Laboratorio de Microprocesadores - 86.07

Trabajo Práctico Obligatorio N°6

Profesor:			Ing. Guillermo Campiglio								
Cuatrimestre/Año:			1°/2020								
Turno de las clases prácticas			Miércoles								
Jefe de trabajos prácticos:			Ing. Pedro Ignacio Martos								
Docente guía:			Ing. Fabricio Baglivo, Ing. Fernando Pucci								
Autores			Seguimiento del proyecto								
Nombre	Apellido	Padrón									
Ivan Eric	Rubin	100 577									

Observaciones:

.....
.....
.....
.....
.....
.....
.....
.....
.....

Fecha de aprobación				Firma J.T.P

Coloquio	
Nota final	
Firma profesor	

1. Objetivo del trabajo

El objetivo de este trabajo es conseguir un mejor entendimiento de los timers que posee el microprocesador ATMega328p. También profundizar el manejo de interrupciones e implementar métodos anti-rebote para los pulsadores.

2. Descripción del trabajo

Se conectarán dos pulsadores al puerto D (específicamente a los pines PD1 y PD2) y acorde a como se esten presionando los botones, se hará parpadear un LED conectado al puerto B a distintas frecuencias. En el caso que no se presione ninguno, el led permanecerá encendido.

3. Diagrama de conexiones en bloques

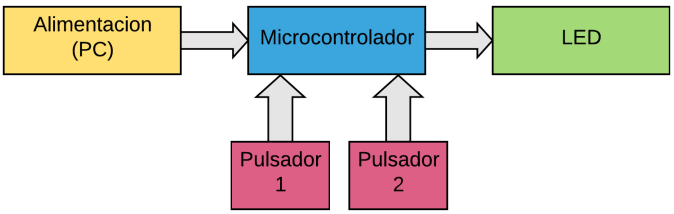

Figura 3.1: Diagrama de bloques.

4. Circuito Esquemático

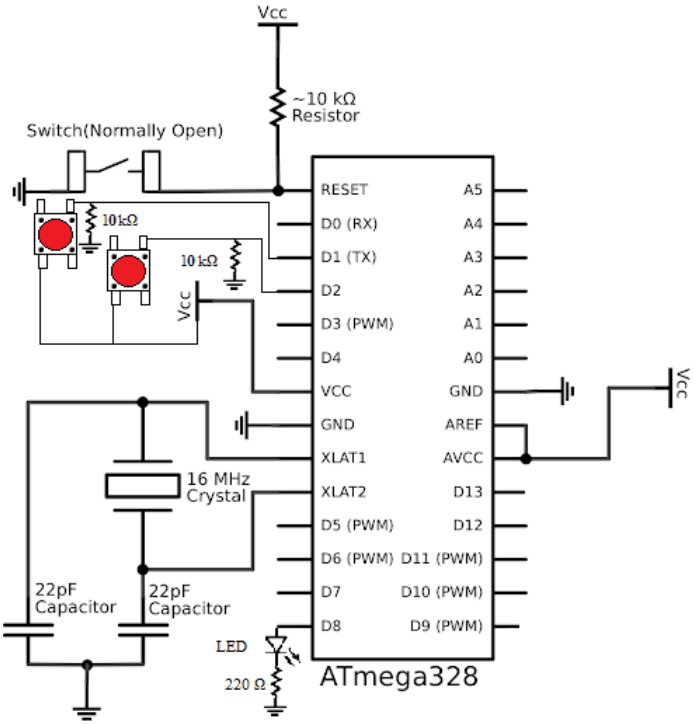

Figura 4.1: Circuito Esquemático.

5. Listado de componentes

- Arduino UNO + Conector PC (864$).
- Microprocesador ATMEGA 328P (Integrado en Arduino).
- Resistencia de $220\Omega \pm 1\,\%$ (5$).
- Diodo Emisor de Luz (LED) (1 x 10$).
- Resistencia de $10\text{k}\Omega \pm 1\,\%$ (5$).
- Pulsadores (2 x 40$).
- Cables Macho-Macho (40 cables x 100$).
- Protoboard (200$).

6. Diagrama de flujo del software

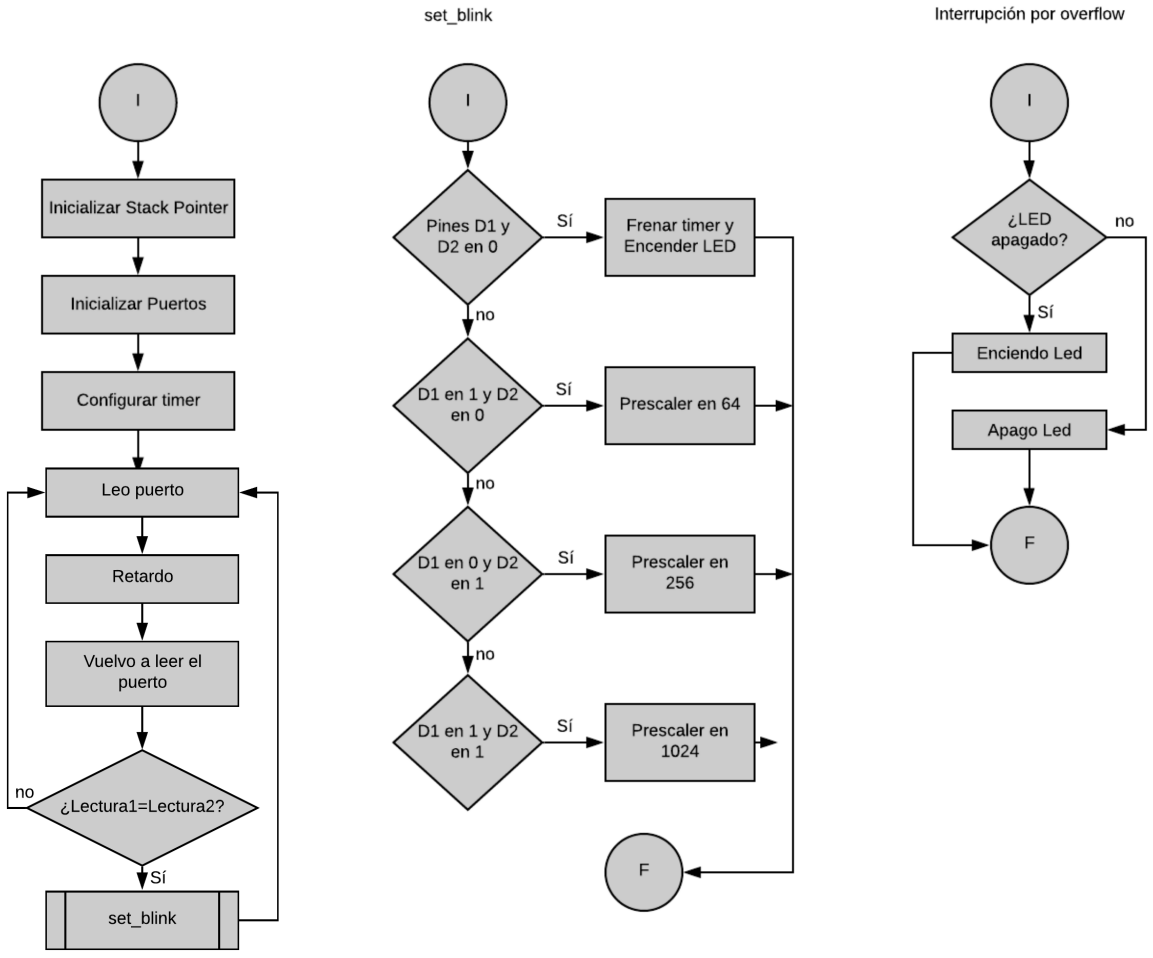

Figura 6.1: Diagrama de flujo.

7. Código de programa

```
/*
 *
   Trabajo Práctico Obligatorio 6
   Autor: Ivan Eric Rubin
 *
 */
.include "m328pdef.inc"

.def dummyr=R16
.def cmp_reg=R17

.cseg

.org 0x0000
   rjmp  config
.org OVF1addr
   rjmp   isr_timer
.org INT_VECTORS_SIZE

config:
; Inicializo el Stack Pointer.
   ldi      dummyr, HIGH(RAMEND)
   out      SPH, dummyr
   ldi      dummyr, LOW(RAMEND)
   out      SPL, dummyr

; Defino PORTB como puerto de salida.
   ldi      dummyr, 0xFF
   out      DDRB, dummyr

; Defino el PORTD como puerto de entrada.
   ldi      dummyr, 0x00
   out      DDRD, dummyr

; Enciendo la interrupcion por overflow del timer.
   ldi      dummyr, (1<<TOIE1)
   sts      TIMSK1, dummyr

; Habilito las interrupciones globales
   sei

main:
   in    dummyr, PIND                        ;Veo el estado del puerto D
   andi  dummyr, (1 << PIND1 | 1 << PIND2)   ;y guardo en un registro los valores
   de los pines
   mov      cmp_reg, dummyr
   rcall delay                               ;Delay para evitar efecto de rebote
   in    dummyr, PIND
   andi  dummyr, (1 << PIND1 | 1 << PIND2)
   cp    cmp_reg, dummyr
   breq  set_blink                           ;Defino el parpadeo si se mantuvo el
   valor luego del delay

   rjmp     main                             ;Sino, vuelvo a leer

; Se compara el puerto D con cada prescaler para definir la frecuencia de parpadeo
set_blink:
   cpi      dummyr, (0 << PIND1 | 0 << PIND2)
   breq  fijo
```

```
58
59      cpi       dummyr, (1 << PIND1 | 0 << PIND2)
60      breq   clock_64
61
62      cpi       dummyr, (0 << PIND1 | 1 << PIND2)
63      breq   clock_256
64
65      cpi       dummyr, (1 << PIND1 | 1 << PIND2)
66      breq   clock_1024
67
68      fijo:
69         ldi       dummyr, 0x00
70         sts       TCCR1B, dummyr          ; Se frena el timer.
71         sbi       PORTB, 0                ; Se enciende el LED.
72      ret
73   ; Segun el puerto D se utiliza el prescaler adecuado
74      clock_64:
75         ldi       dummyr, 0x03
76         sts       TCCR1B, dummyr
77      ret
78
79      clock_256:
80         ldi       dummyr, 0x04
81         sts       TCCR1B, dummyr
82      ret
83
84      clock_1024:
85         ldi       dummyr, 0x05
86         sts       TCCR1B, dummyr
87      ret
88
89   ; Rutina de parpadeo del LED (cuando haya overflow)
90   isr_timer:
91         sbis   PORTB, 0
92         rjmp   enciendo
93         cbi       PORTB, 0
94   reti
95
96         enciendo:
97         sbi       PORTB, 0
98   reti
99
100  delay:
101        ldi       dummyr, 0x00                    ;TCNT0 en 0
102        out       TCNT0, dummyr
103
104        ldi       dummyr, (1 << CS02 | 1 << CS00)        ;timer0 en modo normal
105        out      TCCR0B, dummyr                   ;prescaler 1024.
106
107  delay_loop:
108        in     dummyr, TIFR0                   ;En caso de overflow en timer0
109        sbrs   dummyr, TOV0                    ;esquivo la siguiente instrucción
110        rjmp   delay_loop
111
112        ldi       dummyr, 0X00
113        out       TCCR0B, dummyr                  ;Desactivo timer0
114        ldi       dummyr, (1<<TOV0)
115        out       TIFR0, dummyr                   ;Limpio el flag de overflow
116
117        ret
```

8. Resultados

Luego de haber armado el circuito y cargando el microprocesador con el código se logró controlar el parpadeo del LED en el pin B0 a partir del estado de los pulsadores en los pines D1 y D2.

Para calcular la frecuencia con la que parpadea el LED con cada prescaler se debe tener en cuenta la frecuencia del oscilador que posee el Arduino UNO de 16 MHz y que el timer toma esta como referencia. Se calculan los valores a partir de la siguiente fórmula

$$f_{blink} = \frac{\frac{16MHz}{n_{prescaler}}}{2 \cdot 2^{16}} \tag{8.1}$$

Se obtiene que para el prescaler de 64 $f_{blink} = 1,9Hz$, para el de 256 $f_{blink} = 0,477Hz$ y para el de 1024 $f_{blink} = 0,12Hz$.

Como efecto anti-rebote del pulsador, se agregó un pequeño retardo después de definir el parpadeo para esperar hasta la siguiente detección.

9. Conclusiones

En este trabajo se logró entender el manejo de timers e interrupciones por overflow. También se destaca el gran rango de posibles implementaciones que tienen estos. Se logró generar un parpadeo de LEDs sin necesidad de ir decrementando o incrementando registros para generar retardos como se venía haciendo en los anteriores trabajos. Esto ayuda a que el código sea mas eficiente y robusto.