APPLICATION NOTE

SPI COMMUNICATION BETWEEN ST7 AND EEPROM

by 8-Bit Micro Application Team

INTRODUCTION

The goal of this application note is to present a practical example of communication using the SPI peripheral of the ST7.

It shows an easy way of communicating between a ST7 microcontroller and a M95xxx SPI EEPROM. The purpose is to perform, through SPI, a write in the memory, followed by a read of the written data.

1 ST7 / EEPROM SPI INTERFACE

This section summarizes the main features of the ST7/EEPROM SPI interface. Please refer to the ST7 datasheet for more details.

The Serial Peripheral Interface (SPI) allows full-duplex, synchronous, serial communication between devices. A SPI system may consist of a master and several slaves, or of a system in which devices may be either master or slave.

There is only one master at any one time.

The Bus signals are:

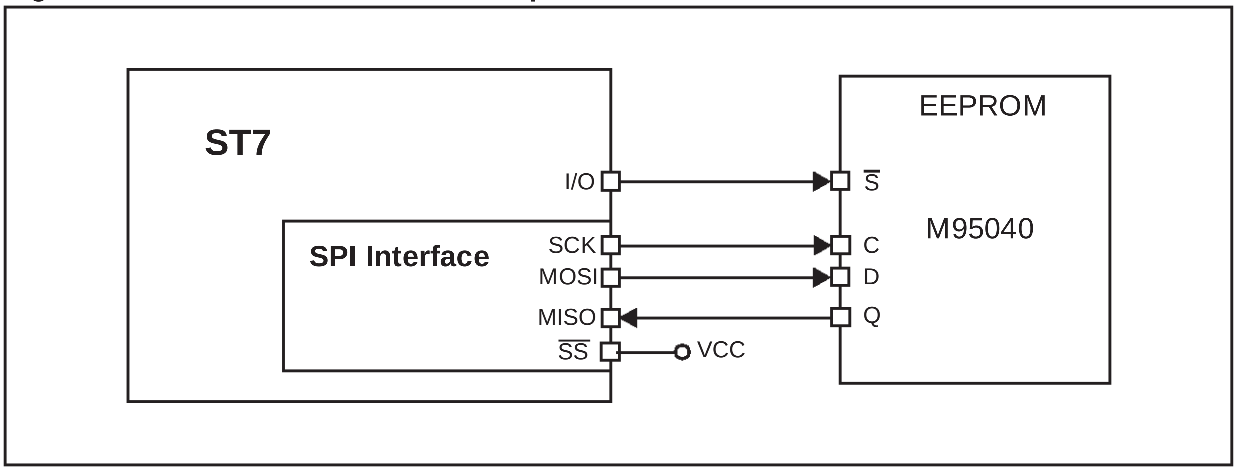
- The serial clock (SCK).
- The MOSI (Master Out Slave In).
- The MISO (Master In Slave Out).

One more pin, the $\overline{SS}$ pin (slave select), is needed for connection between devices to select the slave or the master mode for each device.

In this application the ST7 is always used as master ($\overline{SS}$ pin = high level) and configures the EEPROM mode through an output. The ST7 and SPI interface set-up is shown Figure 1.

During SPI transfer, data is simultaneously transmitted (shifted out serially) and received (shifted in serially). Data are transmitted MSB first. The serial clock is used to synchronize the data transfer during a sequence of eight clock pulses.

Figure 1. ST7 and SPI Interface Set-Up



2 ST72251 CONFIGURATION

2.1 I/O CONTROL

Five pins of the ST72251 are used:

- The 3 data and clock SPI pins (SCK, MOSI, MISO).
- $\overline{SS}$ pin to select master or slave mode.
- An output pin to select and deselect the M95xxx.

In our application, the output for selecting the M95xxx is pin 3 of Port B. It is configured as output push-pull (refer to the datasheet for details).

2.2 SPI PERIPHERAL

2.2.1 General

This peripheral is configured with the SPI Control Register.

7							0
SPIE	SPE	-	MSTR	CPOL	CPHA	SPR1	SPR0

The output must be enabled (SPE = 1).

If $\overline{SS}$ pin is high, ST72251 can be declared as master, setting the MSTR bit.

The transmission speed, in master mode, is selected using SPR0 and SPR1 bits.

CPOL and CPHA bits define timing characteristics.

When the SPIE bit is set, SPI interrupts are enabled (not used in our case).

2.2.2 Baud rates

First the communication speed must be chosen. As the clock is given by the master, the user has to configure the baud rate of the ST7 SCI (SPR1-SPR0 bits of the SPCR register).

For each CPU frequency, four Baud Rates are available. If f_{CPU} = 8MHz, the baud rate range is 62.5kHz to 1MHz.

2.2.3 Clock phase and polarity

Then the clock polarity and the clock phase have to be chosen.

The clock polarity (CPOL bit of the SPCR register) controls the steady state value of the clock when no data is being transferred.

The clock phase (CPHA bit of the SPCR register) selects on which clock transition the bit capture is made and consequently on which clock transition data is latched.

The user must be careful of the fact that some devices do not allow all timing relationships. For instance, the ST95040 EEPROM device accepts only CPOL,CPHA = (0,0) or (1,1) configurations.

3 M95040 EEPROM MANAGEMENT AND CONFIGURATION

3.1 MAIN FEATURES

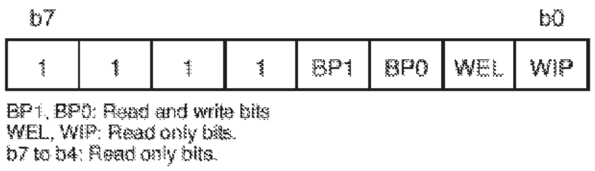
This is a 4K memory composed of two pages of 2K bytes.

All instructions, addresses and data are shifted in and out of the chip MSB first.

The write protect pin ($\overline{W}$) and the hold pin ($\overline{H}$) are not used in our application (both are high level).

3.2 STATUS REGISTER

This device has one status register.



b7							b0
1	1	1	1	BP1	BP0	WEL	WIP

BP1, BP0: Read and write bits
WEL, WIP: Read only bits.
b7 to b4: Read only bits.

The BP1 and BP0 bits of the status register can be used to write protect a block of memory. In this application, both bits are cleared, allowing writing in all the memory.

The WEL bit indicates the status of the write enable latch.

The WIP bit indicates whether the memory is busy with a write operation.

3.3 INSTRUCTION SET

Prior to any operation, the device must be selected ($\overline{SS}$ pin at low level), then an one-byte instruction code must be sent to the EEPROM. The device has a set of 6 instructions (see Table 1).

Table 1. Instruction Set

Instruction	Description	Instruction format
WREN	Set Write Enable Latch	0000 0110
WRDI	Reset Write Enable Latch	0000 0100
RDSR	Read Status Register	0000 0101
WRSR	Write Status Register	0000 0001
READ	Read Data from Memory Array	0000 $A_8$011
WRITE	Write Data to Memory Array	0000 $A_8$010

Notes: A_8 = 1, Upper page selected.
A_8 = 0, Lower page selected.

4 ST7 / EEPROM SPI PROTOCOL

The use of the main functions are briefly described below. For a complete description of the protocol, please refer to the M95040 SPI EEPROM data sheet.

Write_enable: The memory contains a write enable latch. This latch must be set prior to any WRITE or WRSR operation.

Byte_write: This function will write up to 16 bytes of data in the EEPROM. After the WRITE instruction, the memory address must be specified before sending data.

Byte_read: This function reads the memory. After the READ instruction, the memory address is specified. To be able to send the data, the EEPROM must receive the clock from the master: the ST7. This is made by sending a dummy byte. This operation will generate the 8 clock bits needed. The dummy value will not be seen by the EEPROM.

Write_SR: Give the proper value to the Status Register prior to any communication.

Read_SR: This process is similar to the byte read process. It will be used to check if there is a write in progress (WIP bit of the status register).

5 SPI COMMUNICATION BETWEEN AN ST7 AND AN EEPROM

The software included with this application note is only the SPI driver. The complete software can be found in the software library in the ST internet website. It is of course only an example. It is up to the user to adapt it to his specific application.

5.1 GENERAL DESCRIPTION

The software is a polling SPI communication between ST72251 and EEPROM.

The first part of the software performs to the initialization of the ST72251 (core and SPI peripheral) and of the EEPROM (value of status register for no protection).

Then it executes the write cycles. «write_loop» writes in the EEPROM, the values 0 to 127 at addresses 255 down to 128.

The third part executes the read cycles. «read_loop» reads the content of the EEPROM at addresses 255 down to 128.

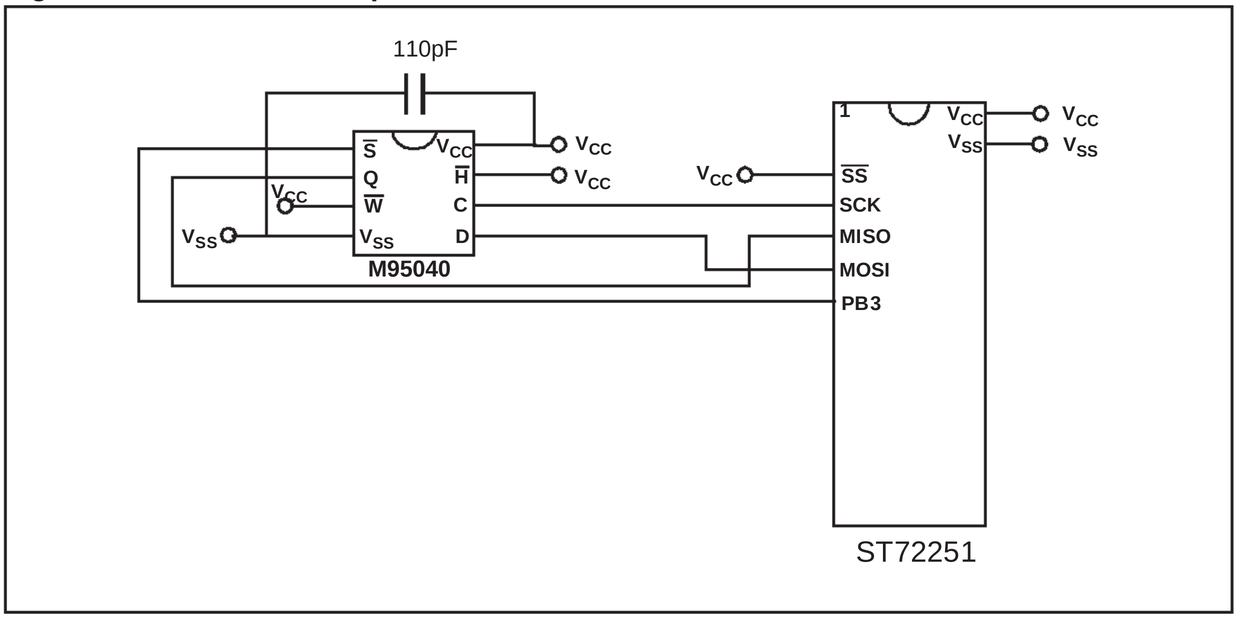
Finally, the software ends in an infinite loop.

5.2 HARDWARE DESCRIPTION

Two components were used for the application:

- one ST72251.
- one M95040 EEPROM.

Figure 2. Hardware description



$\overline{SS}$ pin of ST72251 is high (master device).

The SCK pin is connected with the serial clock pin (C) of the E PROMs.

The MOSI pin with the serial input pins (D).

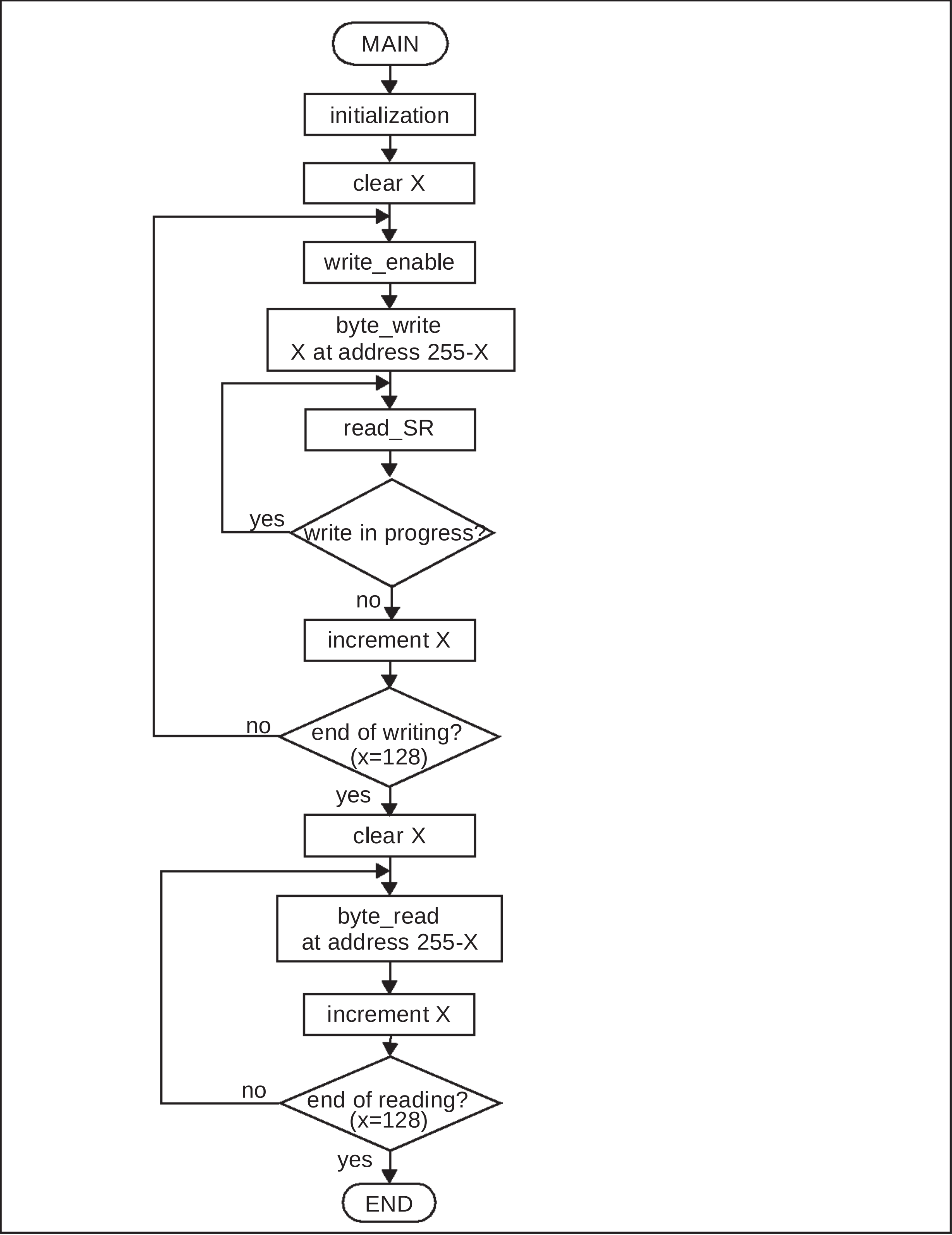
The MISO pin with the serial output pins (Q).

$\overline{S}$ pin of the EEPROM is connected with the PB3 output pin of ST72251.

5.3 FLOWCHARTS

The flowchart of the application is presented below (Figure 3.).

Figure 3. Flowchart: MAIN



5.4 USE OF SEVERAL EEPROMS

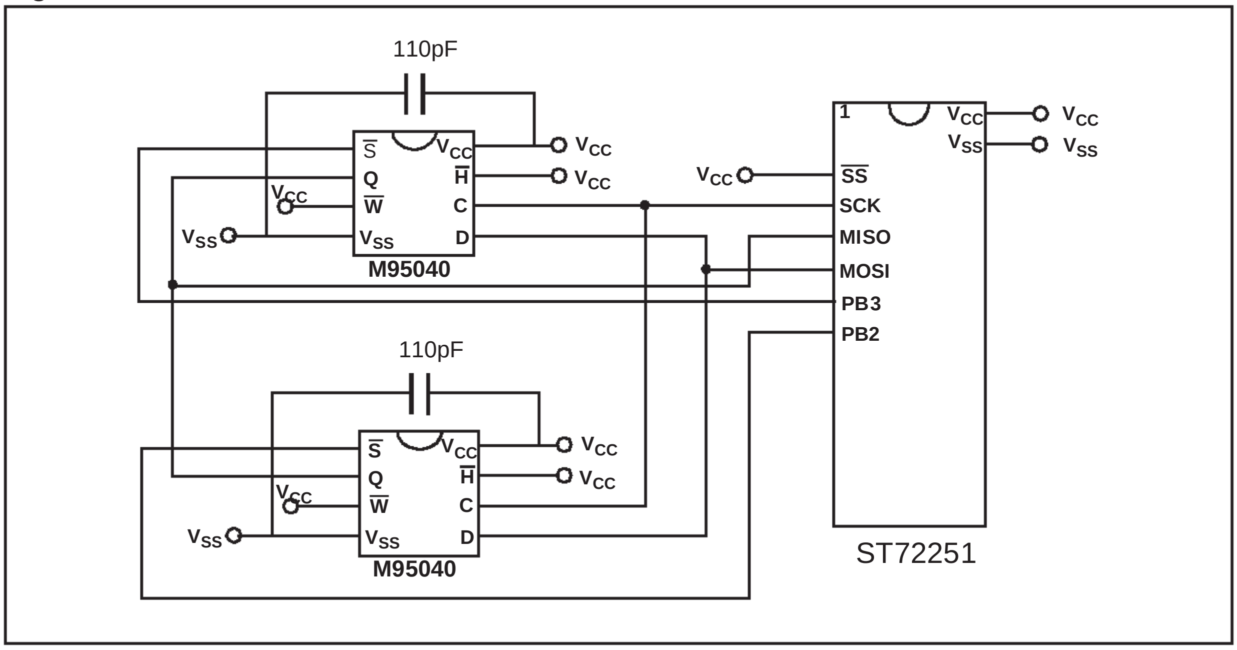
It is possible to communicate with several EEPROMs. Each one is selected in turn. Then the same protocol as described before is followed.

The main point is to **reserve one pin to select each device.**

A hardware schematic using two devices is given below (Figure 4.).

The selection of the devices is made using the PB2 and PB3 pins.

Figure 4. Hardware schematic for two devices



5.5 SOFTWARE

The assembly code given below is guidance only. The file cannot be used alone. The complete software can be found on the ST internet website.

```
st7/

;**************** (c) 1997 STMicroelectronics *********************
;
;PROJECT:           ST7 SPI COMMUNICATION WITH AN ST95040 EEPROM
;COMPILER:          ST7 ASSEMBLY CHAIN
;MODULE:            spi.asm
;CREATION DATE:     12/03/98
;AUTHOR:            PPG 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 SPI software driver for communication between a ST7
;                 and a ST95040 EEPROM.
;                 Polling software strategy without error management.
;
;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
;
; MODIFICATIONS:
;
; 12/03/98 V2.1 - New Assembly version.
;
;******************************************************************************


        TITLE    ”spi.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 st7225 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
;******************************************************************************

        #DEFINE dev_p 3             ; Select the device’s pin.
```

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

;*****************************************************************************
;       External routines   (defined elsewhere)
;*****************************************************************************
; routines

;*****************************************************************************
;         MACROs SUB-ROUTINES LIBRARY SECTION
;*****************************************************************************

;       (must be placed here and not at the end of the file)



        WORDS                      ; define subsequent addresses as words

    segment 'rom'


; Program code ***************************************************************

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

; ----------------------------------------------------------------------------
; ROUTINE NAME: ST7_init
; INPUT/OUTPUT: None.
; DESCRIPTION : Initialise ST7.
; COMMENTS    :
; ----------------------------------------------------------------------------
.ST7_init

                    bset  miscr,#0 ; Normal mode, clock_in = clock_out/2.
                    bset  pbddr,#3 ; PB3 output push pull but we cannot use
                    bset pbor,#3  ; a ld instruction because the rest of
                                  ; the port is used for the SPI function.
        ret

; ----------------------------------------------------------------------------
; ROUTINE NAME: SPI_init
; INPUT/OUTPUT: None.
; DESCRIPTION : Initialize SPI driver.
; COMMENTS    :
; ----------------------------------------------------------------------------
.SPI_init

        ld    A,#$5C      ; SPI= interrupt disable, output enable,ST7=Master
        ld    spicr,A     ; Polatiry=1 and Phase=1, clock= Core clock/8.
        ret
```

```
; -----------------------------------------------------------------------------
; ROUTINE NAME: write_enable
; INPUT/OUTPUT: Device’s pin / None.
; DESCRIPTION : Enable writing processes.
; COMMENTS    : Most significant bit first.
; -----------------------------------------------------------------------------
.write_enable

    ; Select device ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       bres  pbdr,#dev_p ; Tie to low E2PROM S pin.

    ; Send write enable instruction ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld     A,#$06
        ld     spidr,A ; The value is sent when put into the SPIDR.
 wait3  btjf   spisr,#7,wait3 ; Wait for SPIF bit to go up (data sent).
        ld     A,SPIDR; Second step to clear the SPIF bit.
    ; Deselect device ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        bset   pbdr,#dev_p ;Low to high transition on the S pin for the latch.
        ret

; -----------------------------------------------------------------------------
; ROUTINE NAME: byte_write_l
; INPUT/OUTPUT: Device’s pin , memory address, data value / None.
; DESCRIPTION : Write data value at memory address in lower page.
; COMMENTS    : Most significant bit first.
; -----------------------------------------------------------------------------
.byte_write_l

    ; Select device ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        bres  pbdr,#dev_p       ; Tie to low E2PROM S pin.

    ; Send write in lower page instruction ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld    A,#$02
        ld    spidr,A           ; The value is sent when put into the SPIDR.
wait4   btjf  spisr,#7,wait4    ; Wait for SPIF bit to go up (data sent).
       ld   A,SPIDR ;Second step to clear the SPIF bit.
    ; Send write memory address ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld    spidr,Y           ; Write at address 255-X.
wait5   btjf  spisr,#7,wait5    ; Wait for SPIF bit to go up (data sent).
       ld   A,SPIDR ;Second step to clear the SPIF bit.
    ; Write data ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld    spidr,X           ; Data = X.
wait6   btjf  spisr,#7,wait6    ; Wait for SPIF bit to go up (data sent).
       ld   A,SPIDR ;Second step to clear the SPIF bit.
    ; Deselect device ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        bset  pbdr,#dev_p  ; Low to high transition on the S pin for the latch.
        ret
```

```
; -----------------------------------------------------------------------------
; ROUTINE NAME: byte_read_l
; INPUT/OUTPUT: Device’s pin, memory address / data value.
; DESCRIPTION : Read memory value at memory address in lower page.
; COMMENTS    : Most significant bit first.
; -----------------------------------------------------------------------------
.byte_read_l

    ; Select device ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        bres  pbdr,#dev_p       ; Tie to low E2PROM S pin.

    ; Send read instruction ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld    A,#$03
        ld    spidr,A           ; The value is sent when put into the SPIDR.
wait9   btjf  spisr,#7,wait9    ; Wait for SPIF bit to go up (data sent).
       ld    A,SPIDR ;Second step to clear the SPIF bit.

    ; Send read memory address ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld    spidr,Y           ; Read at address 255-X.
wait10  btjf  spisr,#7,wait10   ; Wait for SPIF bit to go up (data sent).
       ld    A,SPIDR ;Second step to clear the SPIF bit.

    ; Generate 8 clock pulses ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld    A,#$D0
        ld    spidr,A           ; Send dummy value to generate the clock.
wait11  btjf  spisr,#7,wait11   ; Wait for SPIF bit to go up (data sent).

    ; Deselect device ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        bset  pbdr,#dev_p  ; Low to high transition on the S pin for the latch.
        ret

; -----------------------------------------------------------------------------
; ROUTINE NAME: write_SR
; INPUT/OUTPUT: Device’s pin, new status register value / None.
; DESCRIPTION : Write new value into status register.
; COMMENTS    : Most significant bit first.
; -----------------------------------------------------------------------------
.write_SR

    ; Select device ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        bres  pbdr,#dev_p       ; Tie to low E2PROM S pin.

    ; Send write into status register instruction ~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld    A,#1
        ld    spidr,A           ; The value is sent when put into the SPIDR.
wait1   btjf  spisr,#7,wait1    ; Wait for SPIF bit to go up (data sent).
       ld    A,SPIDR ;Second step to clear the SPIF bit.

    ; Write into status register ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld    spidr,X           ; New value of status register = X.
wait2   btjf  spisr,#7,wait2    ; Wait for SPIF bit to go up (data sent).
       ld    A,SPIDR ;Second step to clear the SPIF bit.

    ; Deselect device ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        bset  pbdr,#dev_p  ; Low to high transition on the S pin for the latch.
        ret
```

```
; ----------------------------------------------------------------------------
; ROUTINE NAME: read_SR
; INPUT/OUTPUT: Device’s pin / status register’s value.
; DESCRIPTION : Read the value of the status register and put it in SPIDR.
; COMMENTS    : Most significant bit first.
; ----------------------------------------------------------------------------
.read_SR

    ; Select device ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        bres  pbdr,#dev_p       ; Tie to low E2PROM S pin.

    ; Send read status register instruction ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld    A,#$05
        ld    spidr,A           ; The value is sent when put into the SPIDR.
wait7   btjf  spisr,#7,wait7    ; Wait for SPIF bit to go up (data sent).
       ld   A,SPIDR ;Second step to clear the SPIF bit.

    ; Generate 8 clock pulses ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld    A,#$0F
        ld    spidr,A           ; Send dummy value to generate the clock.
wait8   btjf  spisr,#7,wait8    ; Wait for SPIF bit to go up (data sent).

    ; Deselect device ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        bset  pbdr,#dev_p  ; Low to high transition on the S pin for the latch.
        ret


;       ********************************************
;       *                                          *
;       *          MAIN-ROUTINE SECTION           *
;       *                                          *
;       ********************************************


.main

    ; ST7 initialisation ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        call  ST7_init

    ; SPI initialisation ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        call  SPI_init

    ; Beginning of the application ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld    X,#$F0            ; No protected block in memory.
        call  write_SR
        clr   X                 ; Clear X.

    ; Write loop ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
.write_loop
        call write_enable
        ld    A,#255            ; 255 is the first address.
        ld    temp1,X           ; Copy of X register into temp1 register.
        sub   A,temp1           ; A = 255-X.
        ld    Y,A               ; Y = memory address.
        call  byte_write_l      ; Write X at address 255-X of the lower page.

    ; Wait for write in progress flag to get cleared ~~~~~~~~~~~~~~~~~~~~~~~~
again   call  read_SR
       btjt  spidr,#0,again     ; Waiting loop until WIP flag goes low.
                              ; Second step to clear the SPIF bit.
    ; End of writing sequence? ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
        inc   X                 ; Increment X.
        cp    X,#128            ; Have 128 bytes been written into the E2PROM?
        jrult write_loop        ; If not, still write into the E2PROM, else end.
        clr   X                 ; Clear X.

    ; Read loop ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
.read_loop
        ld    A,#255            ; 255 is the first address.
        ld    temp1,X           ; Copy of X register into temp1 register.
        sub   A,temp1           ; A = 255-X.
        ld    Y,A               ; Y = memory address.
        call  byte_read_l       ; Read at address 255-X of the lower page.

    ; Read data value ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ld    A,spidr           ; Get value from E2PROM.Second step to clear SPIF.
        ld    (start,X),A       ; Store it into the 256 bytes table.

    ; End of reading sequence? ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        inc   X                 ; Increment X.
        cp    X,#128            ; Have 128 bytes of the E2PROM been read?
        jrult read_loop         ; If not, still read the E2PROM, else end.

    ; Endless loop ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
loop    jra   loop


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


.dummy_rt        iret    ; Empty subroutine. Go back to main (iret instruction)
.spi_rt          iret    ; SPI Interrupt


;       *******************************************
;       *                                         *
;       *  CALL SUB-ROUTINES LIBRARY SECTION      *
;       *                                         *
;       *******************************************


                 segment 'vectit'

i2c_it:          DC.W    dummy_rt        ;FFF2-FFF3h location
timb_it          DC.W    dummy_rt        ;FFF4-FFF5h location
tima_it:         DC.W    dummy_rt        ;FFF6-FFF7h location
spi_it:          DC.W    spi_rt          ;FFF8-FFF9h location
io_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 ****
```