APPLICATION NOTE

PROGRAMMING ST7 FLASH MICROCONTROLLERS IN REMOTE ISP MODE (IN-SITU PROGRAMMING)

by Microcontroller Division Applications

1 INTRODUCTION

This application note is divided into two parts. The first part describes the ISP and FLASH programming specifications for the following ST7 devices:

– ST72C104

– ST72C124

– ST72C171

– ST72C215

– ST72C216

– ST72C254

– ST72C314

– ST72C334

– ST72C411 (supports two ISP protocols, refer to the datasheet).

The second part of this application note gives an example of how to use the ISP protocol to program the FLASH memory and the option bytes of a ST72C254, using another ST7 as a programming tool.

1.1 WHAT IS ISP?

You can program any of the MCUs listed above by inserting it in the socket of a programming tool available from STMicroelectronics (EPB). You can also program them, using a serial interface, in In-Situ Programming (ISP) mode.

The ISP feature allows you to update the content of Flash program memory when the chip is already plugged on the application board. ISP programming uses a serial protocol to interface a programming tool (which can be the EPB, or any other device that has the specifications described below).The ISP feature can be implemented with a minimum number of added components and board area impact.

The ISP serial communication is based on a Master/Slave Architecture where the master is the ST7 to be programmed. So the clock speed depends on the speed of the ST7 CPU and this fact can produce some timing constraints.

2 ISP SPECIFICATIONS

2.1 ISP HARDWARE DESCRIPTION

In remote ISP mode, the ST7 has to be supplied with power (V_{DD} and V_{SS}) and an internal or external clock signal (you can use any of the oscillator configurations described in the datasheet). This mode needs five signals (plus the V_{DD} signal if necessary) to be connected to the programming tool. These signals are:

– $\overline{\text{RESET}}$: device reset
– V_{SS}: device power supply ground
– ISPCLK: ISP serial clock output
– ISPDATA: ISP serial data input
– ISPSEL: Remote ISP mode selection. This pin must be connected to V_{SS} on the application board through a pull-down resistor.

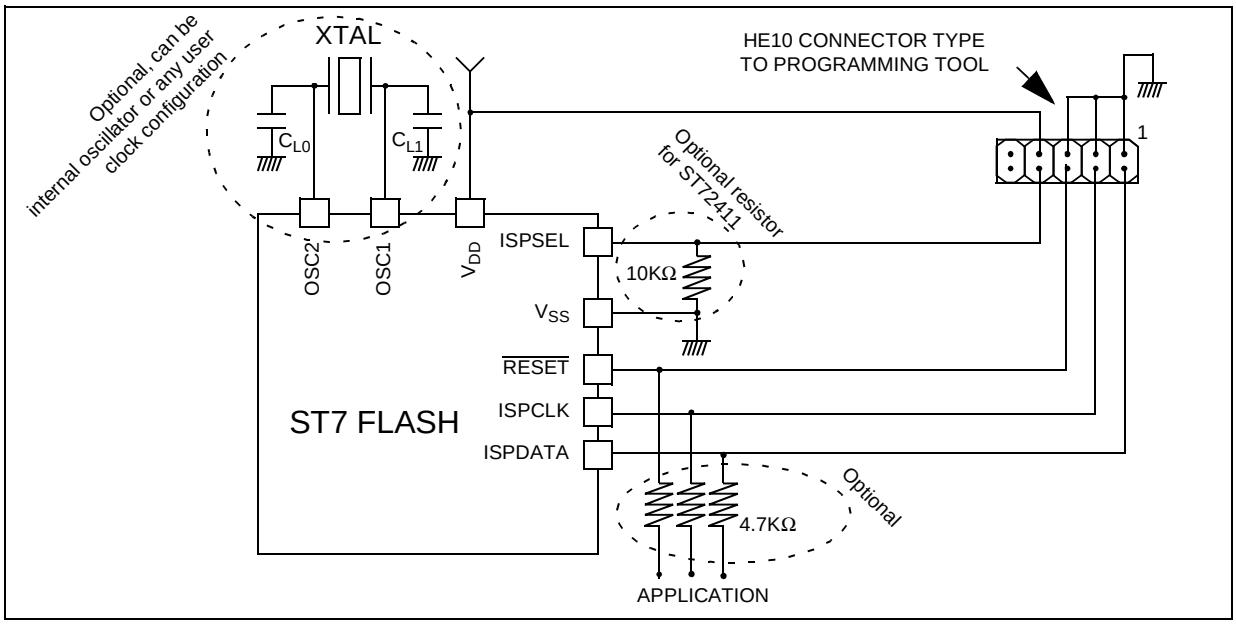
These pins are connected to the ST Programming tool using a HE10 type connector .

ISP mode used pin	ISPDATA	ISPCLK	ISPSEL	$\overline{\text{RESET}}$	V_{SS}	V_{DD}
HE10 connector pin number	2	4	8	6	1,3,5	7

If any of these pins are used for other purposes in the application, a serial resistor can be implemented to avoid a conflict if the other device forces the signal level.

Figure 1 shows a typical hardware interface to a standard ST7 programming tool. For more details on the pin locations, refer to the device pinout description of the ST7.

Figure 1. Typical Remote ISP Interface



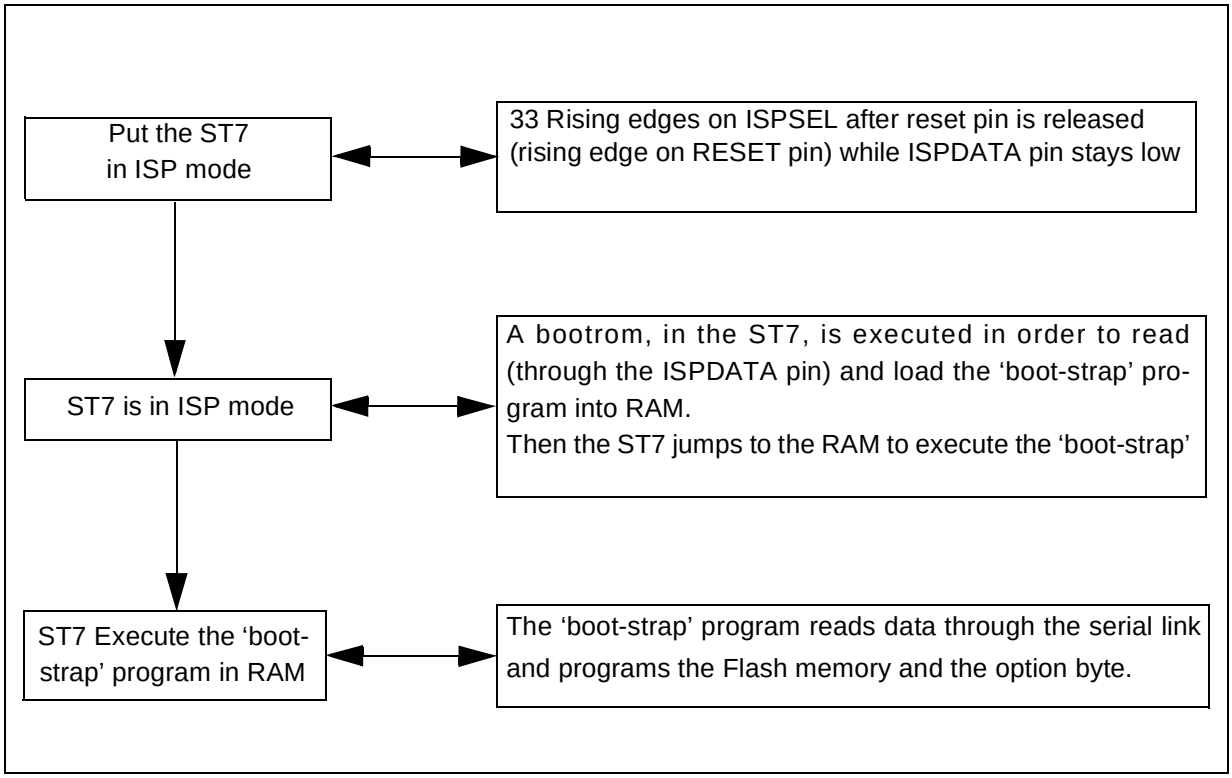
2.2 ISP FUNCTIONAL DESCRIPTION

The ISP mode is selected by a specific sequence on the ISPSEL pin. ISP is performed in three steps and makes use of the ST7's capability of executing RAM-resident code:

– Selection of ISP mode
– Download code in RAM
– Execution of the downloaded code in RAM to program the user program into the FLASH

The programming sequence using the ISP protocol is described in Figure 2.

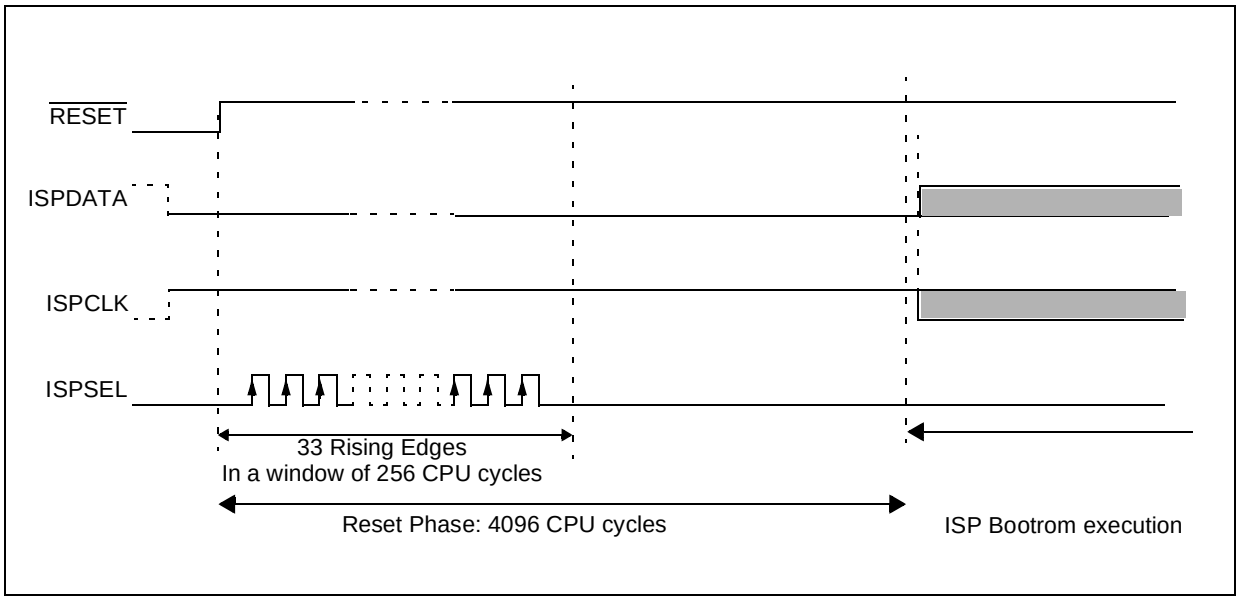
Figure 2. Flowchart of the ISP Protocol .



2.2.1 ISP MODE SELECTION

To enter ISP mode the ST7 needs to be powered-on and connected to an available clock system (internal RC, external RC, external clock or Crystal/resonator). The factory configuration is internal RC (for clock selection refer to the datasheet).
During the reset phase, a sequence on the ISPSEL pin is used to enter ISP mode. ISPDATA must be tied low until the first rising edge of ISPCLK. The sequence is shown in Figure 3.

Figure 3. ISP mode selection



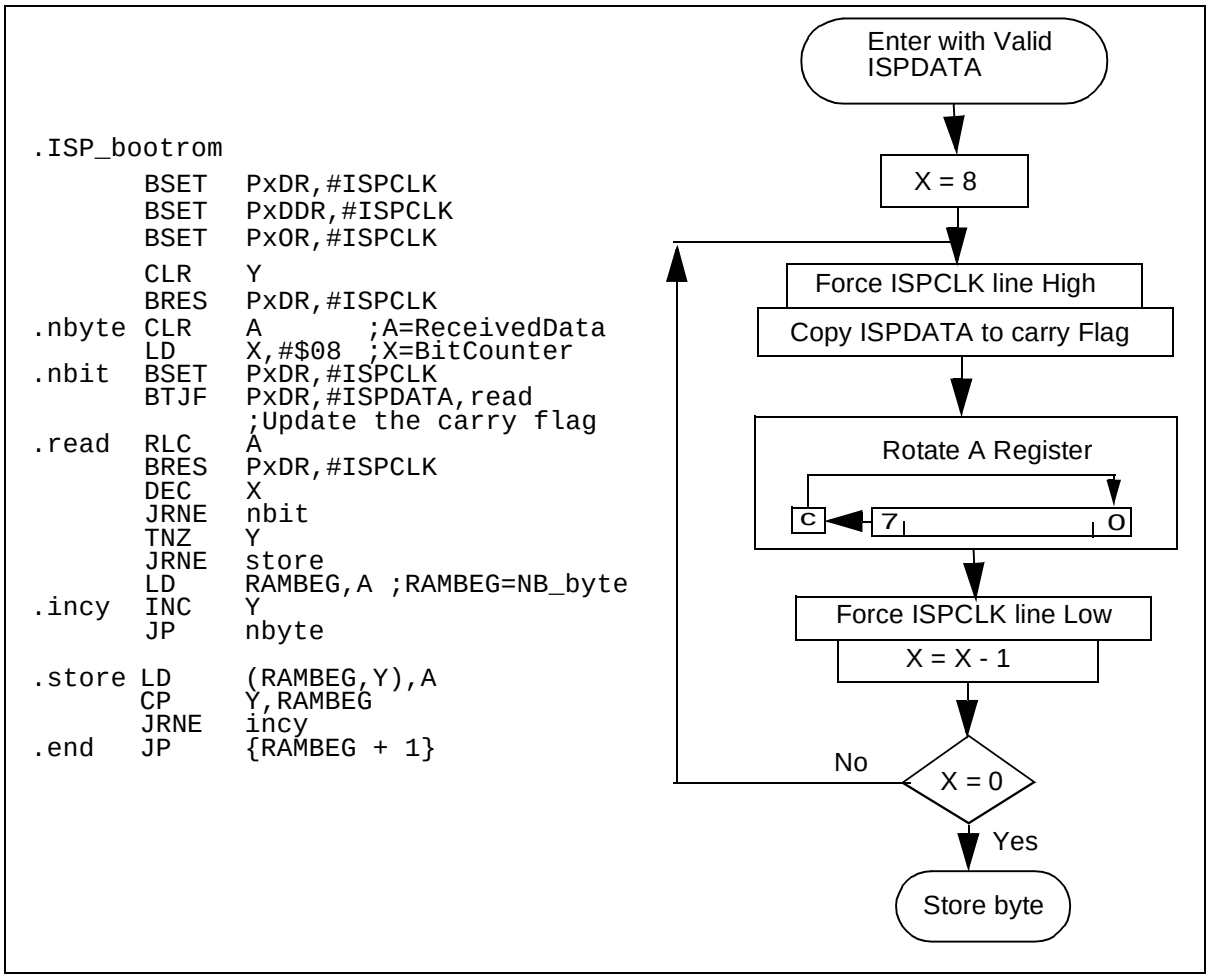
Note: The 33 pulses do not need to be synchronized with the CPU clock. In this mode, the reset phase is still 4096 CPU cycles.

2.2.2 DOWNLOADING CODE IN RAM

At the end of the Reset phase, the reset vector is fetched into the ISP bootrom.
The bootrom code is executed, to configure the port logic and to receive data serially through the ISPDATA pin to be stored in the RAM area. To execute this ISP boot program, the status of the ISPDATA pin must externally forced LOW just after the RESET phase, until the first rising edge of ISPCLK.
Downloading the code in the RAM area is done sequentially from the least significant address of the RAM. The number of bytes to be downloaded (after the first one) is specified in the first data byte transfer and so can not exceed 255 bytes.

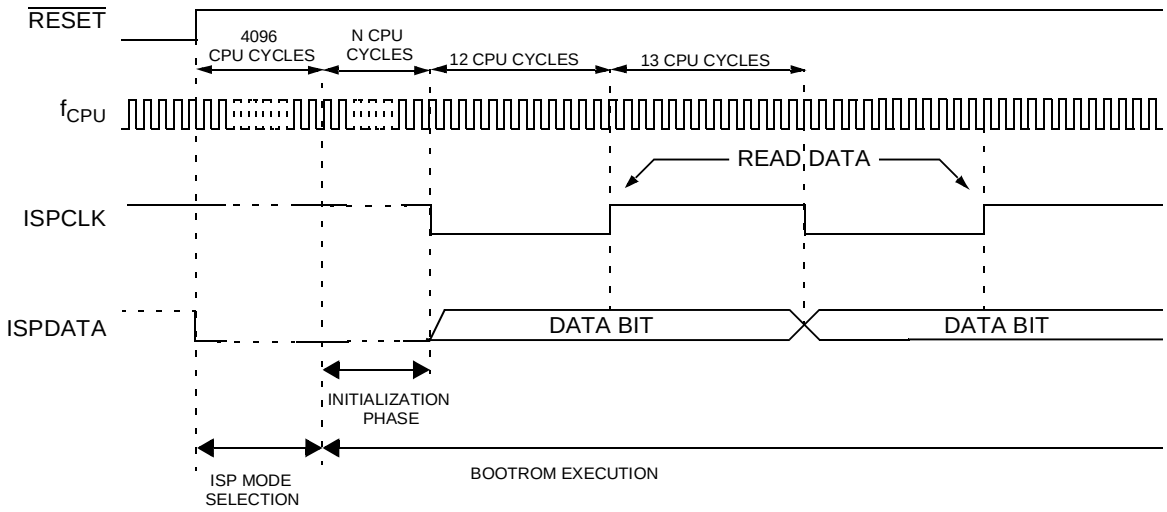
The ISP bootrom program and ISP protocol flowchart are shown in Figure 4.

Figure 4. ISP Bootrom: Code & Flowchart



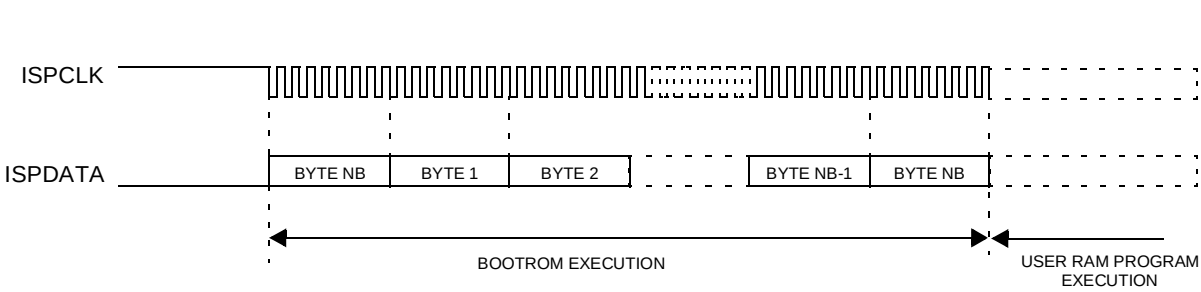
A complete timing diagram is shown in Figure 5 and Figure 6Figure 6.

Figure 5. ISP bit communication



Note: The initialization phase duration is device dependent, timings are given in Section 2.2.3.4

Figure 6. ISP byte communication



2.2.3 EXECUTING CODE IN RAM, FLASH PROGRAMMING

After downloading the program in RAM, jumping to the start address of this program starts the FLASH programming operation.

The FLASH program memory is organised as a single 8-bit wide memory block which can be used for storing both code and data constants. The FLASH program memory is mapped in the upper part of the ST7 addressing space and includes the reset and user interrupt vector area.

2.2.3.1 Programming the FLASH

The FLASH area is driven by the Flash Control & Status Register (EEXCSR).
The FLASH area can program up to 16 bytes in the same erase & programming cycle.The FLASH is mono-voltage, a charge pump generates the high voltage internally to enable the erase and programming cycles. The global programming cycle duration is controlled by an internal circuit.

Flash Control & Status (EEXCSR) Register description

Reset Value: 0000 0000 (00h)

7							0
0	0	0	0	0	OPT	LAT	PGM

– Bit 7:3 = Reserved, forced by hardware to 0.

– Bit 2 = **OPT** *Option byte access*
Option byte enable at high level. It allows read and write access to the option bytes in ISP mode. The two option bytes are selected by their own address (see table below).
0: User FLASH program area selected
1: Option Bytes area selected

– Bit 1 = **LAT** *Latch Access Transfer*
This bit is set by software. It is cleared by hardware at the end of the programming cycle. It can only be cleared by software if the PGM bit is cleared.
0: Transfer latches not accessible
1: Transfer latches accessible

– Bit 0 = **PGM** *Programming control and status*
This bit turns on the charge-pump. This bit must be set to start the programming cycle. At the end of the programming cycle, this bit is automatically cleared, stopping the charge pump.
0: Programming finished or not yet started
1: Programming cycle is in progress

The EEXCSR address and the option byte addresses are device dependent and summarized in the following table:

ST7 FLASH	ST72C104	ST72C124	ST72C171	ST72C215	ST72C216	ST72C254	ST72C314	ST72C334	ST72C411
EEXCSR	0026h	002Dh	0026h	0026h	0026h	0026h	002Dh	002Dh	0026h
Option Byte 1	E000h	C000	E000h	E000h	E000h	E000h	C000	C000	F000
Option Byte 2	E001h	C001	E001h	E001h	E001h	E001h	C001	C001	-

Read Operation (LAT=0)

The FLASH can be read as a normal ROM location when the LAT bit of the EEXCSR register is cleared.

Write Operation (LAT=1)

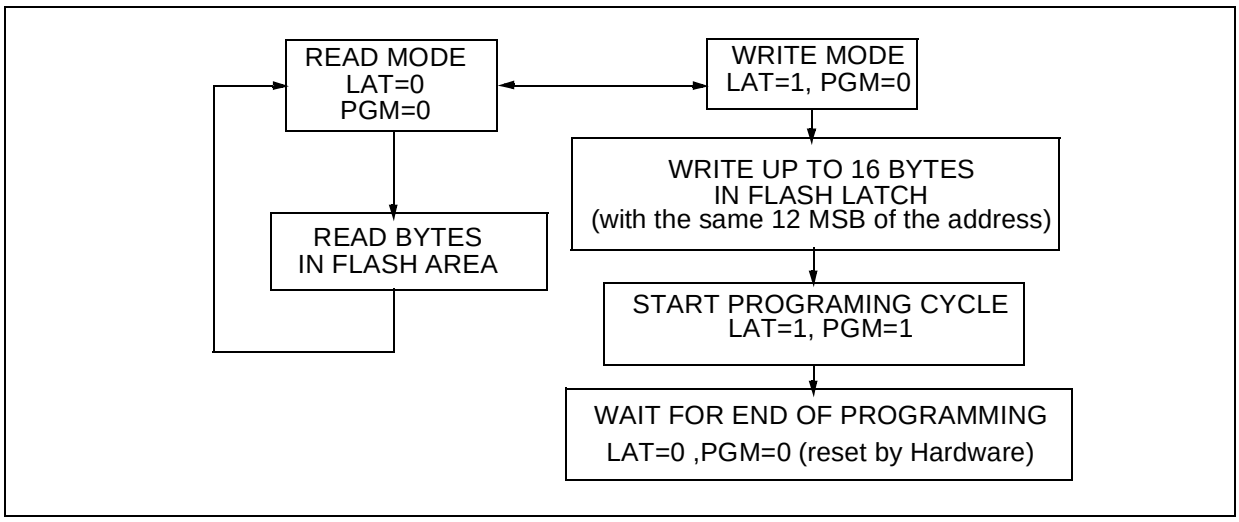
To access the write mode, the LAT bit has to be set by software (the PGM bit remains cleared). When a write access to the FLASH area occurs, the 8-bit data bus is memorized in one of the 16 8-bit data latches. The data latches are selected by the lower part of the address (A<3:0> bits).

When the PGM bit is set by software, all the previous bytes written in the data latches (up to 16) since the last programming, are programmed in the FLASH cells. The effective high address (A<15:4> bits) is determined by the last FLASH write sequence.
If 16 consecutive write instructions are executed by sweeping from A<3:0>=0h to A<3:0>=Fh, with the same higher part of the address (A<15:4> bits), the 16 data latches will be written in the same row of the FLASH matrix. At the end of the programming cycle, the LAT bit is automatically reset, and the 16 data latches are cleared.

To avoid wrong programming, the user must take care that all the bytes written between 2 programming sequences have the same high address: only the four Least Significant Bits of the address can change.

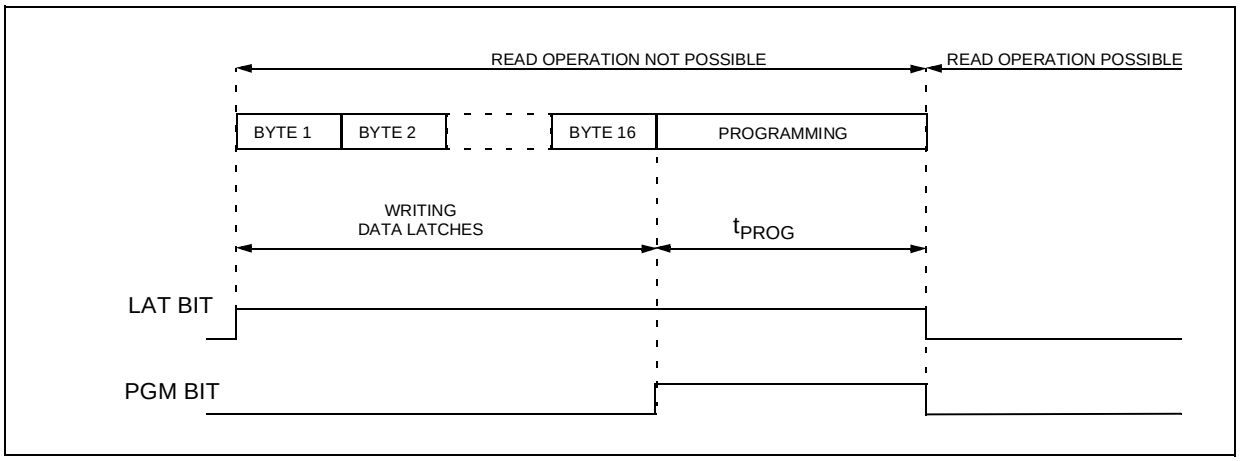
Note: If only N (N<17) write instructions are executed before the PGM bit is set, only the N bytes of the matrix will be written.

Figure 7. FLASH Programming Flowchart



Note: If a programming cycle is interrupted (by software/RESET action), the data integrity in memory will not be guaranteed.

Figure 8. FLASH Programming Cycle



2.2.3.2 Programming the option bytes

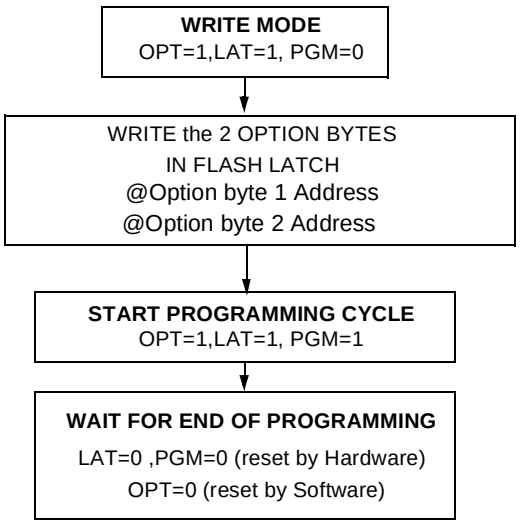
Some ST7 devices have Option Bytes for configuring the device features.
The Option Bytes can be only accessed in ISP mode, they can be programmed by writing the corresponding byte (byte row is included inside FLASH Matrix and can be managed as a normal FLASH row). For more details refer to FLASH Control and Status register description. In ROM devices the Option Bytes are fixed in hardware by the ROM code.
The following is an example algorithm for programming devices with up to two option bytes using the standard ISP protocol.

Figure 9. Option Bytes Programming Code & Flowchart

```
NBCOL     BYTE $2       ;2 option bytes

          CLR      Y
          BRES     PxDR,#ISPCLK
.nbyte    CLR      A     ;A=ReceivedData
          LD       X,#$08;X=BitCounter
.nbit     BSET     PxDR,#ISPCLK
          BTJF     PxDR,#ISPDATA,read
.read     RLC      A
          BRES     PxDR,#ISPCLK
          DEC      X
          JRNE     nbit
          BSET     EEXCSR,#OPT
          BSET     EEXCSR,#LAT
          LD       EEPBEG,Y),A
          INC      Y
          CP       Y, NBCOL
          JRNE     .nbyte

.store    BSET     EEXCSR,#PGM
          BTJT     EEXCSR,#LAT,*
          BRES     EEXCSR,#OPT
.end
```



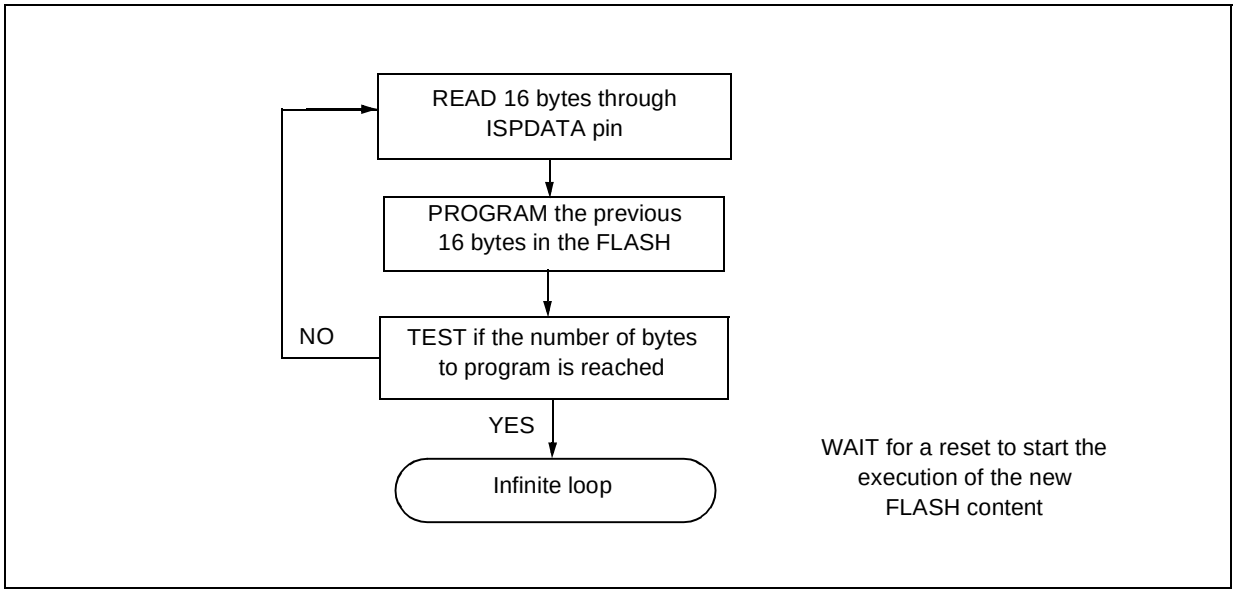
2.2.3.3 Bootstrap Program content

Depending on the downloaded bootstrap program in RAM, the memory programming can be completely customized (number of bytes to program, location of the program, programming of option bytes, or selecting any other serial communication interface for downloading). You can also use the bootstrap program to run a test program in RAM even if you are using a ROM device.

The example given in the second part contains a bootstrap program able to program 8 Kbytes of FLASH and the option bytes of the ST72254 reading in the data to be programmed through the I/O port. In most ST7 devices, you can also use the SPI protocol to read in the data to be programmed in the FLASH. Refer to the SPI protocol example given later in this document.

Whatever the selected protocol, the bootstrap program will always execute the same algorithm shown in Figure 10.

Figure 10. Bootstrap program algorithm.



SPI protocol:

To use this protocol, the SPI has to be configured as follows:

– The $\overline{SS}$ pin should be configured in Master mode.

– The SPI Master output (MOSI) pin should be disabled.

For more details refer to the SPI chapter of the datasheet and the Miscellaneous Register (where available).

The following is a bootstrap code example for programming the FLASH using the SPI protocol.

```
COUNT  BYTE number   ;byte_number
NBCOL  BYTE $10      ;number of latch

.spi   LD           A,#$0B            ;pinNSS soft
       LD           MSCR2,A;set SMOD bit
       LD           A,#$5C
       LD           SPICR,A;SPI
       CLR          Y
.main  LD           SPDAT,X
.rbyte BTJF         SPSR,#SPIF,rbyte
       LD           A,SPDAT
       BSET         EEXCSR,#LAT
       LD           (EEPBEG,Y),A
       INC          Y
       CP           Y,NBCOL
       JRNE         loop
       CALL         store

.loop  CP           Y, COUNT
       JRNE         rbyte
       BTJF         EEXCSR,#LAT,end
       CALL         store
.end   BRES         SPCR,#SPE ; stop spi

.store BSET         EEXCSR,#PGM
       BTJT         EEXCSR,#LAT,*
       LD           A,NBCOL
       ADD          A,#$10
       LD           NBCOL,A
       RET
```

2.2.3.4 ISP Programming Time

The time required to program the FLASH using the ISP protocol can be estimated using the following table:

Table 1. ISP programming time evaluation table

ISP mode selection/ reset	Downloading code in RAM Bootrom Execution	FLASH Programming Bootstrap Execution		
		Data Transfer		Flash Programming
		I/O Protocol	SPI Protocol	
$4096 * T_{CPU}$	$25 * 8 * N_{byteR} * T_{CPU} + T_{init}$	$8 * 25 * T_{CPU} * N_{byteF}$	$4 * T_{CPU} * N_{byteF}$	$T_{prog} * N_{byteF} /16$

Where:

T_{CPU} is the CPU period in seconds (down to 0.125µs),

N_{byteR} is the number of bytes able to be downloaded in RAM (up to 255)

N_{byteF} is the number of bytes to be programmed in FLASH (up to 16 Kbytes i.e. 16386)

T_{prog} is the Typical Flash erase and program cycle (8ms)

T_{init} is the time between the end of the reset phase and the first rising edge on ISPCLK. T_{init} is device dependent)

Table 2. Initialization time (T_{init}) according the ST7.

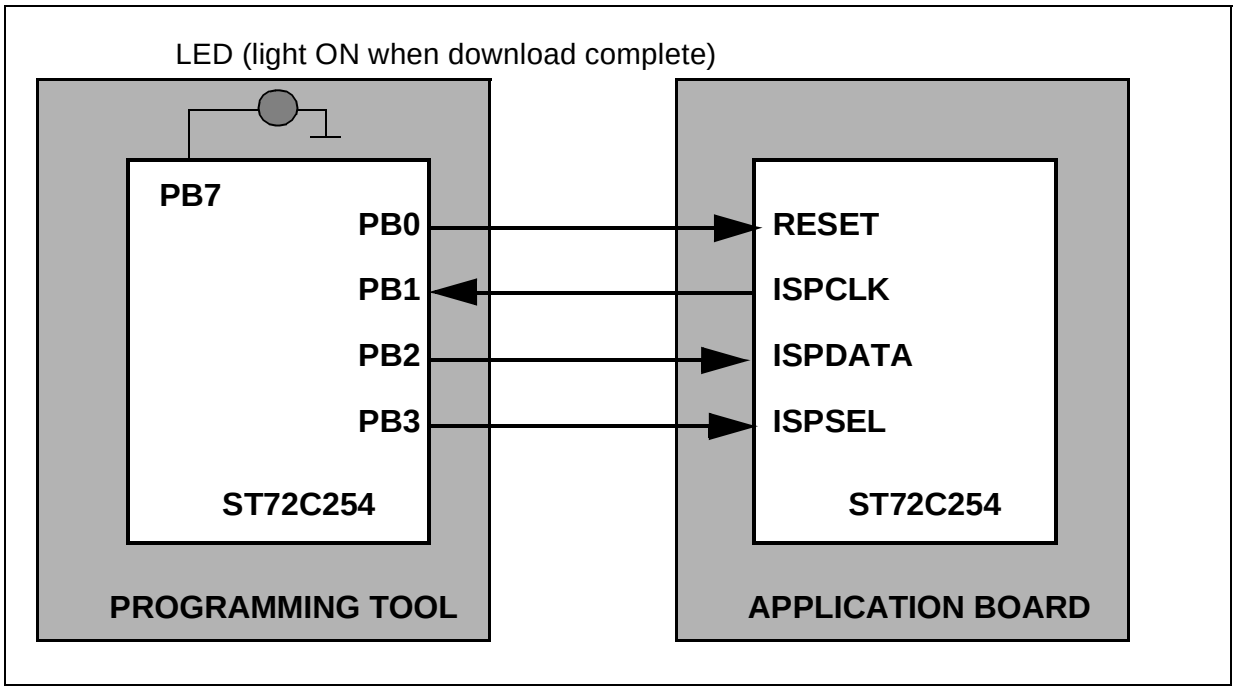
ST7 FLASH	ST72C104	ST72C124	ST72C171	ST72C215	ST72C216	ST72C254	ST72C314	ST72C334	ST72C411
T_{init} (CPU clock)	45	45	45	45	45	45	45	45	1602 ISP_{SC} 1609 ISP_N

3 ST72C254 ISP PROGRAMMING EXAMPLE

3.1 APPLICATION EXAMPLE

The example described here shows how to program the ST72C254 using ISP. The downloading is done from another ST72C254 (acting as a programming tool). The boot-strap program described in this document, programs the FLASH and the Option bytes of the ST72254 using the ISP protocol (not the SPI protocol).

Figure 11. Application Circuit



In this example, there is no external clock on the application board because the ST72C254 is configured to use its internal oscillator: a 4 MHz RC oscillator (T_{CPU}=250ns).

As shown in Section 2.2, ISP is performed in three steps:

– Select ISP mode
– Download code in RAM
– Execute code in RAM

3.1.1 Select ISP mode

Entering ISP mode is done by generating 33 pulses on the ISPSEL pin during the reset phase (window of 256 cycles).

3.1.2 Downloading code in RAM

After selecting ISP mode, the ST72C254 fetches the reset vector into a common bootrom. The bootrom which is then executed, receives data serially through the ISPDATA pin and stores it in the RAM area. Downloading the code in RAM is done sequentially from the lsb address of the RAM (0x0080 in the ST72254).

The number of bytes to be downloaded is specified in the first byte transfer (and stored in address 0x0080 of the RAM). The number of downloaded bytes can not exceed 255.

The transmission speed is controlled by the ST72C254 by generating ISPCLOCK. The PB1 pin of the programming tool is configured as input with pull-up in order to tie the ISPCLK line high until the ISPCLK drives the line. When the ST72C254 is ready to read the first bit, it generate a low transition on ISPCLK. The programming tool sends the first bit. The ST72C254 generate a high transition and stores this bit.

3.1.3 Executing code in RAM

After downloading the code in RAM, the downloaded program (boot-strap program) is executed. Depending on the downloaded program in RAM, the memory programming can be completely customized.

In this example, the bootstrap program is written to read 8 Kbytes through the ISPDATA pin (using the I/O protocol) and program them in the FLASH. The Bootstrap also programs the 2 option bytes with a predefined value.

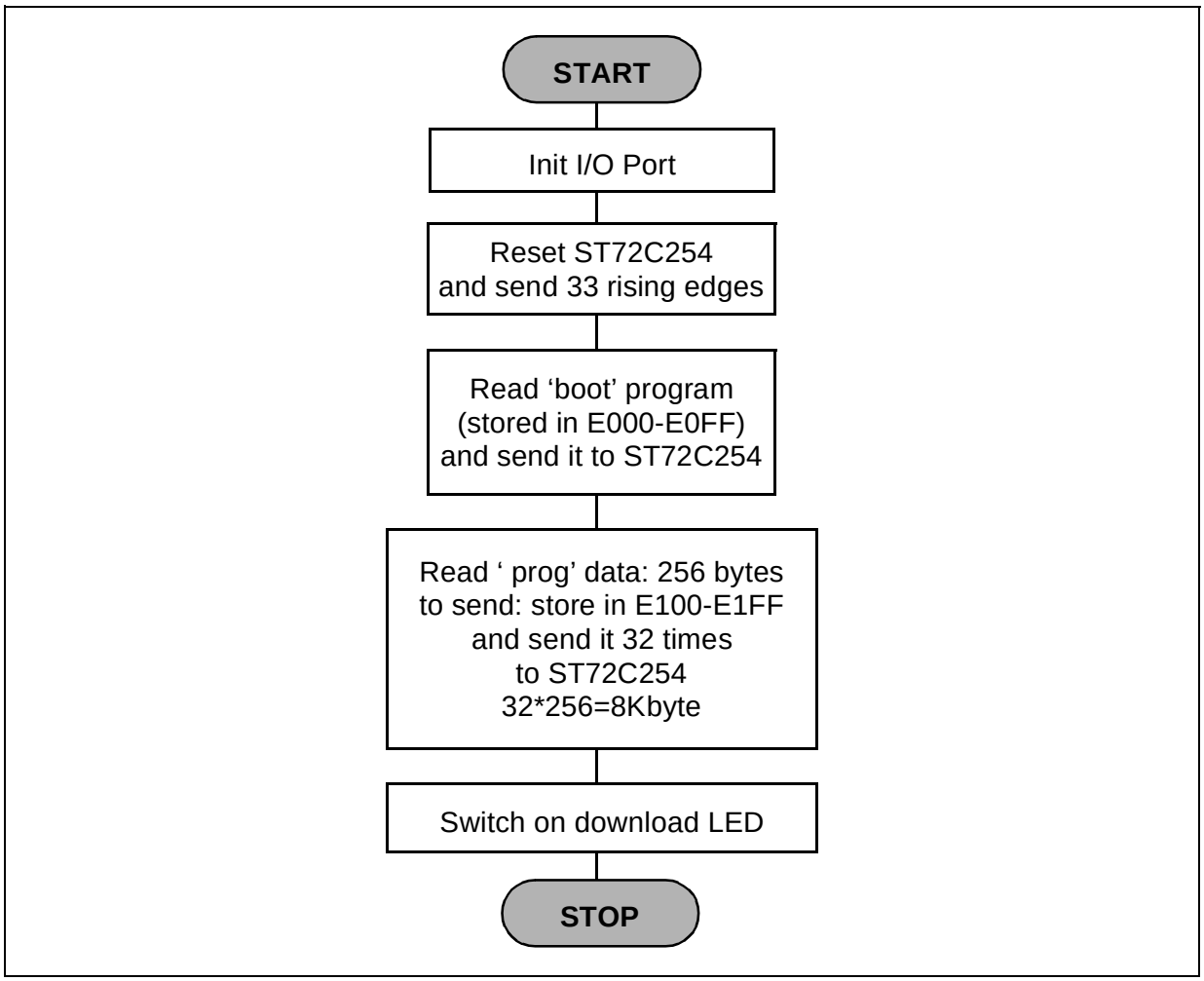
The bootstrap code located in the file ‘bootstra.asm’ is located in ROM at the address E000h-E0FFh. This code will be executed in the RAM of the ST72C254 to be programmed, so the absolute label value will be wrong. To get around this, you can use the following directives.

```
segment byte at 81-14F 'ramexe254'
segment 'ramexe254 > rom_boot'
```

The code following these directives will be place in the segment `‘rom_boot’` but it will be linked to be executed in the segment `‘ramexe254’`.

3.2 CODE & FLOWCHART

3.2.1 Programming tool flowchart



3.2.2 Programming tool code

```
ST7/
;******************************************************************************
; TITLE             LOAD_ISP.ASM
; AUTHOR            ST7 Application Group
; DESCRIPTION       Program ST72254 in ISP mode
; Note:             THIS FILE MUST BE LINKED BEFORE BOOTSTRA.ASM
;******************************************************************************

    TITLE  "LOAD_ISP.ASM"


    #INCLUDE "ST72254.INC"

      #define      ISPDATA     2
      #define      ISPCLK      1

NB_BYTEEQU         $FF         ; Number of bytes loaded in RAM of ST72254

      BYTES
   segment 'ram0'
PTR   DS.B         1

      WORDS
  segment byte 'rom_boot'
.boot  DC.B        {NB_BYTE-1}
                   ; The address 'boot' must be place at the first address of the
                   ; segment 'rom_boot'. So this file must be linked before the
                   ;file 'bootstra.obj' which uses also the segment 'rom_boot'
;*******************************************************************************
;    MAIN PROGRAM
;      ============
;
;*******************************************************************************

    segment 'rom3'

.main  LD     A,#32        ; Init PTR to 32, In this example, 256 bytes
       LD     PTR,A        ; will be send 32 times (256 * 32 = 8 Kbytes)
       LD     A,#%10001101 ; \ PB7 -> OUT -> Operation OK
       LD     PBDDR,A      ; | PB3 -> OUT -> Conn to ISPSEL of ST72254
       LD     A,#%10001111 ; | PB2 -> OUT -> Conn to ISPDATA of ST72254
       LD     PBOR,A       ; / PB1 -> IN Pull-Up -> Conn to ISPCLK of ST72254
       BRES   PBDR,#ISPDATA ;PB0 -> OUT -> Conn to RESET of ST72254
```

```
       CALL   tempo         ;
       LD     X,#33         ; \ 33 Pulses to enter ISP mode
       BSET   PBDR,#0       ; | RESET  = 1
edges  BSET   PBDR,#3       ; | ISPSEL = 1
       BRES   PBDR,#3       ; | ISPSEL = 0
       DEC    x             ; | 16*33 = 528 cycles
       JRNE   edges         ; /
       CALL   send_prog_in_ram
       CALL   send_prog_in_eeprom
       BSET   PBDR,#7       ; Inform that Programming is complete
end_f  JP     end_f         ; Infinite loop

;*********************************************************************
;    TEMPO PROGRAM
;      ==============
;
;*********************************************************************

.tempo LD     A,#$80
tp     DEC    A
       JRNE   tp
    ret

;*********************************************************************
;    SEND 1 BYTE ROUTINE
;      ====================
;
;*********************************************************************

.send_1_byte
       LD     X,#$08        ; Receive 8 bits
trans1 BTJT   PBDR,#ISPCLK,trans1     ; low transition
       RLC    A             ; A.0 put in carry
       JRC    emi1          ;
emi0   BRES   PBDR,#ISPDATA; Data = 0
       JRA    trans2        ;
emi1   BSET   PBDR,#ISPDATA; Data = 1
trans2 BTJF   PBDR,#ISPCLK,trans2      ; high transition
       DEC    X             ; decrement counter
       JRNE   trans1        ; All bits sent ?
       RET
```

```
;*************************************************************************
;    SEND PROG IN ST72254 RAM
;      ========================
;*************************************************************************
.send_prog_in_ram
       CLR   Y             ; Clear Y
data   LD     A,(boot,Y)   ; Read BOOT_PROG at adr 80h of 254
       CALL  send_1_byte  ; Send each byte
       INC   Y             ;
       CP    Y,#NB_BYTE   ; All bytes sent
       JRNE  data          ; If not, continue
       RET
```

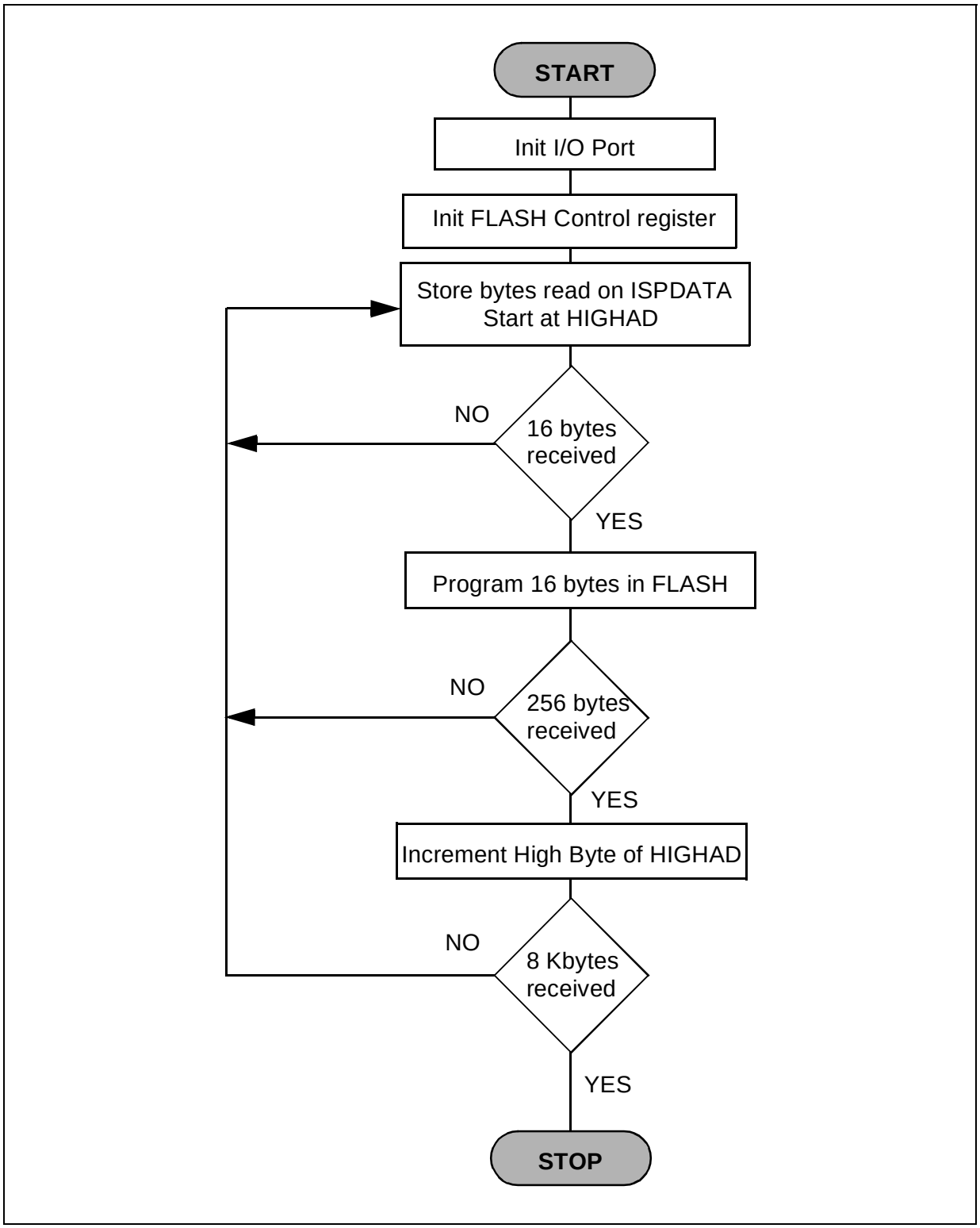
```
;************************************************************************
;    SEND PROG IN ST72254 EEPROM ROUTINE
;      ==================================
;;************************************************************************
.send_prog_in_eeprom      ;
data2  CLR   Y            ; Reset pointer
data1  LD    A,(prog,Y)   ; Read byte to send
       CALL  send_1_byte  ; Send byte
       INC   Y            ; Go to the next byte
       CP    Y,#$00       ; 256 bytes sent ?
       JRNE  data1        ; If not, continue
       DEC   PTR          ; Else, Dec PTR, send 256 bytes
       JRNE  data2        ; All 8 Kbytes loaded ? If not, continue
       RET

;***************************************************************************
;
; CODE THAT WILL BE PUT IN EEPROM OF THE ST72254
;
;***************************************************************************
*

       segment 'rom2'

.prog  DC.B  $20,$20,$20,$53,$54,$37,$20,$4D,$69,$63,$72,$6F,$73,$20,$20,$20
       DC.B  $46,$4C,$41,$53,$48,$20,$50,$52,$4F,$47,$52,$41,$4D,$49,$4E,$47
       DC.B  $20,$20,$20,$53,$54,$37,$20,$4D,$69,$63,$72,$6F,$73,$20,$20,$20
       DC.B  $46,$4C,$41,$53,$48,$20,$50,$52,$4F,$47,$52,$41,$4D,$49,$4E,$47
       DC.B  $20,$20,$20,$53,$54,$37,$20,$4D,$69,$63,$72,$6F,$73,$20,$20,$20
       DC.B  $46,$4C,$41,$53,$48,$20,$50,$52,$4F,$47,$52,$41,$4D,$49,$4E,$47
       DC.B  $20,$20,$20,$53,$54,$37,$20,$4D,$69,$63,$72,$6F,$73,$20,$20,$20
       DC.B  $46,$4C,$41,$53,$48,$20,$50,$52,$4F,$47,$52,$41,$4D,$49,$4E,$47
       DC.B  $20,$20,$20,$53,$54,$37,$20,$4D,$69,$63,$72,$6F,$73,$20,$20,$20
       DC.B  $46,$4C,$41,$53,$48,$20,$50,$52,$4F,$47,$52,$41,$4D,$49,$4E,$47
       DC.B  $20,$20,$20,$53,$54,$37,$20,$4D,$69,$63,$72,$6F,$73,$20,$20,$20
       DC.B  $46,$4C,$41,$53,$48,$20,$50,$52,$4F,$47,$52,$41,$4D,$49,$4E,$47
       DC.B  $20,$20,$20,$53,$54,$37,$20,$4D,$69,$63,$72,$6F,$73,$20,$20,$20
       DC.B  $46,$4C,$41,$53,$48,$20,$50,$52,$4F,$47,$52,$41,$4D,$49,$4E,$47
       DC.B  $20,$20,$20,$53,$54,$37,$20,$4D,$69,$63,$72,$6F,$73,$20,$20,$20
       DC.B  $46,$4C,$41,$53,$48,$20,$50,$52,$4F,$47,$52,$41,$4D,$49,$4E,$47
```

3.2.3 Bootstrap program flowchart



3.2.4 Bootstrap program code

```
ST7/
;******************************************************************************
; TITLE              BOOTSTRA.ASM
; AUTHOR             ST7 Application Group
; DESCRIPTION        Bootstrap loaded Program used by ISP programming
;
; PB5 = ISPDATA
; PB6 = ISPCLOCK
;
; ***** CHANGE LOADED ADRESS *****
; - Change start address in (1)
; - Change low part of the address in (2) and in (4)
; - Change High part of the address in (3)
;
; ***** Change OPTION BYTE *****
; - Change Option byte 1 in (5)
; - Change Option byte 2 in (6)
;
;******************************************************************************

       TITLE  "LOAD_ISP.ASM" ;
       #INCLUDE "ST72254.INC"

       segment byte at 81-14F 'ramexe254'
       segment 'ramexe254>rom_boot'
                    ; the following code is located in the segment 'rom_boot'
                    ; but all the labels are calculated to be placed
       BYTES        ; in the RAM of the ST72254 starting at the address 80h.
                    ; Boot code program start at 81h.
                    ; 1st byte at 80h-> Nb of bytes
.HIGHADDC.W  $E000        ; (1) Adr where prog will be load in FLASH
.LOWAD DC.B  $00          ; (2) Copy of the low part of the address
.IDCODEDC.B  $02          ; Not use in this version
.COUNT DC.B  $00          ; Variable COUNT

       WORDS

;******************************************************************************
;    MAIN PROGRAM OF THE BOOT
;      ========================
;
;******************************************************************************
```

```
.begin
       RSP
       LD     A,#$40        ; \  PB6 (SCK) is output
       LD     PBDDR,A       ; |
       LD     PBOR,A        ; /
       BRES   PBDR,#6       ; ISPCLOCK = 0
        LD    A,EEXCSR      ; read continuously
        AND   A,#$F8        ; Mask reserved bit
       CALLR  eep_prg       ; Program FLASH
       CALLR  ob_prg        ; Program Option Byte
inf_loop
       JRA    inf_loop      ; Infinite loop

;*********************************************************************
;    OPTION BYTE LOADED PROGRAM
;      ==========================
;
;*********************************************************************

.ob_prg
       LD     A,#$E0        ; (3) High part ----> CHANGE HERE
       LD     HIGHAD,A      ;
       LD     X,#$00        ; (4) Low part -----> CHANGE HERE
       LD     {HIGHAD+1},X
       BSET   EEXCSR,#2     ; OPT=1
       BSET   EEXCSR,#1     ; LAT=1
OB_1   LD     A,#$FC        ; (5) OB_1 ---------> CHANGE HERE
       LD     ([HIGHAD.w],X),A; Store OB_1
       INC     X            ; Go to the second OB
OB_2   LD     A,#$6D        ; (6) OB_2 ---------> CHANGE HERE
       LD     ([HIGHAD.w],X),A   ; Store OB_2
       CALLR  flashp
       BRES   EEXCSR,#2     ; OPT=0
       RET

;*********************************************************************
;    FLASH LOADED PROGRAM
;      ====================
;*********************************************************************
.eep_prg
       LD     A,#$0F        ; \ Program FLASH 16 by 16
       LD     COUNT,A       ; /
start  BSET    EEXCSR,#1    ; Write data latches E2LAT=1
       BRES   EEXCSR,#0     ; Program data latches PGM=0
n_data CALLR  read_data     ; Program data
```

```
       LD    ([HIGHAD.w],X),A   ; Write to FLASH the latched data
       CP    X,COUNT       ; 16 bytes stored ?
       JRNE  suit0         ; If not -> go to suit0
       JRA   suit1         ; Else  -> go to suit1
suit0  INC   X             ; Go to next address
       JRNE  n_data        ; and program it
suit1  INC   X             ; \
       CALLR flashp        ; | Program the 16latched datas in FLASH
       LD    A,COUNT       ; | Re-init COUNT
       ADD   A,#$10        ; | and go to the next address
       LD    COUNT,A       ; /
       CP    X,#00         ; Verify if 256 bytes send
       JRNE  start         ; If not, continue (jump to start)
       LD    A,#$0F        ; \
       LD    COUNT,A       ; | If yes, re-init COUNT
       INC   HIGHAD        ; | Increment High part of address
       LD    A,HIGHAD      ; | Verify if address FFFFh was prog
       CP    A,#00         ; | and if yes ---> exit
       JREQ  end_eep       ; /
       JRA   start         ;
end_eepRET

;*********************************************************************
;    FLASH 16 DATAS WRITE PROGRAM
;      ============================
;*********************************************************************

.flashp
       BSET  EEXCSR,#0     ; Program data latches PGM=1
waitprg
       BTJT  EEXCSR,#1,waitprg   ; Wait end of prog E2LAT =0
       BRES  EEXCSR,#0       ; Program data latches PGM=0
       RET

;*********************************************************************
;    RECEIVE 1 BYTE PROGRAM
;      ======================
;*********************************************************************

.read_data
       BRES  PBDR,#6       ; ISPCLOCK = 0
       CLR   A             ;
       LD    Y,#$08        ; Will receive 8 bits
n_bit                      ;
       BSET  PBDR,#6       ; ISPCLOCK = 1
```

```
       BTJF  PBDR,#5,read ; read a bit of ISPDATA
read   RLC   A            ; Put read bit in LSB of A
       BRES  PBDR,#6      ; ISPCLOCK = 0
       DEC   Y            ; Decrement counter
       JREQ  fin          ; If 8 bits receive -> exit
       JRA   n_bit        ; If not, continue
fin    RET


    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."

littlediode
COMPONENTS FOR ALL

Littlediode supplies new, hard to find or obsolete electronic components and semiconductors all over the world.

With over two million different components listed you are sure to find the part you need.

Feel free to visit us today at our online store:

LittleDiode.com

Looking forward to providing you with the best possible service.