Χρήση των Timers και του ADC στον AVR

Εργαστήριο Μικροϋπολογιστών

Κριθαρίδης Κωνσταντίνος, *el*21045 Μπαλάτος Δημήτριος, *el*21170

29 Οκτωβρίου 2024

1 Ζήτημα 3.1

Στην αρχή του προγράμματος αποθηκεύουμε στο cseg σε διαδοχικές θέσεις μνήμης τις τιμές που θα παίρνει το duty cycle που θα δίνεται από τη μεταβλητή `DC_VALUE` .
Θέτουμε το `PORTB` ως έξοδο και το `PORTD` ως είσοδο και αρχικοποιούμε μέσω των `TCCR1A` , `TCCR1B` τον μετρητή `TMR1A` σε λειτουργία Fast PWM, 8 bit, non-inverting output με $N = 256$, έχοντας `BOTTOM` $= 0$ και `TOP` $= \textit{0x00ff} = 255$. Για να ανιχνεύουμε το πάτημα των `PD3` , `PD4` ενεργοποιούμε τις διακοπές `PCINT19` (`PD3`) και `PCINT20` (`PD4`) στον καταχωρητή `PCMSK2` , όπως και γενικά το *Pin Change Interrupt 2* μέσω του `PCICR` . Αρχικοποιούμε το i να ισούται με 6 (που αντιστοιχεί στην 6η θέση του πίνακα, δηλαδή στο duty cycle 50%) και τον καταχωρητή `Z` να δείχνει στην i-οστή θέση του πίνακα στη μνήμη προγράμματος. Τέλος, ενεργοποιούμε την καθολική σημαία διακοπών για να λειτουργεί ο κώδικας.

Στο κύριο πρόγραμμα θέτουμε το duty cycle να ισούται με τη θέση μνήμης που δείχνει ο καταχωρητής Z και περιμένουμε 10ms πριν επαναλάβουμε. Η ρουτίνα εξυπηρέτησης διακοπών pc2isr ανιχνεύει αλλαγή κατάστασης των PD3 και PD4. Έχοντας έναν βοηθητικό καταχωρητή στον οποίο έχουμε αποθηκευμένη κάθε φορά την προηγούμενη κατάσταση του PIND μπορούμε να καταλάβουμε ποιά αλλαγή στα PD3, PD4 είναι που μόλις συνέβη. Συγκεκριμένα, με το cur_PIND AND (NOT prev_PIND) ανιχνεύουμε το bit του PIND που μόλις έγινε 1 ενώ πριν ήταν 0. Αν πατήθηκε PD3 και δεν έχουμε φτάσει στο τέλος του πίνακα προχωράμε το i και το Z κατά 1 θέση μπροστά στον πίνακα, ενώ αν πατήθηκε το PD4 και δεν έχουμε φτάσει στην αρχή του πίνακα, προχωράμε το i και το Z κατά 1 θέση πίσω στον πίνακα.

```
;
; Ex3_1.asm
;
; Created: 10/25/2024 5:06:29 PM
; Author : User
;

.include "m328PBdef.inc"

.org 0x0
        rjmp reset

.org 0xA
        rjmp pc2isr

.def temp = r16
```

```
.def DC_VALUE = r17
.def DC_INC = r18 ; DC_INC = 1 -> DC_VALUE increasing, DC_INC = 0 -> DC_VALUE decreasing
.def i = r19
.def prev_PIND = r20
.def cur_PIND = r21

DUTY: .DB 5, 26, 46, 66, 87, 107, 128, 148, 168, 189, 209, 230, 250 ; 255*(2+8k)/100, k = 0 ... 12. Initial k = 6.
.equ DUTY_LAST = 12
.equ DUTY_START = 6

pc2isr:
        push temp
        in temp, SREG
        push temp
        in cur_PIND, PIND                ; Get the current set PIND bits
        mov temp, prev_PIND
        com temp                                ; Get the previous clear PIND bits
        and temp, cur_PIND                ; Get the 1 bit that was just set and caused the PCINT2 interrupt
        sbrc temp, 3                        ; Skip next instruction if PD3 was not pressed
        rjmp increase
        sbrc temp, 4                        ; Skip next instruction if PD4 was not pressed
        rjmp decrease
        rjmp end
increase:
        cpi i, DUTY_LAST
        breq end
        inc i
        adiw Z, 1
        rjmp end
decrease:
        cpi i, 0
        breq end
        dec i
        sbiw Z, 1
end:
        mov prev_PIND, cur_PIND
        pop temp
        out SREG, temp
        pop temp
        rjmp reset

reset:
        ; Init stack pointer
        ldi temp, high(RAMEND)
        out SPH, temp
        ldi temp, low(RAMEND)
```

```
        out SPL, temp

        ; Set PORTB as output
        ser temp
        out DDRB, temp

        ; Set PORTD as input
        clr temp
        out DDRD, temp

        ; Fast PWM, 8 bit, non-inverting output, N = 256. BOTTOM = 0, TOP = 0x00ff = 255
        ldi temp, (1<<WGM10) | (1<<COM1A1)
        sts TCCR1A, temp
        ldi temp, (1<<WGM12) | (1<<CS12)
        sts TCCR1B, temp

        ; Enable PCINT19 (PD3), PCINT20 (PD4) interrupts
        ldi temp, (1<<PCINT19) | (1<<PCINT20)
        sts PCMSK2, temp

        ; Enable Pin Change Interrupt 2: PCINT[23:16]
        ldi temp, (1<<PCIE2)
        sts PCICR, temp

        ; Set previous state of PIND = 0
        ldi prev_PIND, 0

        ; Initialize i to duty cycle starting position: 6 (50%)
        ldi i, DUTY_START

        ; Load the starting address of the duty cycle value into Z
        LDI ZH, HIGH(2*DUTY+DUTY_START)
        LDI ZL, LOW(2*DUTY+DUTY_START)

        sei

; Replace with your application code
start:
        lpm DC_VALUE, Z
        sts OCR1AL, DC_VALUE

        ldi r24, LOW(10*16)        ;
        ldi r25, HIGH(10*16)        ; Set delay (10ms * 16)
        rcall delay_mS                        ; Delay for 10ms

        rjmp start

; delay of 1000*F1+6 cycles (almost equal to 1000*F1 cycles)
delay_mS:
; total delay of next 4 instruction group = 1+(249*4-1) = 996 cycles
```

```
        ldi r23, 249                        ; (1 cycle)
loop_inn:
        dec r23                                     ; 1 cycle
        nop                                             ; 1 cycle
        brne loop_inn                       ; 1 or 2 cycles

        sbiw r24, 1                                 ; 2 cycles
        brne delay_mS                       ; 1 or 2 cycles

        ret                                             ; 4 cycles
```

2 Ζήτημα 3.2

Το μεγαλύτερο μέρος του κώδικα είναι μετατροπή της παραπάνω άσκησης από Assembly σε C. Μερικά σημαντικά σημεία:

- Ενεργοποιούμε επιπλέον το `ADC0` , με τα κατάλληλα flags:
  - $V_{ref} = 5\,\text{V}$
  - Χρήση `ADC0`
  - Enable
  - No interrupt
  - Refresh 125 kHz
- Αξιοποιούμε μέρος του `PORTD` για input (pins 3, 4) και το υπόλοιπο για output
- Το timer handler αντιμετωπίζει και την αλλαγή αναμένου led του `PORTD` , ανάλογα τον μέσο όρο μετρήσεων του `ADC0`
  - Υπολογίζουμε τον μέσο όρο μέσω κυκλικής λίστας υλοποιημένης με πίνακα

```
/*
 * main.c
 *
 * Created: 10/25/2024 11:40:30 PM
 *  Author: User
 */

#include <xc.h>

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

enum mode = {mode1, mode2};

#define DUTY_START 6
#define DUTY_LAST 12
#define DUTY_INC 0x30 // PD6
#define DUTY_DEC 0x10 // PD5

int8_t i;
uint8_t DC_VALUE;
uint8_t prev_PIND;
```

```
uint8_t DUTY[] = {5, 26, 46, 66, 87, 107, 128, 148, 168, 189, 209, 230, 250};
uint8_t out_led;
uint16_t measures[16], sum;
uint8_t measure_pos = 0;

ISR(PCINT2_vect){
        // Handle button presses
        uint8_t cur_PIND = PIND;
        uint8_t temp = (~prev_PIND) & cur_PIND; //get the pin that was just set
        if((temp & DUTY_INC) && i < DUTY_LAST)
                i++;
        else if((temp & DUTY_DEC) && i > 0)
                i--;
        // Handle ADC
        while (ADCSRA & (1 << ADSC));
        int16_t val = (ADCH << 7) | (ADCL);
        // Update sum;
        sum += val - measures[pos];
        measures[pos] = val;
        pos = (pos + 1) % 16;
        // Use average
        out_led = ((sum >> 4) - 1) / 200;
        if (out_led > 4) out_led = 4;
        PORTD = (1 << out_led);
}

int main(void)
{
        // Set PORTB as output
        DDRB = 0xff;
        // Set PORTC as input
        DDRC = 0x00;
        // Set PORTD as input - output
        DDRD = 0x0f;

        // Fast PWM, 8 bit, non-inverting output, N = 256. BOTTOM = 0, TOP =
 ↪  0x00ff = 255
        TCCR1A = (1<<WGM10) | (1<<COM1A1);
        TCCR1B = (1<<WGM12) | (1<<CS12);

        // Enable PCINT19 (PD3), PCINT20 (PD4) interrupts
        PCMSK2 = (1<<PCINT19) | (1<<PCINT20);

        // Enable Pin Change Interrupt 2: PCINT[23:16]
        PCICR = (1<<PCIE2);

        prev_PIND = 0;

        i = DUTY_START;

        // Init ADC:
```

```
75          //   Vref = 5V, ADC0
76          ADMUX = (1 << VREF0) | (0 << MUX0);
77          //   Enable, no interrupt, no conversion, 125 kHz
78          ADCSRA = (1 << ADEN) | (0 << ADCS) | (0 << ADIE) | (7 << ADPS0);
79
80          sei();
81
82          while(1)
83          {
84                  OCR1AL = DC_VALUE;
85                  _delay_ms(10);
86          }
87  }
```

3 Ζήτημα 3.3