



## Laboratorio de Microprocesadores - 86.07

# Trabajo Práctico N°6: Timers

Profesor:			Ing. Guillermo Campiglio									
Cuatrimestre/Año:			1º/2020									
Turno de las clases prácticas			Miércoles									
Jefe de trabajos prácticos:			Ing. Pedro Ignacio Martos									
Docentes guía:			Ing. Fabricio Baglivo, Ing. Fernando Pucci									
Fecha de presentación:			29/07/2020									
Alumno			Seguimiento del proyecto									
Nombre	Apellido	Padrón										
Maximiliano Daniel	Reigada	100565										

### Observaciones:

.....

.....

.....

.....

.....

.....

.....

.....

Fecha de aprobación			Firma J.T.P

# Índice

1. Objetivo	1
2. Descripción	1
3. Diagrama de conexiones en bloques	1
4. Circuito esquemático	2
5. Listado de componentes	2
6. Diagrama de flujo del Software	3
7. Código de programa	4
8. Resultados	5
9. Conclusiones	6

## 1. Objetivo

El presente trabajo práctico tiene como objetivo progresar con el manejo de registros de timers de los que dispone el microcontrolador ATmega328P, generar interrupciones a partir de eventos asociados a estos, e implementar métodos de antirrebote de teclas.

## 2. Descripción

Mediante la implementación de una placa de microcontrolador de código abierto basado en el microchip ATmega328P, denominada Arduino Uno, se busca desarrollar un programa para hacer parpadear un LED conectado al pin PB0, en 3 frecuencias distintas o que lo deje encendido de manera fija, según los valores que haya en los pines de entrada PD0 y PD1 a partir las indicaciones de la siguiente tabla:

PD0	PD1	Estado del LED
0	0	Encendido fijo
0	1	Parpadea con prescaler CLK/64
1	0	Parpadea con prescaler CLK/256
1	1	Parpadea con prescaler CLK/1024

**Tabla 1:** Valores de referencia.

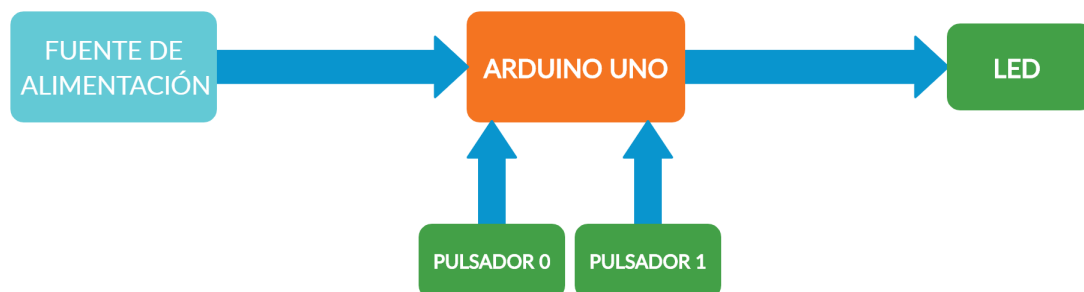
Para resolver esta práctica, se usa el `timer1` por medio de la interrupción por `overflow`. Cada vez que se produce un `overflow` se cambia el estado del LED mediante la rutina asociada a dicha interrupción, es decir, si está prendido se apaga y viceversa.

Cuando el LED se encuentra encendido de manera fija, el timer se encuentra apagado poniendo a cero los bits `CS12/1/0` del registro de configuración `TCCR1B`. En los otros tres casos, el timer cuenta los pulsos de clock divididos por prescaler 64 (`CS12/1/0=011`), 256 (`CS12/1/0=100`) y 1024 (`CS12/1/0=101`).

Por otra parte, en las entradas PD0 y PD1 se conectan 2 switches, que como producen rebotes al ser presionados, hacen necesario implementar rutinas antirrebote para detectar correctamente los valores de entrada.

## 3. Diagrama de conexiones en bloques

A continuación, puede verse el diagrama de conexiones en bloques correspondiente a este trabajo práctico:



**Figura 1:** Diagrama de conexiones en bloques.

## 4. Circuito esquemático

Se procede a realizar el esquemático del circuito a implementar. Cabe destacar que, si bien en la realidad los componentes se encuentran conectado al Arduino Uno y este a su vez a una fuente de tensión externa, resulta más útil representar en el esquema solo los componentes electrónicos esenciales del sistema, conectados a referencias de tensión de un determinado valor:

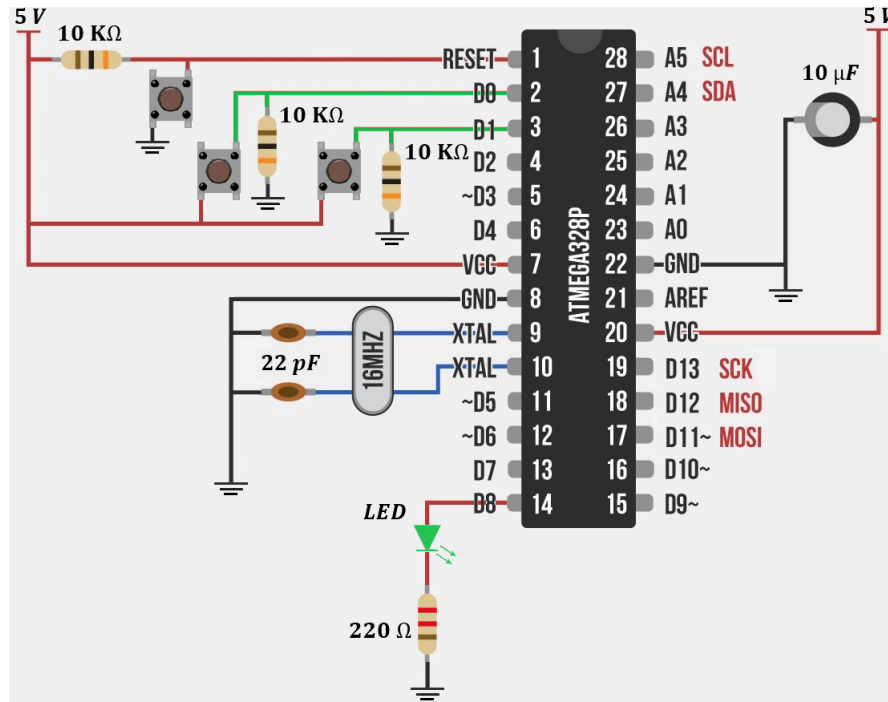


Figura 2: Circuito esquemático.

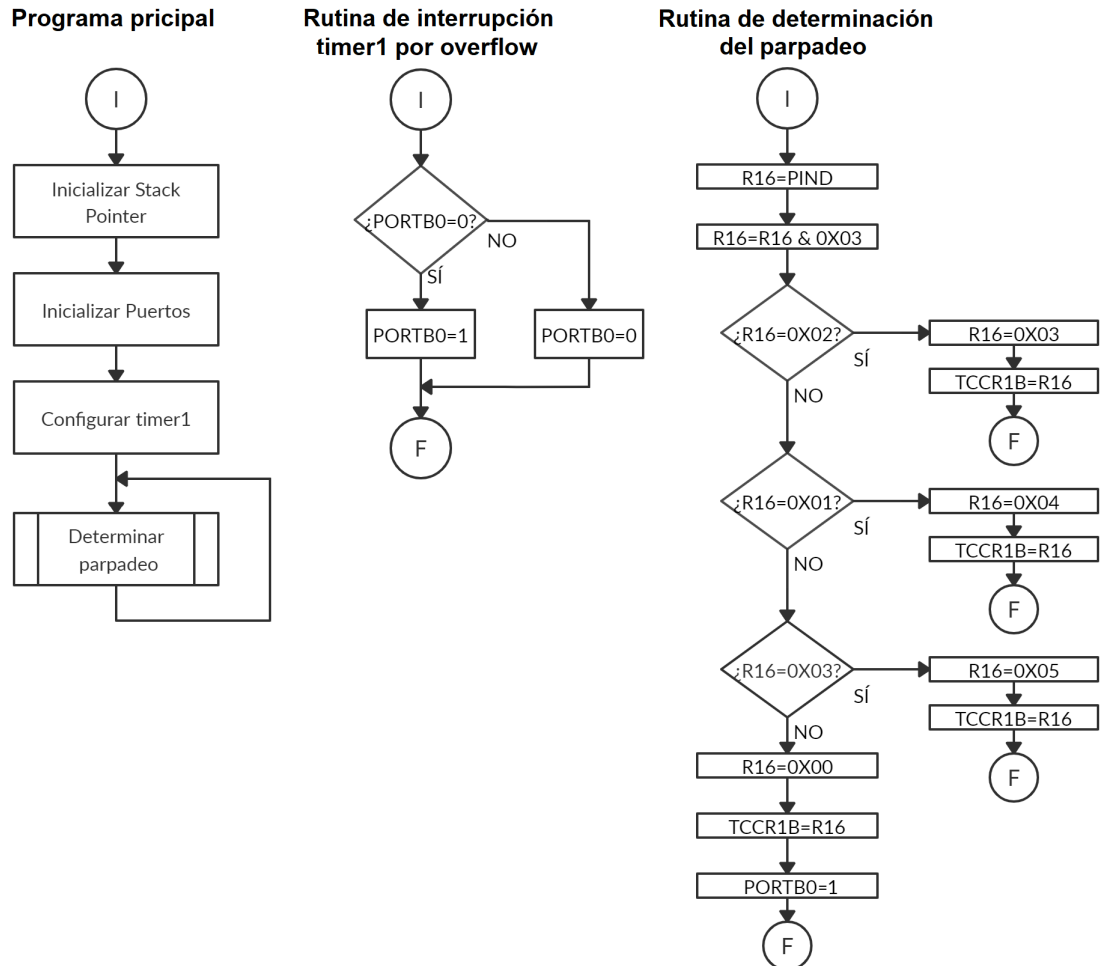
## 5. Listado de componentes

En el desarrollo de este trabajo práctico se utilizan los siguientes componentes:

- 1 LED verde de 5mm
- 1 Resistor de 220  $\Omega$
- 2 Resistores de 10 k $\Omega$
- 2 Pulsadores Dip Tact Switch 6x6x5mm
- 6 Cables de conexión
- 1 Protoboard
- 1 Arduino Uno (con microcontrolador ATmega328P incluido)

## 6. Diagrama de flujo del Software

A continuación, se explicitan los diagramas de flujo correspondientes al código desarrollado:



**Figura 3:** Diagramas de flujo.

## 7. Código de programa

A continuación, puede verse el código desarrollado para este proyecto:

```

1  ;*****
2  ; Código correspondiente al ejercicio planteado en el Trabajo Práctico 6.
3  ;
4  ; Alumno: Reigada Maximiliano Daniel
5  ; Padrón: 100565
6  ;*****
7
8  .INCLUDE "m328pdef.inc"
9
10 .DEF AUX=R16
11
12 .CSEG
13 .ORG 0X0000
14     RJMP    config
15
16 .ORG OVFIaddr
17     RJMP    isr_timer1
18
19 .ORG INT_VECTORS_SIZE
20
21 config:
22     LDI      AUX, HIGH(RAMEND)           ;Inicializo el SP al final de la RAM.
23     OUT      SPH, AUX
24     LDI      AUX, LOW(RAMEND)
25     OUT      SPL, AUX
26
27     LDI      AUX, 0X00                  ;Declaro al puerto D como entrada.
28     OUT      DDRD, AUX
29
30     LDI      AUX, 0XFF                  ;Declaro al puerto B como salida.
31     OUT      DDRB, AUX
32
33     LDI      AUX, (1 << TOIE1)          ;Habilito interrupción de timer1 por overflow.
34     STS      TIMSK1, AUX
35
36     SEI                                       ;Habilito interrupciones globales.
37
38 main:
39     RCALL    determinar_parpadeo
40     RJMP     main
41
42 determinar_parpadeo:
43     IN       AUX, PIND
44     ANDI     AUX, (1 << PIND1 | 1 << PIND0) ;Guardo en AUX el valor de PIND0 y PIND1.
45
46     CPI     AUX, (1 << PIND1 | 0 << PIND0)
47     BREQ    clock_64
48
49     CPI     AUX, (0 << PIND1 | 1 << PIND0)
50     BREQ    clock_256
51
52     CPI     AUX, (1 << PIND1 | 1 << PIND0)
53     BREQ    clock_1024
54
55     LDI     AUX, 0X00                    ;Como PIND0=0 y PIND1=0, detengo el timer1
56     STS     TCCR1B, AUX                  ;y dejo PBO encendido.
57     SBI     PORTB, 0
58     RET
59
60 clock_64:
61     LDI     AUX, 0X03                    ;Como PIND0=0 y PIND1=1, configuro el timer1
62     STS     TCCR1B, AUX                  ;para que cuente los pulsos de clock divididos
63     RET                                       ;por prescaler 64.
64
65 clock_256:
66     LDI     AUX, 0X04                    ;Como PIND0=1 y PIND1=0, configuro el timer1
67     STS     TCCR1B, AUX                  ;para que cuente los pulsos de clock divididos
68     RET                                       ;por prescaler 256.

```

```

69 clock_1024:
70     LDI      AUX, 0X05                ;Como PIND0=1 y PIND1=1, configuro el timer1
71     STS      TCCR1B, AUX              ;para que cuente los pulsos de clock divididos
72     RET                                  ;por prescaler 1024.
73
74
75 isr_timer1:
76     SBIC     PORTB, 0                  ;Si PORTB0=0, salto la próxima instrucción.
77     RJMP     apagar_led
78
79     SBI      PORTB, 0                  ;Pongo PORTB0 en estado lógico alto.
80     RETI
81
82 apagar_led:
83     CBI      PORTB, 0                  ;Pongo PORTB0 en estado lógico bajo.
84     RETI

```

## 8. Resultados

Luego de desarrollar el código descrito en las secciones anteriores y armar el circuito especificado, se logra controlar el parpadeo o estado de encendido fijo del LED conectado al pin PB0 a partir de los pulsadores dispuestos en los pines PD0 y PD1. Cabe destacar, que para cargar el programa en el microcontrolador se debe desconectar momentáneamente los pines del puerto D del resto del circuito, debido a que durante este proceso dichos pines son utilizados como medio de comunicación entre los dos microcontroladores que posee el Arduino Uno.

Por otra parte, considerando que en la placa de Arduino Uno el oscilador de cristal es de  $16MHz$ , y que el Atmega328p toma como referencia de clock dicha frecuencia, se puede calcular la frecuencia de oscilación en cada tipo de parpadeo a partir de:

$$f_n = \frac{16MHz}{2 \cdot 2^{16} \cdot n} \quad (1)$$

donde  $n$  es el valor del prescaler correspondiente a cada tipo de parpadeo, y  $2^{16}$  es la cantidad máxima de valores que se pueden representar mediante los dos registros TCNT1L y TCNT1H.

Con esto último, se tiene que:

PD0	PD1	$n$	$T_n$ [s]	$f_n$ [Hz]
0	1	64	0,524	1,9
1	0	256	2,097	0,477
1	1	1024	8,389	0,120

**Tabla 2:** Frecuencias de parpadeo.

En cuanto a las rutinas antirrebote pedidas en el enunciado, al utilizar el polling como método de detección del estado de los pulsadores, se logra evitar los efectos no deseados producidos por posibles rebotes mecánicos. Esto último se debe a que, al estar constantemente verificando el estado del puerto de entrada, a una frecuencia mayor que a las correspondientes a los posibles parpadeos y mucho mayor que a la que teóricamente se podría llegar a conseguir presionando rápidamente de manera intermitente alguno de los pulsadores, los efectos de rebote terminan resultando insignificantes al momento de determinar el comportamiento del LED.

Finalmente, se procede a realizar un listado de los componentes utilizados y sus costos:

Componentes	Costos
Led verde de 5mm	\$7,00
Pulsadores Dip Tact Switch	\$44,00
Resistor de 220 $\Omega$	\$5,50
Resistores de 10k $\Omega$	\$11,00
Cables de conexión	\$12,00
Protoboard	\$232,00
Arduino Uno	\$720,00
TOTAL	\$1031,50

## 9. Conclusiones

Tras haber realizado todos los pasos pedidos en el enunciado de este trabajo práctico, resta destacar las conclusiones que la experiencia ha aportado.

A partir de este trabajo se logra afianzar el manejo de timers del que puede disponer un microcontrolador, poniendo en práctica el cambio de estado lógico en un pin de salida mediante la interrupción asociada a un evento interno del dispositivo.

Es necesario hacer énfasis en todos los parámetros configurables del timer1 de los que se tuvo que tomar nota y especificar para esta práctica en particular a fin de hacerlo funcionar en modo normal. Así mismo, aquellas opciones de uso que no fueron implementadas, como por ejemplo las posibles configuraciones de los registros TCCR1A, TCCR1C e ICR1, quedan como buenas herramientas a tener en cuenta en futuras aplicaciones.

Por otra parte, el uso del polling como método para evitar efectos no deseados asociados a rebotes en pulsadores, se toma como un ejemplo de caso de aplicación en el que implementar esta metodología en vez del uso de interrupciones externas, simplifica la resolución del problema planteado.