Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών ΕΜΠ 1

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΑΚΑΔ. ΕΤΟΣ 2024-2025

2ή εργαστηριακή άσκηση.

Ομάδα : 58
Ονοματεπώνυμο : Φέζος Κωνσταντίνος / Α.Μ : 03118076
Ονοματεπώνυμο : Αρθούρος Ρήγας Τσουκνίδας Οδυσσέας / Α.Μ : 03120043

Ζήτημα 2.1

Στο συγκεκριμένο ζήτημα καλούμαστε να υλοποιήσουμε μία ρουτίνα εξυπηρέτησης διακοπών η οποία μετράει των αριθμό των διακοπών INT1(PD3) που δίνονται μέσω της πλακέτας NTUAboard_G1, ενώ παράλληλα τρέχει ένας ατέρμονος μετρητής 0-16 στην PORTB της πλακέτας. Επιπροσθέτως όταν το PD5 είναι πατημένο η μέτρηση των διακοπών σταματάει. Καθώς ορισμένες φορές εμφανίζεται το φαινόμενο του σπινθηρισμού χρησιμοποιήσαμε τα INTF1 και INTF0 ψηφία για να ελέγχουμε αν έχουν "περάσει" παραπάνω διακοπές από αυτές που θα θέλαμε. Η αναλυτική λειτουργία του συγκεκριμένου αλγορίθμου ελέγχου σπινθηρισμού δόθηκε στην εκφώνηση του ζητήματος. Ο κώδικας για την υλοποίηση του συγκεκριμένου ζητήματος σε AVR assembly ακολουθεί μαζί με ορισμένα συνοπτικά σχόλια στα κομβικά σημεία του προγράμματος.

```
.include "m328PBdef.inc"

.def counter=r21
.def temp=r22
.equ FOSC_MHZ = 16
.equ DEL_mS = 1000 ; low number so we don't have to wait for ages in
the simulator
.equ DEL_NU = FOSC_MHZ*DEL_mS

.org 0x000
    rjmp reset
.org 0x004
    rjmp isr1

reset:
     /*reset "memory" */
    clr counter
    /*    Initialise stack    */
    ldi temp, LOW(RAMEND)
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

    /*    PORTD input */
    clr temp
    out DDRD, temp

    /*    PORTC output    */
```

```
    ser temp
    out DDRC, temp


    /*    Setup ISR1 interrupt on high edge   */
    ldi temp, (1 << ISC11 | 1 << ISC10)
    sts EICRA, temp

    /*    Enable ISR1 interrupt    */
    ldi temp, (1 << INT1)
    out EIMSK, temp

    /*    Enable general interrupt flag    */
    sei

main0:
    ser r26
    out DDRB, r26

loop1:
    clr r26
loop2:
    out portb, r26
    ldi r24, low(DEL_NU)
    ldi r25, high(DEL_NU) ; set delay
    rcall delay_routine

    inc r26

    cpi r26,16
    breq loop1
    rjmp loop2
/*  Interrupt handler    */
isr1:
    push r26
    push r25
    push r24
    in r24,SREG
    push r24
recheck:
    ldi r24, (1 << INTF1)
    out EIFR, r24
    rcall delay_routine // delay 100ms
    in r24, EIFR
    sbrc r24,INTF0
```

```
    rjmp recheck
    ldi temp, (1 << INTF1)
    out EIFR, temp    // clear interrupt flag
    cli                    // disable interrupts (temporarily)
    in temp, PIND
    sbrc temp, 05    // if PD5 is pressed skip next line
    inc counter
    out PORTC, counter // turn lights up
    sei // re-enable interrupts
    pop r24
    out SREG, r24
    pop r24
    pop r25
    pop r26
    reti // return from interrupt

/*  Delay routine   */
delay_routine:
    ldi r23, 249
loop_inner:
    dec r23
    nop
    brne loop_inner
    sbiw r24,1
    brne delay_routine
    ret
```

Ζήτημα 2.2

Στο ζήτημα 2.2 επεκτείνουμε τα προηγούμενα για να τροποποιήσουμε τη συμπεριφορά των μετρητών. Τώρα το ζητούμενο είναι η ατέρμονη μέτρηση θα είναι 0-31 και θα αποτυπώνεται στην PORTC. Η μέτρηση θα γίνεται με καθυστέρηση 2 δευτερολέπτων. Στην ίδια θύρα εξόδου θα εμφανίζεται και ο αριθμός των πατημένων μπουτόν PB3 – PB0 όταν εμφανίζεται διακοπή INT0(PD2). Πιο αναλυτικά ο μετρητής σταματάει όταν έρχεται μια διακοπή, στην έξοδο εμφανίζεται, ξεκινώντας από τα LSB, το πλήθος των πατημένων μπουτόν και μετά από μία χρονική καθυστέρηση 2 δευτερολέπτων συνεχίζεται η μέτρηση από εκεί που είχε σταματήσει όταν έλαβε την διακοπή. Παρακάτω ακολουθεί ο πηγαίος κώδικας του προγράμματος σε AVR assembly συνοδευόμενος από τα κατάλληλα σχόλια.

```
.include "m328PBdef.inc"

.equ FOSC_MHZ = 16
.equ DEL_mS = 2000
.equ DEL_NU = FOSC_MHZ*DEL_mS

; Interrupt vector table
.org 0x0
rjmp reset
.org 0x2
rjmp ISR0

reset:
    ; initialize stack pointer
    ldi r24, LOW(RAMEND)
    out SPL, r24
    ldi r24, HIGH(RAMEND)
    out SPH, r24

    ; Init PORTC as output, PORTB as input
    ser r26
    out DDRC, r26
    clr r26
    out DDRB, r26

    ;Interrupts on rising edge of INT0 pin
    ldi r24, (1<< ISC01) | (1<<ISC00)
    sts EICRA, r24

    ; Enable the INT0 interrupt (PD2)
    ldi r24, (1<<INT0)
    out EIMSK, r24
    sei ; enable interrupts


loop1:
    clr r26
loop2:
    andi r26, 0x1F
    out portc, r26
    ldi r24, low(DEL_NU)
    ldi r25, high(DEL_NU) ; set delay
    rcall delay_ms

    inc r26
```

```
    cpi r26,32
    breq loop1
    rjmp loop2



 ISR0:
    push r26
    push r25
    push r24
    in r24, SREG ; save r24, r25, SREG
    push r24 ; push to stack

recheck:
    ldi r24, (1 << INTF0)
    out EIFR, r24
    ldi r24, low(16*5)
    ldi r25, high(16*5)
    rcall delay_ms // delay 5ms
    in r24, EIFR
    sbrc r24,INTF0
    rjmp recheck

    ldi r24, (1 << INTF0)
    out EIFR, r24 // clear interrupt flag
    cli // disable interrupts (temporarily)


    ldi r24, 0 ; in r24 we will produce the output for PORTC
    in r25, PINB
    ldi r16, 4 ; this is a counter in order to see only up to bit PB3

count_in_b_loop:
    mov r28, r25
    andi r28, 0x01 ; keep only LSB
    cpi r28, 0x00 ; if LSB is 0 (pressed) increment r24
    breq to_inc
    rjmp continue

to_inc:
    lsl r24
    ori r24, 0x01 ; push bits left and make the LSB 1

continue:
    lsr r25 ; bring second LSB to LSB
```

```
    dec r16
    brne count_in_b_loop

output_leds:
    out portc, r24

    ldi r24, low(DEL_NU)
    ldi r25, high(DEL_NU)
    rcall delay_ms ; delay




    pop r24
    out SREG, r24
    pop r24
    pop r25
    pop r26 ; pop from stack

    reti


/* delay routine */
delay_ms:
    ldi r23, 249
loop_inner:
    dec r23
    nop
    brne loop_inner

    sbiw r24,1
    brne delay_ms
    ret
```

Ζήτημα 2.3

Τέλος στο ζήτημα 2.3 καλούμαστε να δημιουργήσουμε ένα πρόγραμμα που υλοποιεί τον αυτοματισμό ελέγχου ενός φωτιστικού σώματος. Η διακοπή INT1 ανάβει το LED στο LSB(PB0) της εξόδου PORTB για 5 δευτερόλεπτα. Σε περίπτωση που ληφθεί και άλλη διακοπή όσο το LED είναι πατημένο ανάβουν όλα τα LED της PORTB για μισό δευτερόλεπτο και μετά ξανά το LSB για τα υπολειπόμενα 4.5 δευτερόλεπτα. Στο πρόγραμμα μας σε AVR assembly αντιμετωπίσαμε το φαινόμενο του σπινθηρισμού όπως και στο ζήτημα 2.1. Στην

υλοποίηση σε C εισάγουμε ένα μικρό delay των 10-50 mS προτού εξυπηρετήσουμε την διακοπή για να επιβεβαιώσουμε ότι δεν προκαλείται σπινθηρισμός. Παρακάτω φαίνονται οι πηγαίοι κώδικες :

- Ο κώδικας σε AVR assembly :

```
.include "m328PBdef.inc"
; Interrupt vector table
.org 0x0
rjmp MAIN
.org 0x4
rjmp INT1_ISR
MAIN:
    ; Set up external interrupt INT1
    ldi r16, (1 << ISC11) | (1 << ISC10)  ; Set rising edge detection
on INT1 (EICRA)
    sts EICRA, r16                        ; Enable rising edge on INT1

    ldi r16, (1 << INT1)                  ; Enable INT1 interrupt
(EIMSK)
    out EIMSK, r16

    sei                                   ; Enable global interrupts

    ; Set PORTB as output
    ldi r16, 0xFF                         ; Set DDRB = 0xFF (all
outputs)
    out DDRB, r16
LOOP:
    ; Infinite loop, set PORTB = 0x00
    ldi r17, 0x00                         ; Clear PORTB (PORTB = 0x00)
    out PORTB, r17
    rjmp LOOP                             ; Repeat indefinitely

INT1_ISR:
     sei
RECHECK:
    ldi r26, (1 << INTF1)
    out EIFR, r26
     ldi r24, low(100)
     ldi r25, high(100)
    rcall wait_x_ms                   // delay 100ms
    in r26, EIFR
    sbrc r26,INTF0
```

```
    rjmp RECHECK
    ; Check if PORTB == 0x01 or PORTB == 0xFF
    in r16, PORTB                          ; Load PORTB value into r16
    cpi r16, 0x01                          ; Compare with 0x01
    breq TURN_ON_ALL_LEDS                  ; If equal, go to
TURN_ON_ALL_LEDS
    cpi r16, 0xFF                          ; Compare with 0xFF
    breq TURN_ON_ALL_LEDS                  ; If equal, go to
TURN_ON_ALL_LEDS
TURN_ON_PB0:
    ldi r16, 0x01                          ; Set PB0 (PORTB = 0x01)
    out PORTB, r16
    ldi r24, low(5000)
    ldi r25, high(5000)                    ; Wait for 5 s
    rcall wait_x_ms
    ldi r16, 0x00                          ; Turn off PB0 (PORTB =
0x00)
    out PORTB, r16
    reti                                   ; Return from interrupt

TURN_ON_ALL_LEDS:
    ldi r16, 0xFF                          ; Set all LEDs on PORTB
(PORTB = 0xFF)
    out PORTB, r16
    ldi r24, low(500)
    ldi r25, high(500)
    rcall wait_x_ms                          ;Wait for 0.5 s
    ldi r16, 0x01                           ;Turn on only the
first LED
    out PORTB, r16
    ldi r24, low(4500)
    ldi r25, high(4500)                          ; Wait for 4.50 m\s
    rcall wait_x_ms
    ldi r16, 0x00                          ; Turn off all LEDs (PORTB =
0x00)
    out PORTB, r16
    reti                                   ; Return from interrupt

wait_x_ms:
    ldi r16, 16
extra_outer_delay:
    rcall delay_outer
    subi r16,1
    brne extra_outer_delay
    ret
```

```
;this routine is used to produce a delay 993 cycles
delay_inner:
    ldi r23, 247          ; 1 cycle
loop3:
    dec r23               ; 1 cycle
    nop                   ; 1 cycle
    brne loop3            ; 1 or 2 cycles
    nop                   ; 1 cycle
    ret                   ; 4 cycles
;this routine is used to produce a delay of (1000*X1) cycles
delay_outer:
    push r24              ; (2 cycles)
    push r25              ; (2 cycles) Save r24:r25
loop4:
    rcall delay_inner     ; (3+993)=996 cycles
    sbiw r24,1            ; 2 cycles
    brne loop4            ; 1 or 2 cycles
    pop r25               ; (2 cycles)
    pop r24               ; (2 cycles) Restore r24:r25
    ret                   ; 4 cycles
```

- Ο κώδικας σε C :

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdbool.h>
volatile uint32_t time= 0;

ISR (INT1_vect) {
     time = 0;
     _delay_ms(100);
     sei(); //re-enabling interrupt for nested interrupts to be
allowed
     if(PORTB == 0x01 || PORTB == 0x3F) {
          PORTB = 0x3F;
          while(time<500) {
               _delay_ms(1);
               time++;
          }
          PORTB = 0x01;
          while (time< 5000) {
               _delay_ms(1);
```

```
                time++;
            }
            PORTB = 0x00;
        }
        else {
            PORTB = 0x01;
            while (time < 5000) {
                _delay_ms(1);
                time++;
            }
            PORTB = 0x00;
        }
        EIFR = (1<<INTF1); //do i need this?
}

int main() {
        // interrupt on rising edge of INT1
        EICRA = (1<<ISC11) | (1<<ISC10);
        //Enable the INT1 (PD3)
        EIMSK = (1<<INT1);
        sei(); //enable interrupts
        DDRB = 0xFF; //PortB as output

        while(1) {
            PORTB = 0x00;
        }
        return 0;
}
```