Laboratorio de Microprocesadores - 86.07

Trabajo Práctico Obligatorio N°6

Timers

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									
Santiago	López	100566									

Observaciones:

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

Fecha de aprobación				Firma J.T.P

Coloquio	
Nota final	
Firma profesor	

Índice

1. Objetivo

Hacer uso de los timers del micro para manejar la frecuencia de oscilación de un LED de acuerdo al valor de entrada.

2. Descripción

Se reciben dos señales digitales en los pines PD0 y PD1. En base al valor de ambos bits se determina el prescaler para dividir la frecuencia del clock utilizado, y así variar el tiempo en el que se produce un overflow en el contador del timer del micro. El overflow provoca una interrupción durante la cual se cambia el estado de un LED, demostrando así la frecuencia a que trabaja el timer.

3. Diagrama de conexiones en bloque

Las conexiones siguieron el esquema de la Figura 1.

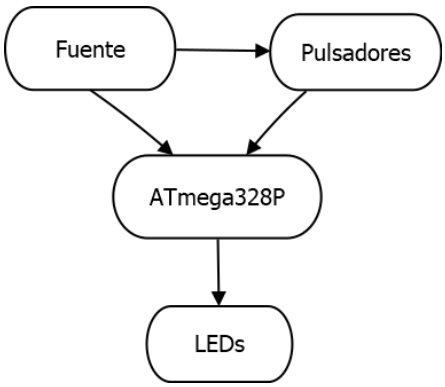

Figura 1: Diagrama de conexiones en bloque.

4. Esquemático

La Figura 2 muestra las conexiones eléctricas efectuadas en el práctico. Los componentes utilizados se encuentran en la sección siguiente. En caso de querer utilizarse la resistencia de *pull-up* interna del micro, se conectaría cada puslador entre el pin de entrada y tierra, configurando el progama para que interprete los 0V como '1' lógico. De esta forma se ahorra el uso de los resistores externos de 10kΩ.

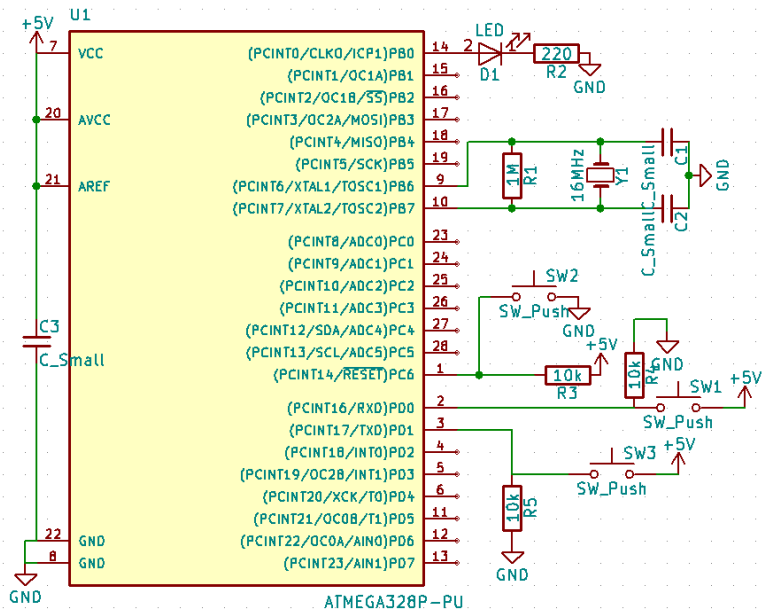

Figura 2: Circuito esquemático.

5. Listado de componentes

Los componentes utilizados fueron los listados a continuación:

- Placa de desarrollo Arduino UNO
- Resistores de 220Ω y 10kΩ
- LED de color rojo
- Pulsadores
- Protoboard
- Cables unipolares

6. Diagrama de flujo

En la Figura 3 se presentan los pasos a seguir por el programa. La primera rutina lleva los setups de los puertos y del stack del micro, y finalmente un loop de llamado a la segunda rutina. La segunda rutina lee la entrada al micro y así determina qué prescaler setear en el registro TCCR1B. La tercer rutina representa la interrupción efectuada al producirse un overflow en el timer.

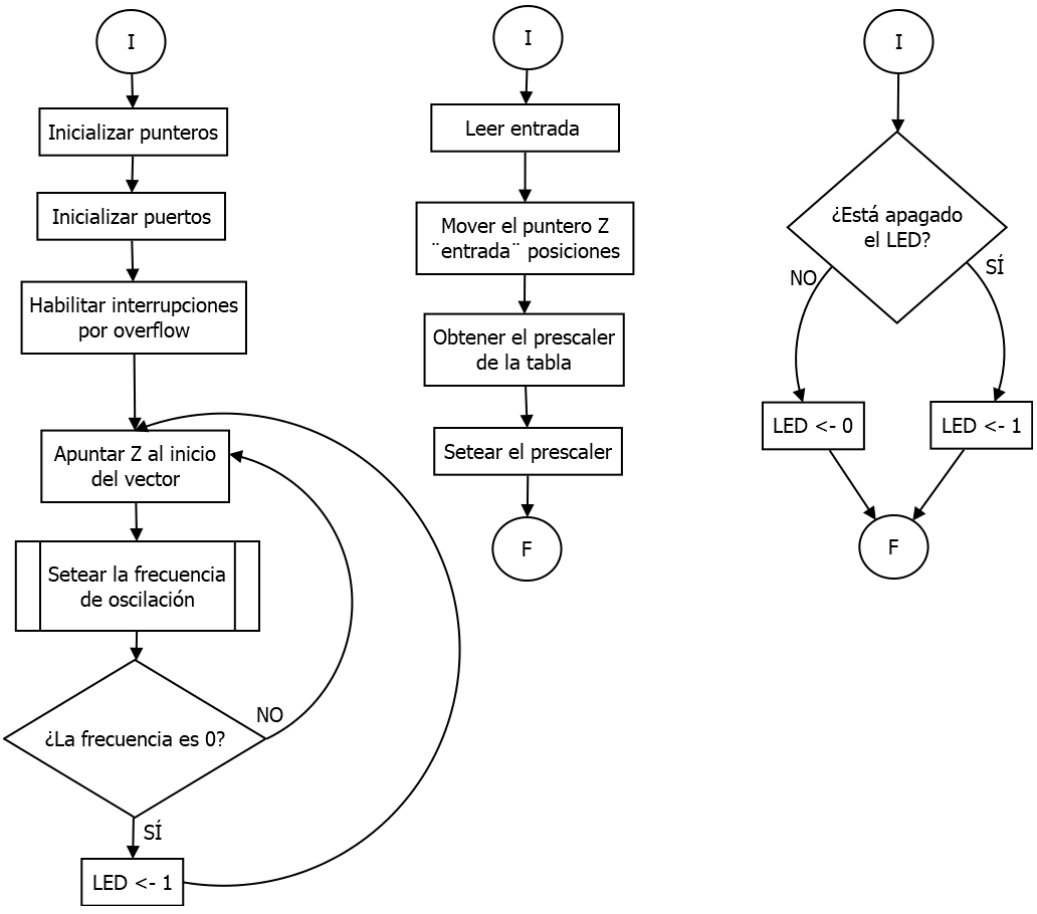

Figura 3: Diagrama de flujo.

7. Código fuente

El programa consiste en el setup de los registros y en buscar el código de prescale para setear la frecuencia de oscilación. Mientras tanto, cada vez que el timer tiene un overflow se produce una interrupción en donde se alterna el estado del LED.

Se implementó una tabla para hallar el estado adecuado para cada prescale con el fin de generalizar el algoritmo ante el caso de tener que añadir o quitar frecuencias de oscilación.

```
   1: .include "m328pdef.inc"
   2:
   3: .dseg
   4:
   5: .def conf = r16
   6: .def freq = r17
   7: .def aux = r18
   8: .def aux1 = r19
   9: .equ input = pind
  10: .equ A = 0  ;pins de entrada
  11: .equ B = 1
  12: .equ led_port = portb
  13: .equ led = 0
  14: .equ len = 4
  15:
  16:
  17: .macro init_sp
  18:     ldi conf, low(RAMEND)
  19:     out spl, conf
  20:     ldi conf, high(RAMEND)
  21:     out sph, conf
  22: .endmacro
  23:
  24:
  25: .macro init_xp
  26:     ldi xl, low (tccr1b)
  27:     ldi xh, high(tccr1b)
  28: .endmacro
  29:
  30:
  31: .cseg
  32: .org 0x0000
  33:     jmp main
  34: .org 0x001A
  35:     jmp timer_isr
  36:
  37:
  38:
  39: .org INT_VECTORS_SIZE
  40:
  41: main:
  42:     init_sp
  43:     init_xp
  44:     call setup_ports
  45:
  46:     call interrupt_enable
  47:     sbi led_port, led
  48:
  49: here:
  50:     call init_zp        ; apunto z al vector de prescales
  51:     call set_frequency
  52:     cpi freq, 0x00
  53:     brne here
  54:     sbi led_port, led   ; si no setee prescale dejo me aseguro que quede prendido el
led
  55:     jmp here
  56:
  57:
  58: set_frequency:
  59:     clr aux1
  60:     in aux, input
  61:     add zl, aux
  62:     adc zh, aux1
  63:     lpm freq, z     ; cargo el valor del vector
  64:     st x, freq
  65: ret
  66:
  67: interrupt_enable:
  68:     ldi conf, 0x01
```

```
69:     sts timsk1, conf ; interrupcion en V
70:     sei
71: ret
72:
73: setup_ports:
74:     clr conf
75:     out input, conf ; PIND como entrada
76:     ldi conf, 0x01
77:     out ddrb, conf  ; PB0 como salida
78: ret
79:
80: init_zp:
81:     ldi zl, low (vector << 1)
82:     ldi zh, high(vector << 1)
83: ret
84:
85: timer_isr:
86:     sbic led_port, led  ; si no esta apagado, lo apago
87:     rjmp turn_off
88:
89:     sbi led_port, led   ; si no, lo prendo y salgo
90: reti ; 1
91:
92: turn_off:
93:     cbi led_port, led
94: reti ; 0
95:
96: vector: .db 0x00, 0x03, 0x04, 0x05
```

8. Costos

A continuación se presenta un listado de los costos de los componentes utilizados en el práctico.

- Arduino UNO - $850
- Resistores - $50
- LED rojo - $20
- Pulsadores - $50
- Protoboard - $240
- Cables unipolares - $150

Sumando un costo total de $1360.

9. Conclusiones

El uso del timer brindó la facilidad de realizar una rutina cada un cierto tiempo, sin la necesidad de llamarla desde el código, permitiendo que esta se ejecute en cualquier parte del código. De esta forma fue posible actualizar la frecuencia de oscilación, sin que el el timer tenga que resetearse a mitad de un ciclo de conteo.