2η Εργαστηριακή Άσκηση
Εργαστήριο Μικροϋπολογιστών

Ανδρέας Στάμος (03120018), Κωνσταντίνος Πίκουλας (03120112)

Οκτώβριος 2023

Περιεχόμενα

1 Ζήτημα 2.1 1

2 Ζήτημα 2.2 3

3 Ζήτημα 2.3 5

1 Ζήτημα 2.1

Οι διακοπές ρυθμίστηκαν στην ανερχόμενη ακμή (την στιγμή δηλαδή που αφήνουμε το κουμπί). Στην ρουτίνα εξυπηρέτησης διακοπής γίνεται ελέγχος αν το PD6 είναι πατημένο, και αν είναι απλά επιστρέφει χωρίς να κάνει κάτι. Προκειμένου να αποφύγουμε την πρόκληση πολλαπλών διακοπών λόγω σπινθηρισμών, κάναμε debounce την ρουτίνα εξυπηρέτησης διακοπών, εκτελώντας την διακοπή, μόνο αν στα τελευταία 5ms δεν συνέβη νέα διακοπή. Ο μετρητής των διακοπών αποθηκεύεται στον `r19` που χρησιμοποιείται αποκλειστικά από την ρουτίνα εξυπηρέτησης διακοπής, οπότε ενδιάμεσα στις διακοπές δεν απαιτείται να αποθηκεύεται στην μνήμη.

Ο κώδικας Assembly είναι ο παρακάτω:

```
.include "m328PBdef.inc"

.equ FOSC_MHZ=16
.def counter=r19 ; counter for interrupts
.def temp=r20
.equ DEL_ms=500
.equ DEL_NU=FOSC_MHZ*DEL_mS
.equ DEBOUNCE_ms=10 ; ms
.equ DEBOUNCE_NU=FOSC_MHZ*DEBOUNCE_mS

.org 0x0
rjmp reset
.org 0x4
rjmp ISR1

ISR1: ; interrupt handle
        in r29, SREG
        push r23
        push r24
        push r25

fix:
        ;START: for ex 1.b ;
        ldi r18, (1<<INTF1)
        out EIFR, r18 ; set INT0 to zero (logic 1)
        ldi r24, low(DEBOUNCE_NU)
        ldi r25, high(DEBOUNCE_NU)
        rcall delay_ms
        sbic EIFR, INTF1 ; skip if bit INTF1 is 0
```

```
30          rjmp fix
31          ;END: for ex 1.b ;
32
33          sbic PIND, 6 ; skip if PD6 is cleared (we pressed the button)
34          inc counter
35          cpi counter, 0x20 ; compare to 32
36          breq zero_counter
37  return_interrupt:
38          out PORTC, counter
39          pop r25
40          pop r24
41          pop r23
42          out SREG, r29
43          reti
44  zero_counter:
45      clr counter ; clear counter because we got to 32
46      jmp return_interrupt
47
48
49  reset:
50      ldi r24, high(RAMEND)
51      out SPH, r24
52      ldi r25, low(RAMEND)
53      out SPL, r25
54
55      ; init PORTD as input
56      clr temp
57      out DDRD, temp
58      ser temp
59      out PORTD, temp
60      ; init PORTC as output
61
62      out DDRC, temp
63      out DDRB, temp
64
65      ; init interrupts
66      clr temp
67      ldi temp, (1 << ISC11 | 1 << ISC10)
68      sts EICRA, temp
69
70      clr temp
71      ldi temp, (1 << INT1)
72      out EIMSK, temp
73
74      clr counter
75      out PORTC, counter
76      sei ; set global interrupt flag
77
78
79
80  loop1:
81      clr r26
82      loop2:
83      out PORTB, r26
84
85      ldi r24, LOW(DEL_NU)
86      ldi r25, HIGH(DEL_NU)
87
88      rcall delay_mS
89
90      inc r26
```

```
91
92      cpi r26, 16
93      breq loop1
94      rjmp loop2
95
96  delay_mS:
97      ldi r23, 249
98      loop_inn:
99      dec r23
100     nop
101     brne loop_inn
102
103     sbiw r24, 1
104     brne delay_mS
105
106
107     ret
```

2 Ζήτημα 2.2

Οι διακοπές ρυθμίστηκαν στην αρνητική στάθμη τάσης (δηλαδή για όσο πατάμε το κουμπί). Ο λόγος είναι πως επιθυμούμε στην ρουτίνα εξυπηρέτησης διακοπής να μπορούμε να ανιχνεύουμε αν το κουμπί παραμένει πατημένο, προκειμένου να παραμένουμε στην ρουτίνα εξυπηρέτησης διακοπής. Λόγω σπινθηρισμών, αν απλά κάναμε poll το PIND, σύντομα θα το βρίσκαμε λανθασμένα πως δεν είναι πατημένο και θα επιστρέφαμε από την διακοπή. Αντίθεα, με ρυθμισμένη την διακοπή στο αρνητικό επιπέδο, απλά ανιχνεύουμε αν σε ένα διάστημα 5 ms συνέβη διακοπή, μηδενίζοντας στον EIFR την INT0, περιμένοντας 5 ms και έπειτα ελέγχοντας στον EIFR αν συνέβη INT0.

Επειδή οι ενδείξεις της "κανονικής λειτουργίας" (την μέτρηση) και της "λετουργίας της διακοπής" είναι στο ίδιο PORT, μέσα στην διακοπή, αποθηκεύουμε πρώτα τι προϋπήρχε στο PORTC και πριν επιστρέψουμε, το ξαναθέτουμε στο PORTC.

Η ζητούμενη λειτουργία (ενεργοποίηση τόσων LSB LEDS στο PORTC όσα κουμπιά έχουν πατηθεί στα PB0-PB4) υλοποιείται ελέγχοντας ένα-ένα τα bits του PINB (με χρήση της sbis) και αν έχουν τιμή 0, τότε κάνουμε αριστερό shift την τιμή του τρέχοντος αποτελέσματος και bitwise OR με το 0x1 (προσθέτουμε έναν ακόμα LSB άσσο).

Ο κώδικας Assembly είναι ο παρακάτω:

```
1   .include "m328PBdef.inc"
2
3   .equ FOSC_MHZ=16
4   .equ DEL_ms=1000
5   .equ DEL_NU=FOSC_MHZ*DEL_mS
6   .equ DEBOUNCE_ms=5
7   .equ DEBOUNCE_NU=FOSC_MHZ*DEBOUNCE_mS
8
9   .org 0x0
10  rjmp reset
11  .org 0x2
12  rjmp ISR0
13
14  add1:
15      lsl r28
16      ori r28, 0x1
17      ret
18
19      ;r27, r28 are used exclusively by ISR1
20  ISR0:
21      push r24
22      push r25
23      in r27, SREG
24
25      isr0_loop:
26      clr r28
27
```

```
28      sbis PINB, 0
29      rcall add1
30      sbis PINB, 1
31      rcall add1
32      sbis PINB, 2
33      rcall add1
34      sbis PINB, 3
35      rcall add1
36      sbis PINB, 4
37      rcall add1
38
39      ;store what was displayed in portc before
40      in r29, PORTC
41
42      out PORTC, r28
43
44      ;while PD2 is pressed the 'interrupt' function should happen.
45      ldi r30, (1<<INTF0)
46      out EIFR, r30
47      ldi r24, LOW(DEBOUNCE_NU)
48      ldi r25, HIGH(DEBOUNCE_NU)
49      rcall delay_mS
50      sbic EIFR, INTF0
51      rjmp isr0_loop
52
53      ;recover to portc what was displayed before
54      out PORTC, r29
55
56      pop r25
57      pop r24
58      out SREG, r27
59      reti
60
61
62
63  reset:
64  ;Init Stack Pointer
65      ldi r24, LOW(RAMEND)
66      out SPL, r24
67      ldi r25, HIGH(RAMEND)
68      out SPH, r25
69
70  ;Init PORTB,PORTD as input
71      clr r26
72      out DDRB, r26
73      out DDRD, r26
74      ser r26
75      out PORTB, r26
76      out PORTD, r26
77  ;Init PORTC as output
78      out DDRC, r26
79
80      ldi r24, 0 ;negative-edge triggered
81      sts EICRA, r24
82      ldi r24, (1 << INT0)
83      out EIMSK, r24
84
85      sei
86
87  ;other function (according to exercise) unrelated to counting interrupts
88  loop1:
```

```
89      clr r26
90      loop2:
91      out PORTC, r26
92
93      ldi r24, LOW(DEL_NU)
94      ldi r25, HIGH(DEL_NU)
95
96      rcall delay_mS
97
98      inc r26
99
100     cpi r26, 32
101     breq loop1
102     rjmp loop2
103
104 delay_mS:
105     ldi r23, 249
106     loop_inn:
107     dec r23
108     nop
109     brne loop_inn
110
111     sbiw r24, 1
112     brne delay_mS
113
114     ret
```

3 Ζήτημα 2.3

Αρχικά σκεφτήκαμε στις ανανεώσεις να προκαλούμε nested διακοπές που μετρούν τον νέο χρόνο και, έπειτα, μηδενίζοντας τον μετρητή `r25:r24` πρακτικά θα μηδενιζόταν ο χρόνος της γονικής διακοπής, οπότε εκείνη θα τερμάτιζε. Παρατηρήσαμε ότι αυτό δεν συνέβαινε. Ο λόγος που δεν συνέβαινε είναι πως με τον μηδενισμό του μετρήτη, ανάλογα που είχε συμβεί η διακοπή-παιδί στην γονική διακοπή, τύχαινε το εξής (κάπως σαν race condition):

Η διακοπή-παιδί μηδένιζε τον μετρητή `r25:r24`. Στην γονική διακοπή αφαιρούνταν 1 από τον μετρητή και έπειτα γινόταν ο ελέγχος αν ο μετρητής μηδένισε. Επειδή ο μετρητής είχε μηδενίσει με την αφαίρεση, πήγαινε στο `0xFFFF` που σημαίνει ότι είχε ακόμα να περιμένει αντί μηδενικού χρόνου, $\texttt{0xFFFF} \cdot 1000$ κύκλους $\cdot \frac{1}{16\ \text{MHz}} \approx 4$ sec.

Το πρόβλημα το ανιμετωπίσαμε, ανανεώνοντας απλά τον χρόνο που πρέπει να τρέξει η γονική διακοπή αντί να προσπαθήσουμε να την ακυρώσουμε (στην αρχή γράψαμε μια λύση όπου ακυρώναμε την γονική διακοπή αφαιρώντας το frame της από την stack, όμως αυτό γενικά δεν είναι καλή προγραμματιστική ιδέα).

Έτσι αν είμαστε στο στάδιο όπου είναι αναμένο το 1 LED, τα ανάβουμε όλα για 500ms και έπειτα αλλάζουμε τον μετρητή της γονικής σε 2500ms, ενώ αν είμαστε στο στάδιο όπου είναι αναμένα όλα τα LEDs, απλά ανανεώνουμε τον χρόνο των όλων LEds σε 500ms.

Επιπρόσθετα, προκειμένου να αποφύγουμε τυχόντα άλλα race conditions και προκειμένου να εξασφαλίσουμε την ορθότητα του προγράμματος, επιτρέψαμε τις διακοπές μόνο μέσα στην delay. (εξάλλου σύμφωνα με το datasheet του ATMega328PB η διακοπή αποθηκεύεται ότι συνέβη στον `EIFR` με την ρουτίνα εξυπηρέτησης διακοπής να εκτελείται μόλις στον `SREG` το Interrupts flag γίνει ενεργό.)

Στον κώδικα C ακολουθήσαμε παρόμοια λογική, όμως προκειμένου να χρησιμοποιήσουμε την έτοιμη ρουτίνα `_delay_ms`, χωρίσαμε τα χρονικά διαστήματα αναμονής σε διαστήματα του 1 ms και διατηρήσαμε μετρητή για το πόσα διαστήματα του 1 ms πρέπει να γίνουν. Επιπρόσθετα στην κατάσταση που όλα τα LEDs ήταν αναμένα, καταφέραμε να επιτυχούμε τον αρχικό στόχο μας να "ακυρώνουμε" την delay της γονικής διακοπής με τον μηδενισμό του αντίστοιχου μετρητή, επιβάλλοντας την ατομικότητα (απενεργοποιώντας και ενεργοποιώντας τις διακοπές) του τμήματος όπου πρώτα ελέγχουμε αν είναι θετικός ο μετρητής και έπειτα τον μειώνουμε κατά 1.

Ο κώδικας Assembly είναι ο παρακάτω:

```
1  .include "m328PBdef.inc"
2
3
4  .equ FOSC_MHZ=16          ;MHz
5  .equ NORMAL_TIMEOUT=3000  ;ms
6  .equ BLINK_TIMEOUT=500    ;ms
7  .def temp=r18
```

```
.def flag=r17        ; this also represents a flag
                     ; temp = 0XFF means an ongoing timer is active
                     ; temp = 0X00 means no ongoing timer is active
.equ DEBOUNCE_ms=100 ; ms
.equ DEBOUNCE_NU=FOSC_MHZ*DEBOUNCE_mS
.equ NORMAL_CYCLES = NORMAL_TIMEOUT * FOSC_MHZ
.equ BLINKING_CYCLES = BLINK_TIMEOUT * FOSC_MHZ
.def active_interrupt_counter=r30
.org 0x0
rjmp reset
.org 0x4
rjmp ISR1




 reset:
    ldi r24,low(RAMEND)
    out SPL,r24
    ldi r24,high(RAMEND)
    out SPH,r24

    ; init interrupts
    ldi r24, (1 << ISC11) | (1 << ISC10)
    sts EICRA, r24

    ldi r24, (1 << INT1)
    out EIMSK, r24
    sei

    ldi r24, low(NORMAL_CYCLES) ; load timeout
    ldi r25, high(NORMAL_CYCLES)
    ldi r26, low(BLINKING_CYCLES)
    ldi r27, high(BLINKING_CYCLES)
   ;set PORTB as output
   ser temp
   out DDRB, temp
   clr temp
   out PORTB, temp
   clr flag

main:
    jmp main


ISR1:
    in r29, SREG
    push r29
    sei
    cpi flag, 0x00 ; check if light is still on
    brne LIGHT_EM_UP
    ldi temp, 0x01
    out PORTB, temp ; PB0 lights up
    ser flag ; set flag
    rcall delay_ms_normal ; 3 sec
    clr temp
    out PORTB, temp
    clr flag ; clear flag
    ldi r24, low(NORMAL_CYCLES)
    ldi r25, high(NORMAL_CYCLES)
    pop r29
```

```
    out SREG, r29
    reti
LIGHT_EM_UP:
    ser temp ; turn on leds
    out PORTB, temp
    inc active_interrupt_counter
    cpi active_interrupt_counter, 0x01
    ldi r24, low(NORMAL_CYCLES) ; renew timeout
    ldi r25, high(NORMAL_CYCLES)
    ldi r26, low(BLINKING_CYCLES)
    ldi r27, high(BLINKING_CYCLES)
    brne exit_int

    rcall delay_mS_blinking
    ldi temp , 0x01
    out PORTB, temp
exit_int:
    dec active_interrupt_counter
    pop r29
    out SREG, r29
    reti

 delay_mS_normal:
    ldi r23, 249
    loop_inn:
    dec r23
    nop
    brne loop_inn

    sbiw r24, 1
    brne delay_mS_normal


    ret
 delay_mS_blinking:
    ldi r31, 249
    loop_inn_blinking:
    dec r31
    nop
    brne loop_inn_blinking

    sbiw r26, 1
    brne delay_mS_blinking


    ret
```

Ο κώδικας C είναι ο παρακάτω:

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


bool active = false; // active = false means that PB0 is on for 3 seconds
                     // only when light PB0 is already on (active = true), should we light up
 ↪  all lights for 0.5sec
int normal_counter = 3000;      // counter for timer. (30 * 100 msec  = 3sec)
int blink_timer = 500; //counter for timer.
ISR (INT1_vect) {
```

```
13      //debouncing
14      do {
15          EIFR = 1<<INTF1;
16          _delay_ms(5);
17      } while (EIFR & 1<<INTF1);
18
19      if (active) {
20          normal_counter = 3000;
21          blink_timer = 500;
22          PORTB = 0xFF;
23          while(blink_timer-- > 0) {
24              sei();
25              _delay_ms(1);
26              cli(); //ensuring that blink_timer-- >0 happens atomically
27          }
28          PORTB = 0x01;
29      } else {
30          active = true;
31          PORTB = 0x01;
32          normal_counter = 3000;
33          while(normal_counter-- > 0) {
34              sei();
35              _delay_ms(1);
36              cli(); //ensuring that normal_counter-->0 happens atomically
37          }
38          PORTB = 0x00;
39          active = false;
40      }
41  }
42
43  int main(int argc, char** argv) {
44      EICRA = 1 << ISC11 | 1 << ISC10;
45      EIMSK = 1 << INT1;
46      sei();
47
48      DDRB  = 0xFF; // set PORTB as output
49
50      while(1); // infinite loop
51
52      return 0;
53  }
```