APPLICATION NOTE

MULTIPLE INTERRUPT SOURCES MANAGEMENT FOR ST7 MCUS

by Microcontroller Division Application Team

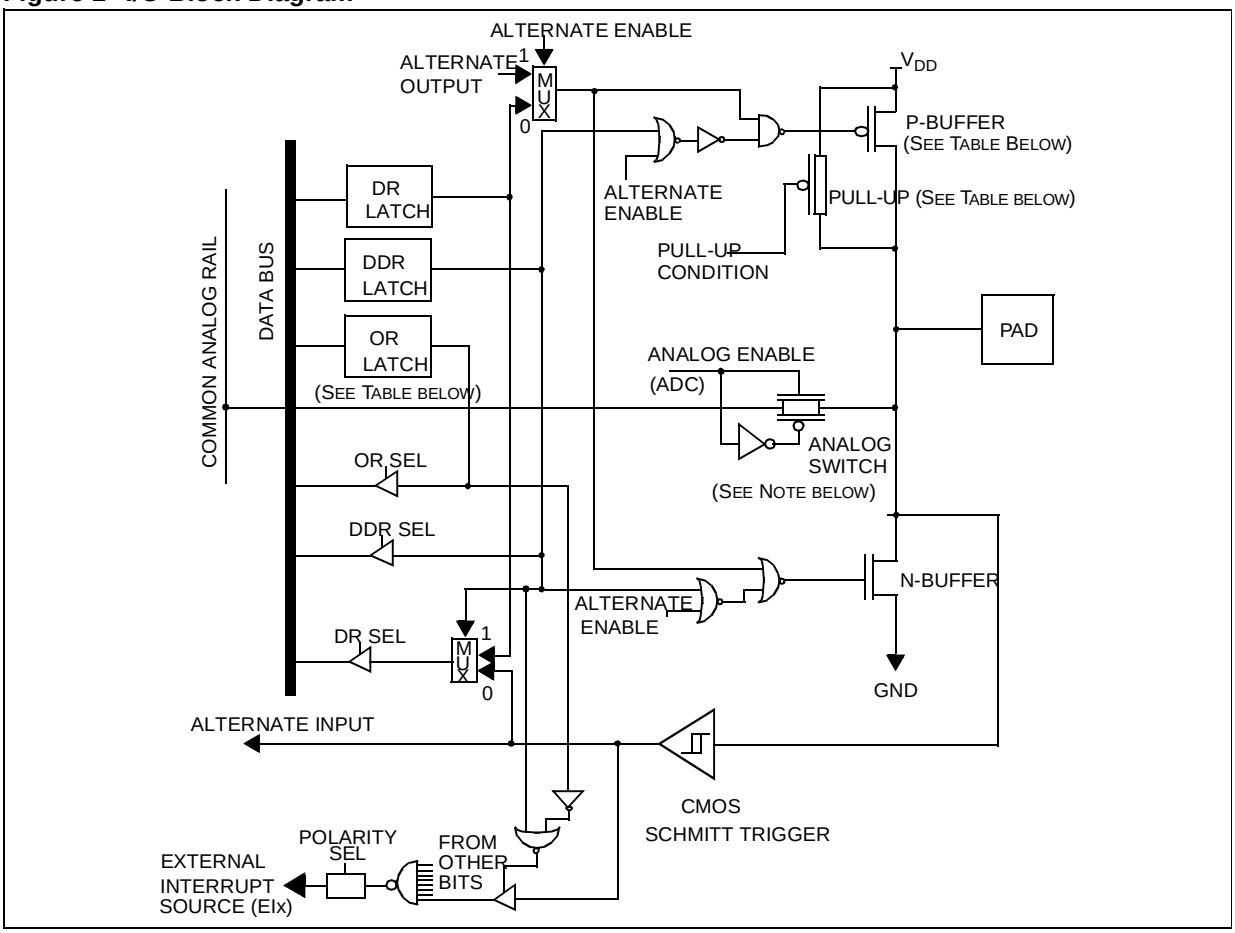
INTRODUCTION

The goal of this application note is to present a technique for managing several external I/O interrupts with a member of the ST7 series of MCUs (here a ST72251).

1 I/O CELL STRUCTURE

Each pin of the microcontroller I/O port can be programmed independently as digital input (with or without interrupt generation) or digital output. Some have also alternate function (SPI, SCI, Timers...). If you want to change the I/O port configuration, take care to respect the state transition diagram (please, refer to the datasheet for more details).

Figure 1. I/O Block Diagram



2 INTERRUPT GENERATION

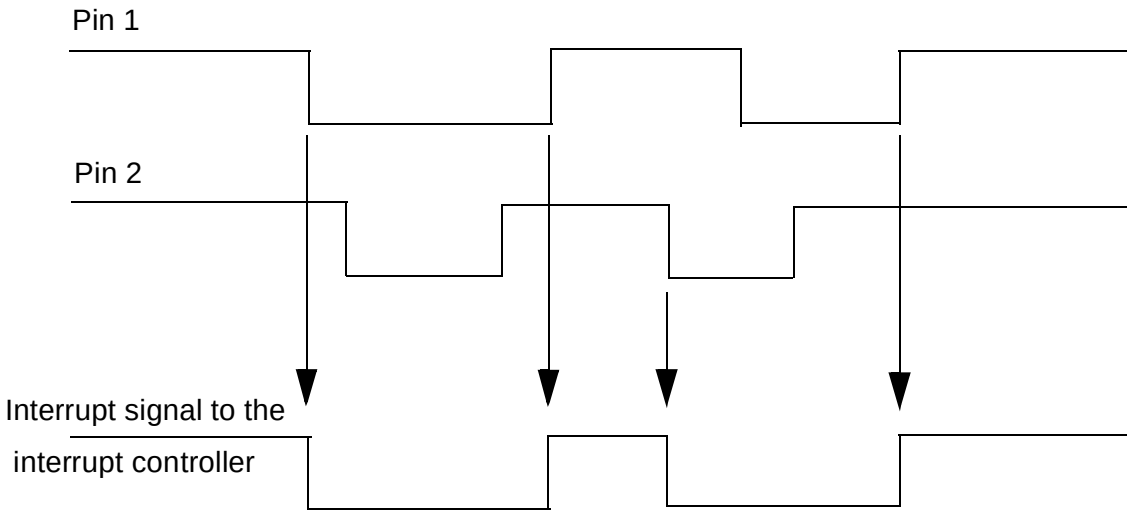
When you use several pins as interrupt pull up inputs, problems may occur if more than one input generates an interrupt within the same period of time. Some will be “masked”, and then won’t be taken into account.

All pins configured as external interrupt mode are connected together in a logical “NAND” function to the same core interrupt vector (see figure 1). An external interrupt is an edge and/ or a low level applied on a pin configured as an input with pull up and interrupt.

To get a high level at the interrupt line, all interrupt pins must be high.

The sensitivity chosen for interrupts in our application is falling edge (please, refer to the miscellaneous register).

Let’s take a look on the possible problem:



As we can see, some interrupts may be inhibited by others if the software doesn’t prevent such events. It’s because of the logical NAND performed in the chip that the interrupt vector won’t be able to treat a nested interrupt.

This application note presents then a solution in order to detect and to treat all falling edges of potential interrupt pins.

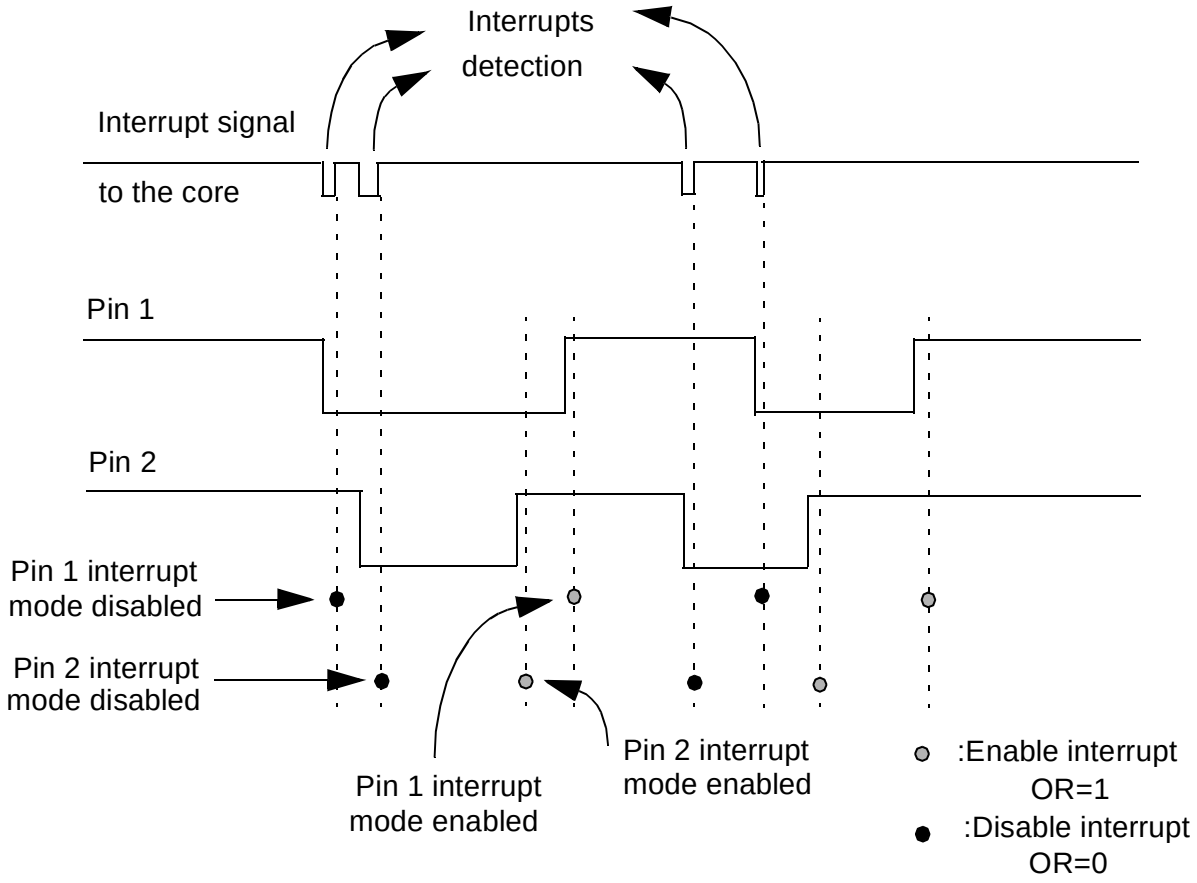
3 DETECTING INTERRUPTS

The interrupt’s edge or level sensitivity has to be programmed by software using the miscellaneous register. In this application note, we then chose “falling edge”.

We use the Port C (PC0 to PC4) as input with pull-up and interrupt (external interrupt 1). To configure it correctly, please refer to the datasheet. We use PA0 as output to apply on Port C pins a low level to create an interrupt.

The main point of the technique used here consists on enabling and disabling inputs which are potential sources of interrupt. Once an interrupt is detected (its falling edge), it's automatically treated (the main program goes to the interrupt subroutine) and then disabled. If another interrupt occurs, all pins are tested (if the OR register bit is set, the interrupt is enabled) in order to see a falling edge which hasn't been treated yet. Once a high level is detected (by polling) on a pin which previously generated an interrupt, the interrupt mode is enabled once again (setting OR bit).

You can have a different interrupt subroutine depending on the pin responsible of the interrupt.

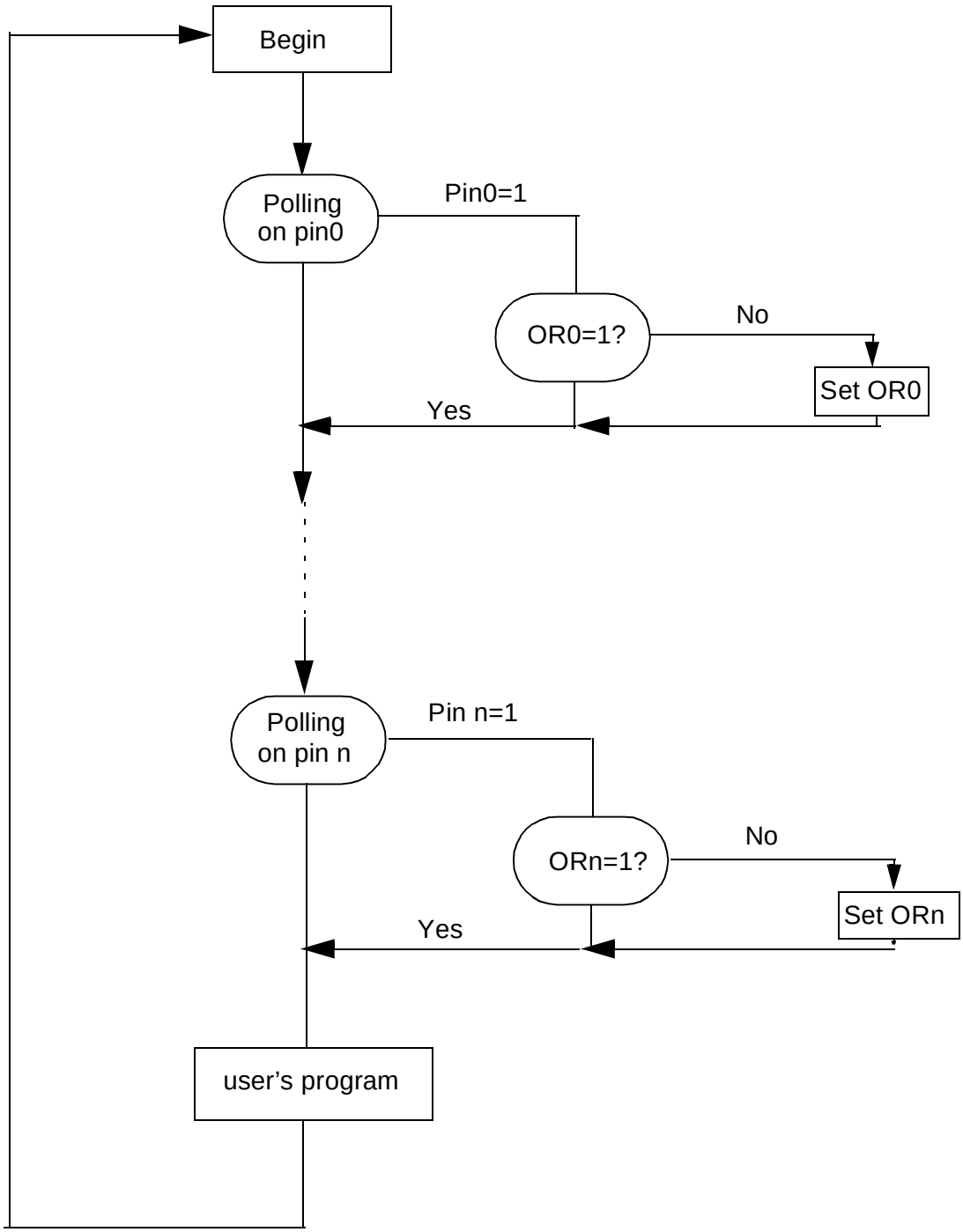


When using this technique, the user is able to detect every single falling edge interrupt with no risk of overlapping between them.

The interrupt subroutine must test every pin and make the right relative jump to the interrupt procedure according to the pin which generated the interrupt.

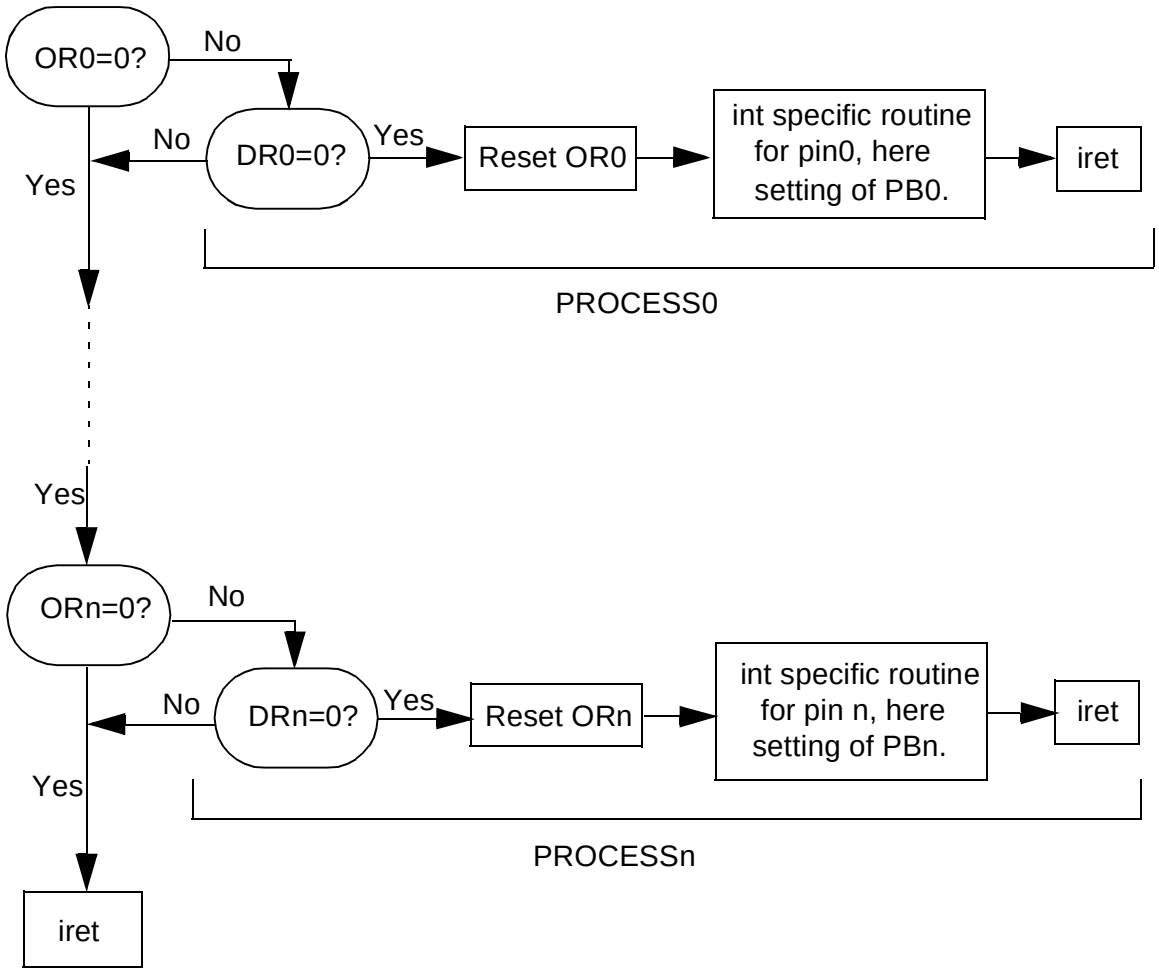
4 FLOWCHARTS

Figure 1. Polling subroutine



n is the number of pins able to generate an interrupt.

Figure 2. interrupt subroutine



n still represents the number of pins able to generate an interrupt.

In the main program, port C pins are tested to be enabled and the LED corresponding to the last pin which caused the interrupt toggles until another interrupt occurs.

5 SOFTWARE

The assembly code given below is guidance only. The complete software with all the files can be found in the software library.

```
st7/                ; the first line is reserved
                    ; for specifying the instruction set
                    ; of the target processor
;*******************************************************************
; TITLE:            SOFTIT.ASM
; AUTHOR:           PPG Microcontroller Applications Team
; DESCRIPTION:      Main program (use of the Halt mode for decoding
;                   a keypad).
;
;*******************************************************************
      TITLE  “SOFTIT.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
;*******************************************************************

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

  segment ‘rom’
;*******************************************************************
;     Initializations routines
;*******************************************************************

.Init  LD   A,#80
       LD   MISCR ,A              ; falling edge sensitive , normal mode.

       LD    A,#$1F
       LD    PCOR,A               ; Port C defined as input with pull up
       CLR   A
       LD    PCDDR,A              ; and interrupt (PC0 to PC4).
```

```
       LD   A,#$3F
       LD   PBDDR,A             ; PortB defined as output push-pull
       LD   PBOR,A              ; (to make corresponding LEDS toggle).

       BSET  PADDR,#0
       BRES  PAOR,#0            ; PA0 defined as output (source of the it).

       RET
;******************************************************************************
;    Tempo routine
;******************************************************************************

.delay                          ; waiting loop of 45ms for fcpu=8MHz.
       LD   Y,#200

loop2  LD   X,#$FF
loop1  DEC   X
       JRNE  loop1
       DEC  Y
       JRNE loop2

       RET

;******************************************************************************
;    Macro routine to enable interrupts on the considered pin.
;******************************************************************************

enable_it    MACRO  num,label

       btjt  PCOR,#num,label    ;if PCnum is in it mode, jump to label
       bset  PCOR,#num          ;else end of it and PCnum interrupt enable.
       jra   label

       MEND
;*************************************************************
;    Program code
;*************************************************************
.main
       CALL  Init
       RIM                      ; Enable interrupts.
loop
       bset  PADR,#0

       btjt  PCDR,#0,enb0       ; Enable PCx if previous IT is over.
.go0   btjt  PCDR,#1,enb1
```

```
.go1   btjt  PCDR,#2,enb2
.go2   btjt  PCDR,#3,enb3
.go3   btjt  PCDR,#4,enb4

.go4   LD   A,PBDR
       CLR  PBDR
       CALL delay
       LD   PBDR,A              ; Toggling of PBx(LEDx), responsible of the IT.
       CALL delay
       CLR  PBDR
       CALL delay
       LD   PBDR,A
       CALL delay
       BRES PADR,#0             ; falling edge and low level (PA0=0).
       CALL delay
       JRA   loop               ; Infinite loop, wait an interrupt occurs.

.enb0
    enable_it    0,go0

.enb1
    enable_it    1,go1

.enb2
    enable_it    2,go2

.enb3
    enable_it    3,go3

.enb4
    enable_it    4,go4

.enb5
    enable_it    5,go5

; ****************************************************************
; 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  jp main            ; Empty subroutine. Go back to main (iret instruction)

.ext0_rt iret

.ext1_rt
      btjt PCOR,#0,it0     ; if OR=0, pin disabled (floating input).
p1    btjt PCOR,#1,it1     ; if OR=1 -> interrupt subroutine.
p2    btjt PCOR,#2,it2
p3    btjt PCOR,#3,it3
p4    btjt PCOR,#4,it4

.return iret


.it0
      btjt PCDR,#0,p1      ; if no low level on the pin -> no IT.
      bres PCOR,#0         ; if IT -> disable the pin.
      jra  process0        ; IT process.
.it1
      btjt PCDR,#1,p2
      bres PCOR,#1
      jra  process1

.it2
      btjt PCDR,#2,p3
      bres PCOR,#2
      jra  process2

.it3
      btjt PCDR,#3,p4
      bres PCOR,#3
      jra  process3

.it4
      btjt PCDR,#4,return
      bres PCOR,#4
      jra  process4

.process0
      clr  PBDR            ; clear LEDS.
```

```
       bset PBDR,#0  ; Set the LED corresponding to the pin responsible of
       iret          ; the IT.

.process1
       clr  PBDR
       bset  PBDR,#1
       iret

.process2
       clr  PBDR
       bset  PBDR,#2
       iret

.process3
       clr  PBDR
       bset  PBDR,#3
       iret

.process4
       clr  PBDR
       bset  PBDR,#4
       iret

.spi_rt iret

.tima_rt iret

.timb_rt iret

.i2c_rt 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 ST72251 based application.
; *********************************************************************


; *********************************************************************
```

```
; Each interrupt vector uses two addresses in rom, that's what the directive
; DC.W means. It says "reserve a word location (.W) in rom (DC) and code
; the routine's label in those two addresses.
; Yet, when an interrupt occurs, for example from the timerB, timerb's routine
; address (timb_rt) will be loaded in the PC and the program will jump to this
; label if allowed. It will execute this routine and then will go back to the main
; program (see interrupt chapter in the datasheet for a more precise description
; of how to handle interrupts in ST72 micros).
; *************************************************************

                    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
      END           ; Be aware of the fact that the END directive should not
                    ; stand on the left of the page like the labels's names.
```

SOFTWARE

"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."