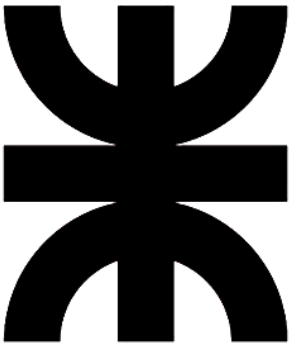
UNIVERSIDAD TECNOLÓGICA NACIONAL



FACULTAD REGIONAL PARANÁ

CARRERA: Ingeniería Electrónica
CÁTEDRA: Técnicas Digitales II

Trabajo Práctico N°10
Control de intensidad de LED con PWM

ALUMNOS:
Battaglia Carlo
Escobar Gabriel

Paraná, 17 de noviembre de 2022

1. Actividades

1. El circuito "base" del cual se deberá partir es el utilizado en el trabajo práctico Nro 8.

2. El circuito deberá tener, además:

a. Un teclado matricial de 3x3 teclas, dispuestos convenientemente.

3. El funcionamiento general del circuito es:

a. El sistema, al encendido, deberá estar todo apagado (incluido el display).

b. En la medida que se presiona una tecla, se deberá mostrar la tecla presionada en el display (de 1 a 9).

c. Transcurridos 3 segundos, y si no se presiona ninguna tecla, se apagará el display que indica la tecla presionada.

d. En el momento que se presiona la tecla, se enviará una cadena hacia la computadora informando la tecla presionada. Dicha cadena será:

i. $TD2,<número tecla>*

ii. Donde <número tecla> es el valor ASCII de la tecla presionada. Los símbolos < y > no se envían.

iii. Ejemplo: Se presiona la tecla número 3, se envía a la PC la cadena $TD2,3*

2. Código

En el primer bloque de código hacemos las declaraciones correspondientes.

Renombramos los registros $r16$ y $r17$ como *temp* y *aux*.

Declaramos al registro general de entrada/salida $GPIOR0$ para utilizarlo a modo de registro de banderas, en el que alojaremos la bandera $PressKey$ para detectar pulsaciones del teclado, y $ContarSeg$ para contar segundos.

```
;
;***********************************
; Técnicas Digitales II
; Autor: Battaglia - Escobar
; for AVR: atmega328p (Arduino UNO)
; clock frequency: 16MHz
;***********************************

.ifndef F_CPU
.set F_CPU = 16000000
.endif
;=============================================
; Declarations for register
.def temp     = r16
.def aux    = r17

;=============================================
; Declarations for label
.set Flags0     = GPIOR0
.set PressKey   = $0
.set ContarSeg    = 1

;=============================================
; Etiquetas
.equ  baud = 9600     ;Baud Rate

```

```
;=========================================
; Data Segment
.dseg
DATA_KB:       .byte 1
tres_seg:      .byte 1
BaseTime1ms:     .byte 1
BaseTime20ms:    .byte 1
BaseTime100ms:     .byte 1

;=========================================
; EEPROM Segment
.eseg
VAR_EEPROM:    .db   $AA

;=========================================
; Code Segment
.cseg
.org RWW_START_ADDR
```

Seguidamente, en el cuerpo principal del programa, inicializamos la el *StackPointer* apuntándolo a la parte baja de la memoria RAM.

Luego llamamos a las funciones correspondientes para inicializar la USART0, el TIMER0 y los puertos. Cada una de estas funciones será abordada más adelante.

Antes de entrar en el bucle principal, habilitamos las interrupciones.

Ahora sí en el bucle principal *loop*, esperamos hasta que la bandera *PressKey* indique la pulsación de una tecla, enviando entonces el código "$TD2, - * \n", con el número presionado en lugar del -.

Luego se baja la bandera en cuestión y se escribe el digito en *PORTD*.

Finalmente levantamos la bandera *ContarSeg*, para indicar que debe comenzar a contarse 3 segundos antes de apagar el display nuevamente.

```
;======================
; Main body of program:
Reset:
  ldi R16, LOW(RAMEND)       ; Lower address byte RAM byte lo.
  out SPL, R16             ; Stack pointer initialise lo.
  ldi R16, HIGH(RAMEND)      ; Higher address of the RAM byte hi.
  out SPH, R16         ; Stack pointer initialise hi.
; write your code here
  call Init_USART0
  call Init_Timer0
  call Init_Port

  ;ldi r19,(1 << DDB5)
  ;out DDRB, r19

  ;clr r18
  sei
Loop:
  ;sbic Flags0, TresSeg
  ;rcall display_off

  sbis Flags0,PressKey
  rjmp Loop

  ldi temp, '$'
  rcall Tx_Byte_USART0     ; Send Byte
```

```
  ldi temp, 'T'
  rcall Tx_Byte_USART0    ; Send Byte
  ldi temp, 'D'
  rcall Tx_Byte_USART0    ; Send Byte
  ldi temp, '2'
  rcall Tx_Byte_USART0    ; Send Byte
  ldi temp, ','
  rcall Tx_Byte_USART0    ; Send Byte
  lds temp,DATA_KB
  rcall Tx_Byte_USART0    ; Send Byte
  ldi temp, '*'
  rcall Tx_Byte_USART0    ; Send Byte
  ldi temp, '\n'
  rcall Tx_Byte_USART0    ; Send Byte

  in temp, Flags0           ; Tecla enviada
  cbr temp, (1 << PressKey)     ; PressKey = 0 en Flags0
  out Flags0, temp

  rcall write_digit

  clr temp
  sts tres_seg, temp      ;reseteo el contador de 3 segundos

  in temp, Flags0
  sbr temp, (1 << ContarSeg)  ;empiezo a contar 3 segundos
  out Flags0, temp

  rjmp Loop             ; loop back to the start
```

Esta subrutina inicializa el timer 0 en modo normal con un preescalado de 64 y habilita las interrupciones por overflow.

```
;=========================================
; Init_Timer0
; Inicia el Timer 0 para desborde cada 1 mseg
; Tovf = 2^n * Prescaler / Fio
; Mod = Tovf * Fio / Prescaler
Init_Timer0:
  ;Set the Timer Mode to Normal
  ;TCCR0A
  ; COM0A1 | COM0A0 | COM0B1 | COM0B0 | - | - | WGM01 | WGM00
  ; 0    0      0      0        0   0
  in  temp,TCCR0A
  cbr temp,1<<WGM00
  cbr temp,1<<WGM01
  out TCCR0A,temp

  ;TCCR0B
  ; FOC0A | FOC0B | - | - | WGM02 | CS02 | CS01 | CS00
  ; 0   0       0   0    1    1
  in  temp,TCCR0B
  cbr temp,1<<WGM02
  out TCCR0B,temp

  ;Activate interrupt for Overflow
  ;TIMSK0
  ; - | - | - | - | - | OCIE0B | OCIE0A | TOIE0
  ;           0    0      1
```

```
  lds temp,TIMSK0
  sbr temp,1 << TOIE0
  sts TIMSK0,temp

  ;Set TCNT0
  ldi temp,6
  out TCNT0,temp

  ;Start the Timer (prescaler %64)
  in  temp,TCCR0B
  sbr temp,1<<CS00
  sbr temp,1<<CS01
  cbr temp,1<<CS02
  out TCCR0B,temp
  ret
```

Init_Port inicializa los puertos a utilizar, poniendo a los puertos D y C como salida y al puerto B como entrada.

```
;=========================================
; Init_Port
; Inicia el Puerto PD7-4 como salida
; Puerto C3-C0 como entrada
Init_Port:
  ldi temp, (1<<DDD7) | (1<<DDD6) | (1<<DDD5) | (1<<DDD4)
  out DDRD, temp      ; PortD como salida
  clr temp        ; Borro el PortD
  ldi temp, (1<<PD7) | (1<<PD6) | (1<<PD5) | (0<<PD4)
  out PORTD, temp     ;Puerto en ALTO

  ldi temp, (1<<DDC5) | (1<<DDC4) | (1<<DDC3) | (1<<DDC2) | (1<<DDC1)
    | (1<<DDC0)
  out DDRC, temp      ; PortC como salida
  clr temp        ; Borro el PortC
  out PORTC, temp

  ldi aux, (1<<DDB3) | (1<<DDB2) | (1<<DDB1) | (1<<DDB0)
  out PORTB, aux      ; Activo resistencias de Pull-Up
  out DDRB, temp      ; PortB como entrada

  ret
```

La subrutina *Init_USART0* inicializa la transmisión serie a una velocidad de 9600 baudios.

```
;=========================================
; Init_USART0
; Inicializa la transmisión serie: 8N1 9600
; Parámetro: No
; Retorno: No
Init_USART0:
{
      ;Set Baud Rate
      ldi r17,high(F_CPU/(16*baud)-1)
      ldi r16,low(F_CPU/(16*baud)-1)
      sts UBRR0H, r17
      sts UBRR0L, r16

      ; Enable receiver and transmitter
      ldi r16, (1<<TXEN0)
```

```
16        sts UCSR0B,r16
17
18        ; Set frame format: 8data, 1stop bit
19        ldi r16, (0<<USBS0)|(3<<UCSZ00)
20        sts UCSR0C,r16
21        ret
22  }
```

Y *Tx_Byte_USART0* envía bytes individuales a través de la comunicación serie ya establecida.

```
1  ;=========================================
2  ; Tx_Byte_USART0
3  ; Envía dato por USART0
4  ; Parámetro: R16 -> dato que se envía
5  ; Retorno: No
6  Tx_Byte_USART0:
7  {
8         ; Wait for empty transmit buffer
9         lds r17,UCSR0A      ;Load into R17 from SRAM UCSR0A
10        sbrs r17,UDRE0      ;Skip next instruction If Bit Register is
       set
11        rjmp Tx_Byte_USART0
12        ; Put data (r16) into buffer, sends the data
13        sts UDR0,r16
14        ret
15 }
```

En cuanto a las interrupciones, hacemos uso de la interrupción por overflow del *timer*0.

Como previamente definimos un preescalado de 64, y la frecuencia de trabajo del microcontrolador es 16$[MHz]$, la frecuencia resultante es 250$[kHz]$.

Dado que timer 0 es de 8 bits, sabemos que ocurrirá una interrupción por overflow cada $255 * 1/250000[s]$, es decir, 1,02$[ms]$.

Si en lugar de contar desde 0 forzamos al timer a contar a partir de 6, la interrupción ocurrirá cada 250 pulsos de clock. Esto se traduce entonces a $250 * 1/250000[s]$, lo que equivale exactamente a 1$[ms]$.

Esto nos permite contar milisegundos de forma directa, y lo utilizaremos para esperar 200$[ms]$ entre cada lectura de teclado.

A su vez, utilizaremos esta cuenta de 200$[ms]$ para detectar el paso de 3$[s]$. Esto lo logramos contando 15 veces la ocurrencia de los 200 overflows anteriores, esto es $15 * 200[ms] = 3000[ms] = 3[s]$, y utilizaremos esta referencia temporal para apagar el display.

```
1  ;=========================
2  ; Rutinas de interrupción
3
4  isr_OVF0_handler:
5    push temp
6    ldi temp,6
7    out TCNT0,temp
8    lds temp, BaseTime1ms
9    inc temp
10   cpi temp,200       ;chequea si pasaron 200ms
11   brne notime
12   rcall Read_KEY       ;Leo el teclado
13   ldi temp, 0
```

```
  sbis Flags0, ContarSeg
  rjmp notime
  lds temp, tres_seg
  inc temp
  cpi temp, 15      ;si pasaron 15 veces 200ms son 3000ms=3s
  brne no_3_seg

  rcall display_off

  ldi temp, 0
no_3_seg:
  sts tres_seg, temp
  ldi temp, 0
notime:
  sts BaseTime1ms, temp
  pop temp
  reti          ; Timer0 overflow interrupt
```

La siguiente subrutina *Read_KEY* lee la tecla presionada en el teclado matricial y la escribe directamente sobre *PORTD*, donde se encuentra conectado el display de 7 segmentos.

```
;=============================================
; Read_KEY
; Lee un teclado matricial y escribe directamente a PORTD
; Parámetro: No
Read_KEY:
ROW_1:
  ;ldi aux,0x70 ;Activo R1    0111 0000
  ;out PORTD,aux
  CBI PORTD,7
  nop
  in  temp,PINB
  andi temp,0x0f
  cpi temp,0x0b ;0000 1011
  breq IS_KEY_3
  cpi temp,0x0d ;0000 1101
  breq IS_KEY_2
  cpi temp,0x0e ;0000 1110
  breq IS_KEY_1
  rjmp ROW_2
IS_KEY_3:
  ldi temp,'3'
  sts DATA_KB,temp
  in temp, Flags0         ; Tecla presionada
  sbr temp, (1 << PressKey)   ; PressKey = 1 en Flags0
  out Flags0, temp
  rjmp out_read_key
IS_KEY_2:
  ldi temp,'2'
  sts DATA_KB,temp
  in temp, Flags0         ; Tecla presionada
  sbr temp, (1 << PressKey)   ; PressKey = 1 en Flags0
  out Flags0, temp
  rjmp out_read_key
IS_KEY_1:
  ldi temp,'1'
  sts DATA_KB,temp
  in temp, Flags0         ; Tecla presionada
```

```
38    sbr temp, (1 << PressKey)    ; PressKey = 1 en Flags0
39    out Flags0, temp
40    rjmp out_read_key
41  ROW_2:
42    ;ldi aux,0xB0 ;Activo R2    1011 0000
43    ;out PORTD,aux
44    SBI PORTD,7
45    CBI PORTD,6
46    nop
47    in  temp,PINB
48    andi temp,0x0f
49    cpi temp,0x0b ;0000 1011
50    breq IS_KEY_6
51    cpi temp,0x0d ;0000 1101
52    breq IS_KEY_5
53    cpi temp,0x0e ;0000 1110
54    breq IS_KEY_4
55    rjmp ROW_3
56  IS_KEY_6:
57    ldi temp,'6'
58    sts DATA_KB,temp
59    in temp, Flags0          ; Tecla presionada
60    sbr temp, (1 << PressKey)    ; PressKey = 1 en Flags0
61    out Flags0, temp
62    rjmp out_read_key
63  IS_KEY_5:
64    ldi temp,'5'
65    sts DATA_KB,temp
66    in temp, Flags0          ; Tecla presionada
67    sbr temp, (1 << PressKey)    ; PressKey = 1 en Flags0
68    out Flags0, temp
69    rjmp out_read_key
70  IS_KEY_4:
71    ldi temp,'4'
72    sts DATA_KB,temp
73    in temp, Flags0          ; Tecla presionada
74    sbr temp, (1 << PressKey)    ; PressKey = 1 en Flags0
75    out Flags0, temp
76    rjmp out_read_key
77  ROW_3:
78    ;ldi aux,0xD0 ;Activo R3    1101 0000
79    ;out PORTD,aux
80    SBI PORTD,6
81    CBI PORTD,5
82    nop
83    in  temp,PINB
84    andi temp,0x0f
85    cpi temp,0x0b ;0000 1011
86    breq IS_KEY_9
87    cpi temp,0x0d ;0000 1101
88    breq IS_KEY_8
89    cpi temp,0x0e ;0000 1110
90    breq IS_KEY_7
91    rjmp out_read_key
92  IS_KEY_9:
93    ldi temp,'9'
94    sts DATA_KB,temp
95    in temp, Flags0          ; Tecla presionada
```

```
  sbr temp, (1 << PressKey)    ; PressKey = 1 en Flags0
  out Flags0, temp
  rjmp out_read_key
IS_KEY_8:
  ldi temp,'8'
  sts DATA_KB,temp
  in temp, Flags0          ; Tecla presionada
  sbr temp, (1 << PressKey)    ; PressKey = 1 en Flags0
  out Flags0, temp
  rjmp out_read_key
IS_KEY_7:
  ldi temp,'7'
  sts DATA_KB,temp
  in temp, Flags0          ; Tecla presionada
  sbr temp, (1 << PressKey)    ; PressKey = 1 en Flags0
  out Flags0, temp
  ;rjmp out_read_key
out_read_key:
  in aux, PORTD
  ori aux,0xE0          ;Desactivo los renglones
  out PORTD,aux
  ret
```

La subrutina encargada de apagar el display es la siguiente, *display_off*. Ésta apaga todos los segmentos del display y resetea la bandera *ContarSeg*, puesto que ya pasaron 3 segundos.

```
display_off:
  clr temp
  out PORTC, temp
  cbi PORTD, 4

  in temp, Flags0
  cbr temp, (1 << ContarSeg)  ;dejo de contar 3 segundos
  out Flags0, temp

  ret
```

La siguiente subrutina convierte el dígito almacenado en *DATA_KB* a 7 segmentos y lo muestra en el display conectado a *PORTD*.

```
write_digit:
  push temp
  lds temp, DATA_KB
  subi temp, 0x30
  rcall BCD_to_7_segment
  out PORTC, r16
  cbi PORTD, 4
  sbrc temp, 6
  sbi PORTD, 4
  pop temp
ret
```

La función que convierte los números BCD a su codificación en 7 segmentos es la siguiente, que hace uso de una tabla de conversión.

```
;=============================================
; BCDTo7Segment
;
```

```
; Convierte el valor, pasado en el registro r16, a una representación en
; display de 7 segmentos, de manera que:
; Dp g f e d c b a
; B7 B6 B5 B4 B3 B2 B1 B0
;
BCD_to_7_segment:
push ZH
push ZL
ldi ZH,HIGH(2*BCDTo7Seg) ; Carga la tabla
ldi ZL,LOW(2*BCDTo7Seg)
add ZL,r16
lpm
mov r16,R0
pop ZL
pop ZH
ret

; Tabla de conversión decimal a 7 segmentos
BCDTo7Seg:
.db 0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F
```