2η Εργαστηριακή Αναφορά

Παπαδόπουλος Χαράλαμπος 03120199

Στρίφτης Γεώργιος 03121200

Άσκηση 1

Σκοπός της άσκησης ήταν να προστατεύσουμε τις ρουτίνες εξυπηρέτησης διακοπών από το φαινόμενο του σπινθηρισμού καθώς και να υλοποιήσουμε μια μέθοδο απενεργοποίησης των διακοπών με push-button.

Παρατίθεται το βασικό κομμάτι του κώδικα, παραλείποντας τις αρχικοποιήσεις και την υλοποίηση της καθυστέρησης για εξοικονόμηση χώρου:

```
main:
    in r17, PIND
    sbrs r17, 5                 ; Check if PD5 is pressed
    rjmp disable_int            ; If yes, diable interrupts
    rcall enable_int

continue:
    out PORTB, r20

    rcall long_delay            ; 500ms

    inc r20

    sbrc r20, 4
    clr r20
    rjmp main



ISR1:
    push r24
    push r25
    in r24, SREG
    push r24


    ldi r17, (1<<INTF1)         ; spark gap protection
    out EIFR, r17

    rcall short_delay
```

```
    in r17, EIFR
    cpi r17, 0b00000010
    brne skip
    pop r24
    out SREG, r24
    pop r25
    pop r24
    rjmp ISR1

    skip:
    inc int_counter
    SBRC int_counter, 6                    ; Check for overflow
    clr int_counter
    out PORTC, int_counter

    pop r24
    out SREG, r24
    pop r25
    pop r24

    reti

 disable_int:
    cli
    ldi r27, (0<<INT1)
    out EIMSK, r27

    ldi r17, (1<<INTF1)
    out EIFR, r17

    rjmp continue

enable_int:
    ldi r17, (1<<ISC11) | (1<<ISC10)
    sts EICRA, r17

    ldi r17, (1<<INT1)
    out EIMSK, r17

    ldi r17, (0<<INTF1)
    out EIFR, r17

    sei

    ret
```

Όσον αφορά το κομμάτι της απενεργοποίησης των διακοπών μέσω του push-button PD5, ελέγχουμε αν το PD5 είναι πατημένο στην αρχή της main μέσω της εντολής sbrs (ελέγχουμε το

set λόγω αντίστροφης λογικής στην πλακέτα). Όσο είναι πατημένο, θα γίνεται ξανά το loop της main και μόνο όταν αφεθεί θα κληθεί η enable_int, η οποία ενεργοποιεί ξανά τις διακοπές και συνεχίζει την μέτρηση από εκεί που είχε μείνει.

Άσκηση 2

Σκοπός αυτής της άσκησης ήταν μέσω της διακοπής να ελέγξουμε κάποια δεύτερη εξωτερική είσοδο (PINB).

```
loop1:
   clr r26

loop2:
   out PORTC, r26

   ldi r24, low(DEL_NU)
   ldi r25, high(DEL_NU)          ; Set delay (number of cycles)
   rcall delay_mS

   inc r26

cpi r26, 32                  ; compare r26 with 16
breq loop1
rjmp loop2



ISR0:
   push r24
   push r25
   in r24, SREG
   push r24
   push r26

   in r17, PINB
   com r17
   andi r17, 0b00001111

   ldi r18, 0b00001111                ; auxillary register
   sbrc r17, 0                  ; whenever we find a pressed button we rotate the aux-reg
   lsl r18                      ; causing the overflow to bring one extra bit upfront
   sbrc r17, 1
   lsl r18
   sbrc r17, 2
   lsl r18
```

```
    sbrc r17, 3
    lsl r18

    com r18
    andi r18,0b00001111
    out PORTC, r18

    pop r26
    pop r24
    out SREG, r24
    pop r24
    pop r25

    reti
```

Για την επίτευξη του στόχου μας αξιοποιούμε έναν επιπλέον καταχωρητή (r18) ο οποίος έχει τα τέσσερα τελευταία bit του «αναμμένα». Έτσι, διαβάζοντας το PINB όποτε βρίσκουμε ένα PIN πατημένο, κάνουμε μια περιστροφή αριστερά (δεξιά στην πλακέτα) ανάβοντας ένα led κάθε φορά.

Άσκηση 3

Σκοπός της άσκησης είναι να υλοποιήσουμε αυτοματισμό για να ανάβουμε και σβήνουμε ένα φωτιστικό σώμα. Παρακάτω φαίνεται ο κώδικας σε assembly:

```
setup:
    clr r20
    ldi r17, LOW(RAMEND)
    out SPL, r17
    ldi r17, HIGH(RAMEND)
    out SPL, r17

    .equ delay = 5000                    ;ms
    .equ freq = 16
    .DEF counter = r17
    ldi r24, low(delay)
    ldi r25, high(delay)


    .def status = r28
    clr status

    ser r26
    out DDRB, r26
```

```
    ldi r17, (1<<ISC11) | (1<<ISC10)
    sts EICRA, r17

    ldi r17, (1<<INT1)
    out EIMSK, r17

    sei

main:
    clr r19
    out PORTB, r19
    cpi status, 0xFF
    breq lights_on
    cpi status, 0x0F
    breq refresh
    rjmp main

lights_on:
    ldi r19, 1
    out PORTB, r19
    ldi r24, low(5000)
    ldi r25, high(5000)
    rcall wait_delay_ms
    cpi status, 0x0F
    breq refresh
    clr r19
    out PORTB, r19
    clr status
    rjmp main

refresh:
    ldi r19, 0b00001111
    out PORTB, r19
    ldi r24, low(500)
    ldi r25, high(500)
    rcall wait_delay_ms
    ser status
    rjmp lights_on

ISR1:
    push r17
    push r22
    in r22, SREG
    push r22


spark:
    ldi r17, (1<<INTF1)
    out EIFR, r17
```

```
    rcall short_delay

    in r17, EIFR
    cpi r17, 0b00000010
    brne skip
    rjmp spark
skip:
    cpi status, 0
    brne refresh_int
    ser status
    rjmp int_end


    refresh_int:
    ldi r24, 1
    clr r25
    ldi status, 0x0F

    int_end:
    pop r22
    out SREG, r22
    pop r22
    pop r17

    reti


short_delay:
    ldi r24,low(5)
    ldi r25, high(5)
    rcall wait_delay_ms
    ret



wait_delay_ms:
    ldi counter, freq       ; 16MHz so 16 loops
    rcall delay_1ms         ; 3 + 15.994 cycles
    sbiw r24, 1             ; 1 cycle
    brne wait_delay_ms              ; 1 or 2 cycles
    ret                     ; 4 cycles

; 16.000 cycles ~ 1ms
delay_1ms:
    rcall delay_inner       ; 3 + 993 cycles
    dec counter             ; 1 cycle
    brne delay_1ms          ; 1 or 2 cycles
    ret                     ; 4 cycles

; 993 cycles ~ 1/16 ms
```

```
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
```

Χρησιμοποιούμε έναν καταχωρητή status για να ξεχωρίσουμε τρεις καταστάσεις:

1. **status** = 0x00: Φώτα σβηστά
2. **status** = 0xff: Φώτα αναμμένα (PB0)
3. **status** = 0x0f: Πρέπει να γίνει refresh

Στην αρχή της ρουτίνα διακοπής ελέγχουμε για σπινθηρισμό και έπειτα κάνουμε το εξής:

- Αν **status** = 0x00 τότε τον θέτουμε ίσο με 0xff και ανάβουμε τα φώτα για 5 sec.
- Αν **status** = 0xff τότε τον θέτουμε 0x0f, μηδενίζουμε τους καταχωρητές r24, r25 που χρησιμοποιεί η ρουτίνα ***wait_delay_ms*** έτσι ώστε να τερματίσει και μετά ανάβουμε όλα τα φώτα για 0.5 sec.

Παρόμοια λογική έχουμε και στον κώδικα σε C:

```
#define F_CPU 16000000UL
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>


volatile int check = 0;
volatile int running = 0;
volatile int count = 0;

ISR(INT1_vect)            // External INT1 ISR
{   if(running) {
          PORTB=0X0F;
          _delay_ms(500);
          count = 0;
}
check = 1;
EIFR = (1 << INTF1);    // Clear the  flag of interrupt INTF1
}

int main(){
          // Interrupt on rising edge of INT1 pin
```

```
        EICRA=(1<<ISC11) | (1<<ISC10);

        // Enable the INT1 interrupt (PD3))
        EIMSK= (1<<INT1);

        sei();                  // Enable global interrupts

        DDRB=0xFF;              // Set PORTB as output

        PORTB=0x00;             // Turn off all LEDs of PORTB

        while(1)
        {
                while(check) {
                        while(count < 5000) {
                                if(count==100) running = 1;
                                PORTB = 0x01;
                                _delay_ms(1);
                                count++;
                        }

                        count = 0;
                        PORTB=0x00;
                        running = 0;
                        check = 0;
                }
        }
}
```

Για το delay χρησιμοποιούμε την ***_delay_ms()*** που μας παρέχεται από την βιβλιοθήκη ***util/delay.h***. Εκτελούμε 5000 loops και την καλούμε κάθε φορά για 1 ms.

Το **check** τίθεται ίσο με 1 εάν είναι αναμμένα τα φώτα και 0 εάν όχι.

Το **running** τίθεται εάν είναι αναμμένα μετά από κάποιο χρόνο τα φώτα (100ms στη συγκεκριμένη περίπτωση) για να αποφύγουμε φαινόμενα σπινθηρισμού.

Έτσι, την πρώτη φορά που καλείται το interrupt θέτει το **check** = 1 και μετά εάν κληθεί πάλι και **running** = 1 τότε ανάβει όλα τα φώτα για 0,5 sec.