P123: HW D6: Reaction Timer done with micro

REV 21; April 27, 2015.

Total points: 16 points. Due Monday, May 4, 2015

Contents

<table>
<thead>
<tr>
<th>1</th>
<th>Reaction Timer (micro) (16 points)</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Hardware (4 points)</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Code (12 points)</td>
<td>2</td>
</tr>
</tbody>
</table>

1 Reaction Timer (micro) (16 points)

1.1 Hardware (4 points)

No, this isn’t a typo—this is a different way to make a reaction timer.

Assume you start with your lab computer.

Here are some specifications for the circuit:

- two double-throw pushbuttons (SPDT), A and B, start and stop the timing;
- 1 ms resolution is sufficient, for your timer—you needn’t take care of that; timer rollover rate, you may assume, has been adjusted for this result
- debounce only where necessary;
- A and B drive two edge-sensitive interrupt inputs to the 8051, INT0 and INT1. Let the first start the timing, the second stop it.
- let a third pushbutton, C, reinitialize the setup by driving a bit that the processor can sample at its built-in port pin, P3.1. When this button is pushed, the processor clears the display and awaits interrupts.
- let the built-in port pin P3.0 drive a red warning LED (it wants 1.5mA @2V). You decide whether to drive the LED with a High or a Low (only one of the two works!);
- the “reaction time” is to be displayed on the 16-bit display of the LCD board [at ports 0 and 1 (LSD at port 0], using the external bus), in decimal form. The displays include a latch, as you probably know.

Please draw the necessary hardware. Don’t feel obliged to detail the standard elements, like clock and reset and so on for the 8051.

---

Revisions: edit code problem, providing most of program (4/15); reword code problem, trying to make more explicit what is asked, and noting that displaying is done in MAIN (4/12); replace ET2 (interrupt control) with TR2 (timer enable); change switches to SPDT, to ease debouncing; note that displays can be assumed to include latch, and that 1ms resolution is assumed done for the student, in timer rollover-rate setup (4/11—after handout); and a few other odds and ends.
1.2 Code (12 points)

The problem is written with port assignments that fit the Dallas (big board) computer. The SiLabs machine could do the job in very similar form. We don’t want you to write for that version.

Here’s what we’d like your code to do:

• assume you have an internal timer working for you on the 8051, already properly initialized so that it will generate an interrupt every millisecond, if this interrupt is ENABLED.

We would like you to write . . .

• a MAIN program that does just a little:
  – it takes care of initializations (we have written some of them in the template file: we have initialized timer and interrupts for you)
  – SHOWIT: this subroutine puts out the current COUNT value on the display at external-bus ports 0 and 1; LSD on port 0
  – it checks for the “C” pushbutton. When “C” is pressed, MAIN re-initializes

  – a COUNTUP Subroutine: This subroutine increments a pair of timekeeping registers, the registers that the MAIN program continually displays. COUNTUP is called each time the 8051’s hardware timer overflows, generating a TIMER2 INTERRUPT; this happens automatically (we have written this into the template file). You need do nothing to make this happen.

  COUNTUP does several tasks:
  * it increment a 16-bit register pair, in two locations, named COUNT_LO and COUNT_HI, counting in decimal form. You’ll find a programming trick for multi-byte incrementing, in SiLabs Lab C4, §22L.3.4, p.10 (posted).
  * In addition to incrementing the count, the COUNTUP routine should watch for an overflow: if the COUNT overflows, turn on a warning LED at P3.0 and zero the count display. That LED stays lit till the whole program is re-initialized by the “C” pushbutton. Note that the instruction “INC” affects no flags, so it won’t serve to check for overflow;

  – Start and Stop ISR’s
    * the ISR for INT0 should start the internal timer by executing the following command: SETB TR2
    * the ISR for INT1 should disable this timer by executing the following command: CLR TR2

On the next page is the beginning of such a program. We’ve written the timer initializations, so that you can simulate your program with RIDE, if you want. BUT we find that RIDE7—the current version of the Raisonance IDE—omits the timer from its simulation. To simulate the timer you will need to install the older version of RIDE, posted as kit51_747 or kit51_761 on the course website. If you do so, in the debugger remember to add TIMER2 to the windows you WATCH. We’ll post the template file, react_time_MT_apr15.a51, on the course site.

Incidentally, we have set up the timer to run very fast (very small set of counts before overflow), so that you can single-step the program and see how it behaves.
; react_time_MT_apr15.a51 program to display time between two interrupt events ('reaction timer') 4/07
; internal timer sets pace for incrementing of a register (R0); INT0 starts permitting this incrementing,
; INT1 stops the incrementing. Program then displays the resulting value (measuring time between the two interrupts
; insert timer flag clear (4/07)

$NOSYMBOLS ; keeps listing short
$INCLUDE (C:\MICRO\8051\RAISON\INC\REG320.INC) ; Raison’s DS310 register defs. file
$INCLUDE (C:\MICRO\8051\RAISON\INC\VECTORS320.INC) ; Tom’s vectors definition file
STACKBOT EQU 07Fh ; put stack at start of scratch indirectly-addressable block (80h and up)
HITIME SET 0FFh ; set start count close to the overflow val, so it’s not too dull to watch
LOTIME SET 0E0h ; ...and this is the low byte
COUNT_LO EQU 30h ; two storage locations, for 16-bit output
COUNT_HI EQU 31h ; This is the low byte of the 16-bit count
OVER_LED EQU P3.0 ; overflow warning LED
REINIT_BUTTON EQU P3.1

ORG 0h
LJMP STARTUP

ORG 4C0h
STARTUP: MOV SP, #STACKBOT
MOV DPTR, #8000h ; point to displays
    SETB OVER_LED ; turn off overflow LED, initially
ACALL CLEAR_COUNT
    ACALL TIMERINIT ; this initializes timer appropriately, but doesn’t enable its interrupts
    ACALL INTERRUPT_INITS

; -------MAIN program------
TIGHT_LOOP: JNB REINIT_BUTTON, REINIT ; check pushbutton: if pressed, go reinitialize
    ACALL SHOWIT ; display the current count
SJMP TIGHT_LOOP
REINIT: ACALL CLEAR_COUNT ; land here if C button is pushed
    SJMP TIGHT_LOOP

; --SUBROUTINES---
COUNTUP:

SHOWIT:

CLEAR_COUNT: ; clear count, on startup and on overflow:

;------------------
ISRO: This is response to INT ZERO:
ORG INTOVECTOR ; this is defined in VECTORS3210.INC, included above.
; It is address 03h, the address to which micro hops
; in response to interrupt ZERO
ISRO:

;------------------
ISRL: This is response to INT ONE:
ORG INTOVECTOR ; this is defined in VECTORS3210.INC, included above.
; It is address 13h, the address to which micro hops
; in response to interrupt ONE
ISRL:
; ISR TIMER 2: response to Timer2 overflow
ORG TIMER2VECTOR
ISRTIMR:

;----------------------------------------
TIMERINIT:
MOV RCAP2H, #HITIME ; Init timer registers (tiny count:
MOV RCAP2L, #LOTIME ; timer counts UP from initial value
MOV TH2, #HITIME ; this is for quick initial cycle (won’t start from zero)
MOV TL2, #LOTIME
RET

; ----NOW ENABLE INTERRUPTS----
INTERRUPT_INITS: SETB IT0 ; make INT0 Edge-sensitive (p. 22)
SETB IT1 ; ditto for INT1
SETB EX0 ; ...and enable INT0
SETB EX1 ; ...and INT1
    SETB ET2 ; permit timer to interrupt on rollover
SETB EA ; Global int enable (pp.31-32)
RET

END