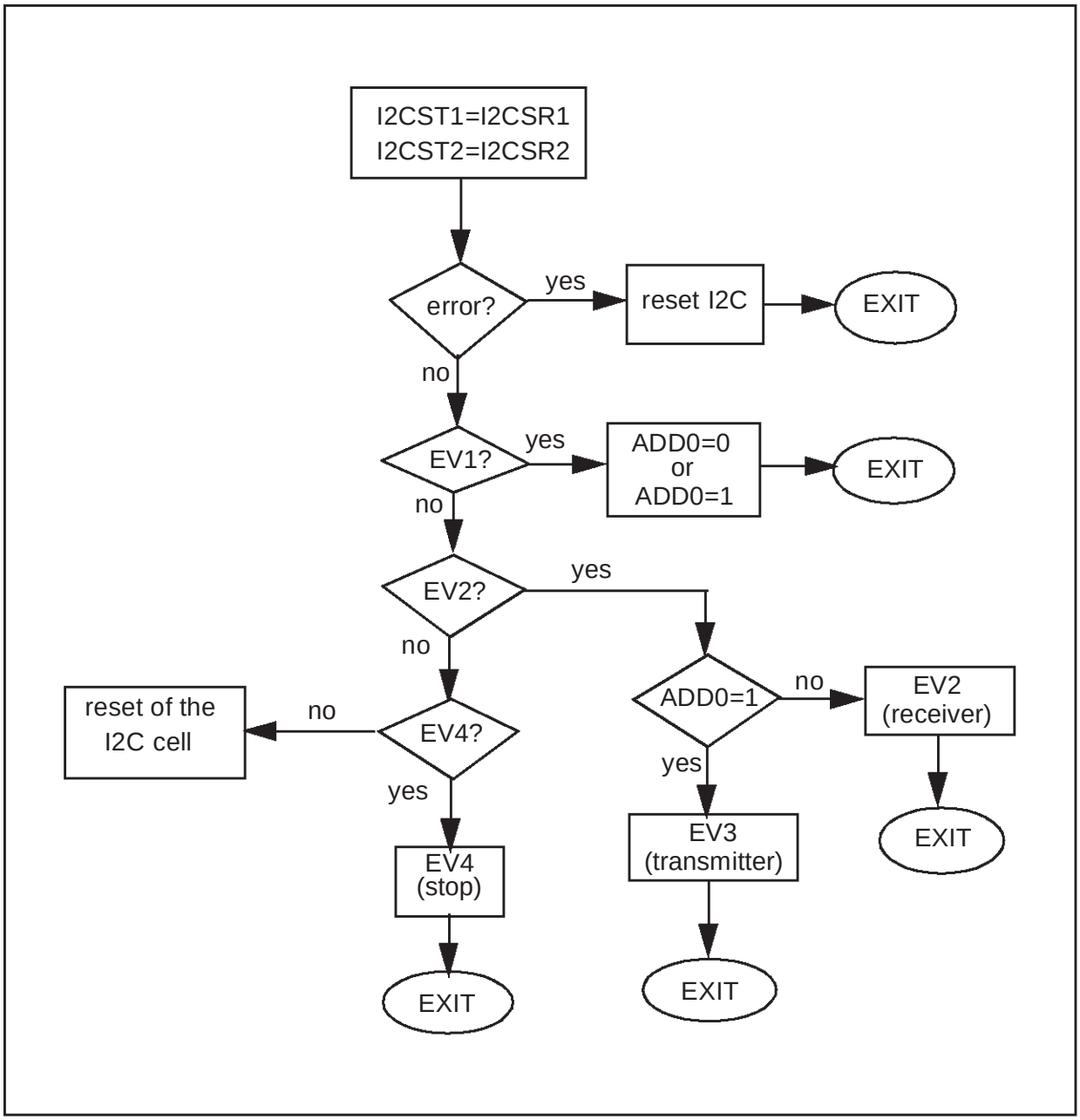
– a transmission of received data to the master.

Figure 3. Main Program Flowchart



Events in the following flowchart refer to events defined into the paragraph “Transfer sequencing” on page 2.

Figure 4. Interrupt Flowchart



4 SOFTWARE

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

```
st7/                    ; the first line is reserved
                        ; for specifying the instruction set
                        ; of the target processor

;*******************************************************************
; TITLE:                 MODULE1.ASM
; AUTHOR:                PPG Microcontroller Applications Team
; DESCRIPTION:           Short Demonstration Program
;
;
;*******************************************************************

    TITLE  “MODULE1.ASM”
                            ; this title will appear on each
                            ; page of the listing file
    MOTOROLA                ; this directive forces the Motorola
                            ; format for the assembly (default)
    #INCLUDE “st72251.inc”  ; include st72251 registers and memory mapping
file
    #INCLUDE “constant.inc” ; include general constants file

;*******************************************************************
;    Variables, constants defined and referenced locally
;    You can define your own values for a local reference here
;*******************************************************************

;* I2C_CR Bit definitions ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#define PE   5                  ; Peripheral enable.
#define ENGC  4                 ; Enable general call.
#define START  3            ; Start condition generation.
#define ACK   2                 ; Acknoledge 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           ; 10bit master addressing mode.
#define TRA   5                 ; Transmitter / receiver.
```

```
#define BUSY  4                    ; Bus busy (between start and stop condi-
tion.
#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_0AR1 Bit definitions ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 #define ADD0       0

;* I2C register initial values ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;* Control register: I2C_CR      --- --- PE ENGC START ACK STOP ITE
#define CR_INIT_VALUE   $25    ; 0  0  1  0  0  1  0  1

;*****************************************************************************
;    Public routines (defined here)
;*****************************************************************************
    WORDS

  segment 'rom'

;*****************************************************************************
;ROUTINE NAME : I2Cm_Init
;INPUT/OUTPUT : None.
;DESCRIPTION : I2C peripheral initialisation routine.
;*****************************************************************************
.I2Cs_Init

  LD  A,#$30
  LD  I2COAR1,A                ; I2C Bus address of the interface
  LD  A,#$40
  LD  I2COAR2,A                ; fcpu=8MHz.

  TNZ  I2CDR                   ; Touch registers to remove pending interrupt.
  CLR  I2CCR                   ; Force reset status of the control register.
  LD  a,#CR_INIT_VALUE ; Set initial control register value.
  LD  I2CCR,A                  ; a first time to set PE.
  LD  I2CCR,A                  ; a second time to load the CR value.
```

```
  RET

;*************************************************************
;    Program code
;*************************************************************
.main
    BRES PADDR,#4
     BRES PAOR,#4
     BRES PADDR,#6
     BRES PAOR,#6    ;PA4 (SCL) and PA6 (SDA) configured as floating inputs.

     CLR index

.init  CALL I2Cs_Init          ;Initializations.

     RIM                       ;Enable interrupts.
loop   JRA   loop              ;Infinite loop.

; ***************************************************************
; This set of instructions uses simple assembly mnemoniques.
; We can notice that the loop label is defined only locally (no dot
; in front of it) so it can not be seen by others modules linked
; with this file.
; ***************************************************************

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

.dummy  iret
.sw_rt  iret                   ; Empty subroutine. Go back to main (iret in-
struction)
.ext0_rt iret
.ext1_rt iret
.spi_rt iret
.tima_rt iret
.timb_rt iret
.i2c_rt

     LD   A,I2CSR1
     LD   I2CST1,A             ;Store I2CSR1 into I2CST1.
     LD   A,I2CSR2
```

```
    LD   I2CST2,A     ;Store I2CSR2 into I2CST2.
    BCP   A,#%00010010    ; Acknowledge Failure or Bus Error?
    JREQ  test_EV1     ; if not -> test_EV1.

      BCP          A,#%00000010
      JRNE         rst         ; if BERR, jump to rst
      LD           A,index     ; if AF and index=last byte to transmit -> EV3_1.
      CP           A,#3
      JRPL         EV3_1
    ; if error, reinitialization:

rst   CLR   I2CCR      ; Force reset status of the control register.
    TNZ   I2CDR       ; Touch registers to remove pending interrupt.
    BRES  I2CCR,#PE
    BSET  I2CCR,#PE            ;reset I2C cell
    LD   a,#CR_INIT_VALUE      ; Set initial control register value.
    LD   I2CCR,A
    IRET

test_EV1
    BTJF  I2CST1,#ADSL,test_EV2     ;own address is recognized
    BTJT  I2CDR,#ADD0,exit
      BSET I2COAR1,#ADD0            ;set ADD0 used as a flag to indicate
                                    ; the slave transmitter mode.
      IRET
EXIT BRESI2COAR1,#ADD0              ;slave receiver mode : ADD0=0.
   IRET

test_EV2    ;or EV3
    BTJF  I2CST1,#BTF,test_EV4 ;if BTF=0 -> test if a Stop is received.
    BTJT  I2COAR1,#ADD0,EV3    ;if the slave is in transmitter mode -> EV3.
EV2   LD   A,I2CDR             ;if not, the slave is the receiver.
      LD   X,index
      LD   (tab,X),A           ;store the received value into tab.
      INC   index

end_EV2 IRET

EV3 LD   X,index
    LD   A,(DATA,X)            ;the slave transmitter mode
    LD   I2CDR,A               ;transmission of values the slave received.
   INC index
end_ev3 IRET

EV3_1 LDA,#$FF                 ; Non acknowledge (AF=1).
```

```
    LD I2CDR,A                  ; The last byte has been transmitted.
    IRET

test_EV4
     BTJF  I2CST2,#STOPF,rst    ;end of reception or transmission.

 EV4 CLR   index
    BTJF I2COAR1,#ADD@,end_it   ;clear the table if ADD0=1 (end of transmission).
.end_it IRET

     segment 'vectit'
; *****************************************************************
; This last segment should always be there in your own programs.
; It defines the interrupt vector adresses and the interrupt routines' labels
; considering the microcontroller you are using.
; Refer to the MCU's datasheet to see the number of interrupt vector
; used and their addresses.
; Remind that this example is made for a ST7225 based application.
; *****************************************************************
                    DC.W  dummy      ;FFE0-FFE1h location
                    DC.W  dummy      ;FFE2-FFE3h location
.i2c_it             DC.W  i2c_rt     ;FFE4-FFE5h location
                    DC.W  dummy      ;FFE6-FFE7h location
                    DC.W  dummy      ;FFE8-FFE9h location
                    DC.W  dummy      ;FFEA-FFEBh location
                    DC.W  dummy      ;FFEC-FFEDh location
.timb_it            DC.W  timb_rt    ;FFEE-FFEFh location
                    DC.W  dummy      ;FFF0-FFF1h location
.tima_it            DC.W  tima_rt    ;FFF2-FFF3h location
.spi_it             DC.W  spi_rt     ;FFF4-FFF5h location
                    DC.W  dummy      ;FFF6-FFF7h location
.ext1_it            DC.W  ext1_rt    ;FFF8-FFF9h location
.ext0_it            DC.W  ext0_rt    ;FFFA-FFFBh location
.softit             DC.W  sw_rt      ;FFFC-FFFDh location
.reset              DC.W  main       ;FFFE-FFFFh location


       ; This last line refers to the first line.
       ; It used by the compiler/linker to determine code zone
    END
```

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.