Trabajo práctico N°4
Timers y PWM

Laboratorio de microprocesadores
Primer Cuatrimestre del 2021

Becker, Gonzalo Agustín	104291

Docentes:
Gerardo Stola
Fernando Cofman
Guido Salaya

Índice

1. Objetivos

Los objetivos del presente informe son mostrar el funcionamiento de las diferentes formas de temporización que posee un microcontrolador, a través de los *timers*, y estudiar además la utilización de la herramienta PWM (Pulse Width Modulation, por sus siglas en inglés) para generar, en este caso, ondas cuadradas con un duty cycle variable. Con esto en mente, a lo largo del texto se intenta desarrollar la explicación de dos ejemplos prácticos en donde se aplican estos temas.

2. Parpadeo de un LED a distintas frecuencias

2.1. Descripción del proyecto

En el siguiente link se encuentra la explicación del presente proyecto.

- Explicación de la primer parte

En este caso, se ha utilizado a un Arduino Nano como ISP para programar al microcontrolador. Para más detalles sobre este procedimiento, referirse al "TP N°1: Manejo de puertos", sección 2, de esta misma serie.

Este ejemplo funcionará a partir de dos pulsadores, los cuales controlarán la frecuencia de oscilación de un LED. La siguiente tabla resume las frecuencias que se obtendrán según que pulsador esté presionado:

Pulsador 1	pulsador 2	Estado
0	0	Encendido fijo
0	1	$f \simeq 0,95Hz$
1	0	$f \simeq 0,24Hz$
1	1	$f \simeq 0,06Hz$

Posteriormente se ahondará a mayor profundidad en el cálculo de las frecuencias. En adición, se estudiará la implementación de un programa en C para conseguir este comportamiento, y se explicará el concepto de *timer*, con todas sus características que aquí se usan.

2.2. Circuito esquemático

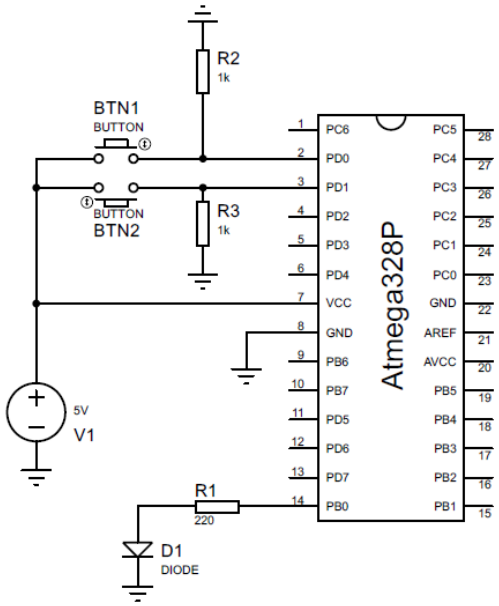

Figura 1: Circuito a implementar

2.3. Diagrama de flujo

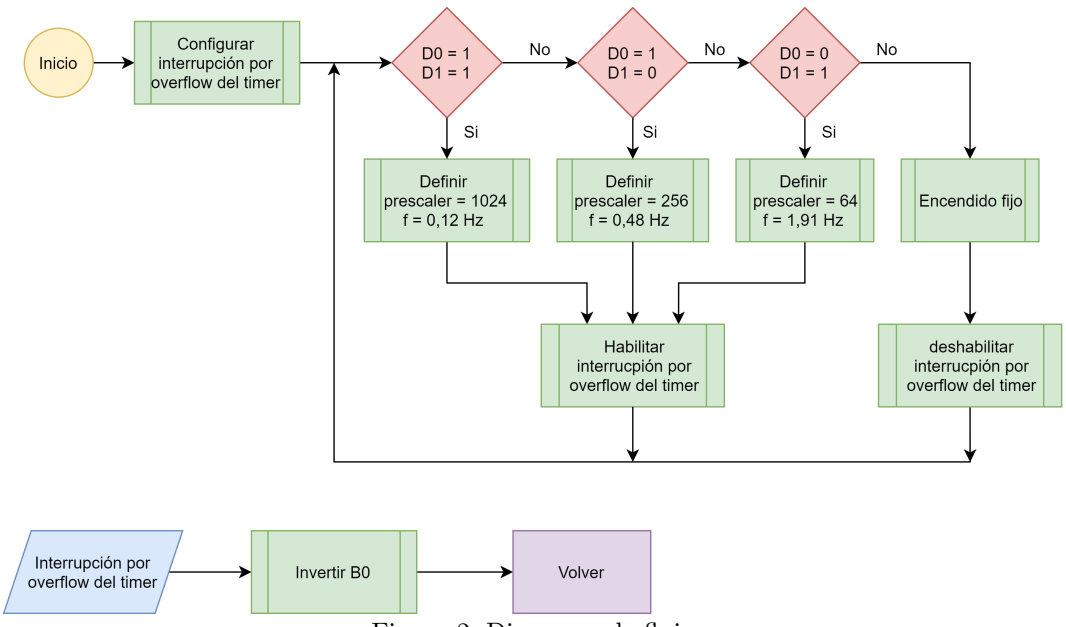

Figura 2: Diagrama de flujo

2.4. Programación en C

A continuación se describe el código en C del proyecto. Se incluyen explicaciones de las herramientas que provee C, en adición a las descripciones de los principales segmentos del código.

2.4.1. Inicialización de los puertos y del timer

Para incializar el timer 1 con interrupciones por overflow se utiliza el siguiente código:

```
int main(void)
{
  DDRB = (0x01 << LED);      //Configurar a B0 como salida, LED = PB0
  DDRD = 0x00; // Configurar al puerto D como entrada

  TCNT1 = 0; //Inicializar el timer 1 a 0

  TCCR1A = 0x00;
  TIMSK1 = (1 << TOIE1) ;    // Habilitar interrupción por overflow en el timer 1
  PORTB = 0b00000001; //Encender PB0
```

Esta configuración permitirá al timer operar de forma correcta, ya que el timer 1 operará en modo normal, y contará hasta $0xFFFF = 2^{16}$. Cada vez que ocurra overflow y el timer 1 se reinicie a 0, ocurrirá una interrupción por overflow, ya que el bit TOIE1 de TIMSK1 está en estado alto.

2.4.2. Control de la frecuencia

Como se puede visualizar a continuación, la frecuencia es controlada a través de D0 y D1. Según cuál sea la selección, se modifica el prescaler, y se logra así que el periodo se incremente o decremente.

```
while(1)
{
  aux = PIND & 0b00000011;
  if(aux == 0b00000000){
    //Encendido fijo
    cli();
    PORTB = 0b00000001;
  }else if(aux == 0b00000010){
    //Prescaler = 64
    TCCR1B = (1<<CS10) | (1<<CS11);
    sei();
  }else if(aux == 0b00000001){
    //Prescaler = 256
    TCCR1B = 1<<CS12;
    sei();
  }else{
    //Prescaler = 1024
    TCCR1B = (1<<CS10) | (1<<CS12);
```

```
    sei();
  }
}
```

En los casos en los que hay oscilación, la frecuencia puede calcularse fácilmente sabiendo que el periodo de la señal se ve incrementado en un factor de *prescaler* y que la frecuencia del clock interno del microcontrolador es 8.10^6Hz. De esta forma, el periodo de la señal cuadrada será, para cada caso:

- $prescaler = 64 \quad \Rightarrow \quad T = \frac{1}{f} = \frac{2 \cdot 2^{16} \cdot 64}{8 \cdot 10^6} = 1,05s \Rightarrow f = 0,95Hz$
- $prescaler = 256 \quad \Rightarrow \quad T = \frac{1}{f} = \frac{2 \cdot 2^{16} \cdot 256}{8 \cdot 10^6} = 4,19s \Rightarrow f = 0,24Hz$
- $prescaler = 1024 \quad \Rightarrow \quad T = \frac{1}{f} = \frac{2 \cdot 2^{16} \cdot 1024}{8 \cdot 10^6} = 16.78s \Rightarrow f = 0,06Hz$

En todos los casos el duty cycle es del 50 %.

2.4.3. Interrupción por overflow

Por último, la interrupción por overflow cambia el estado del LED en B0 en cada llamada; de esta forma se obtiene una oscilación.

```
ISR (TIMER1_OVF_vect)    // Timer1 ISR
{
  PORTB ^= (1 << LED);
  TCNT1 = 0;
}
```

2.5. Lista de materiales

Material	costo ($)
2x pulsador	26.7
4x LED	17.8
Atmega328P	504.7
Arduino nano	539.0
4 x Resistor 220 Ω	14.2
2 x Resistor 1k Ω	3.6
Total	1106.0

NOTA: Se incluyen los costos del programador. Los precios son de Abril de 2021.

2.6. Resultados

En la figura 6 se encuentra la implementación que se ha realizado del circuito explicado en este informe. No han habido inconvenientes en su implementación.

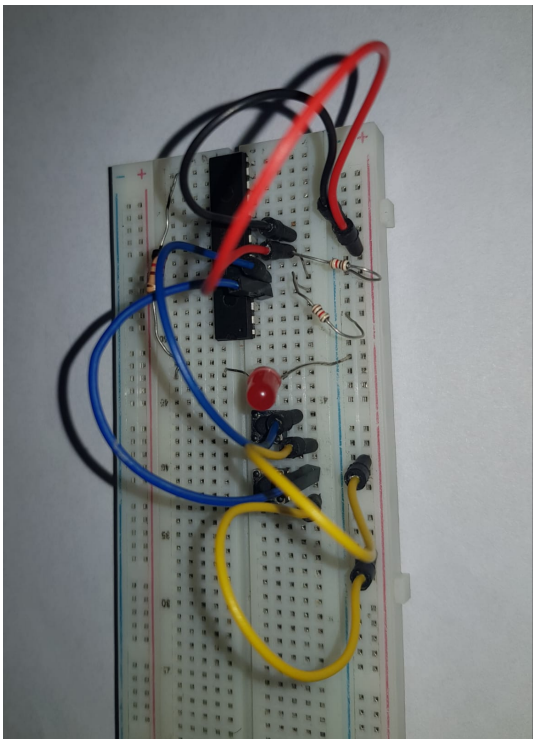

Figura 3: Circuito del proyecto

3. Control del brillo de un LED

3.1. Descripción del proyecto

En el siguiente link se encuentra la explicación del proyecto de esta sección.

- Explicación de la segunda parte

Este ejemplo funcionará a partir de dos pulsadores, los cuales controlarán el duty cycle de una señal cuadrada que será aplicada sobre un LED. Dado que al variar el duty cycle varía el valor medio de la señal, la potencia total otorgada al LED variará, y consecuentemente el brillo que se observará será variable. Claro es, la frecuencia deberá ser lo suficientemente alta como para que no se llegue a observar el apagado del LED en cada periodo.

3.2. Circuito esquemático

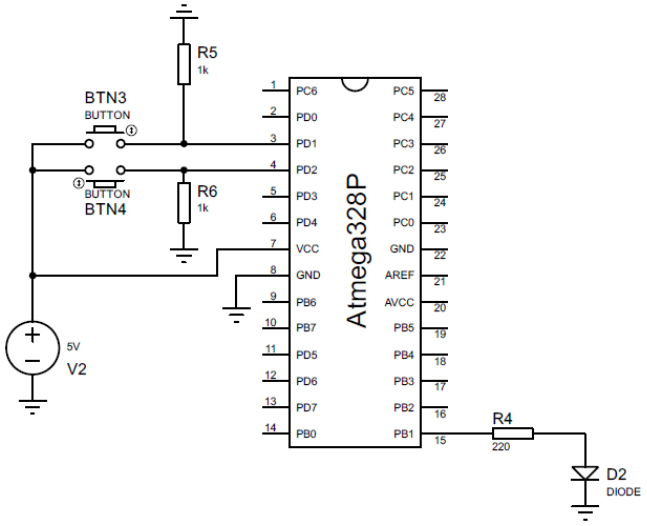

Figura 4: Circuito a implementar

3.3. Diagrama de flujo

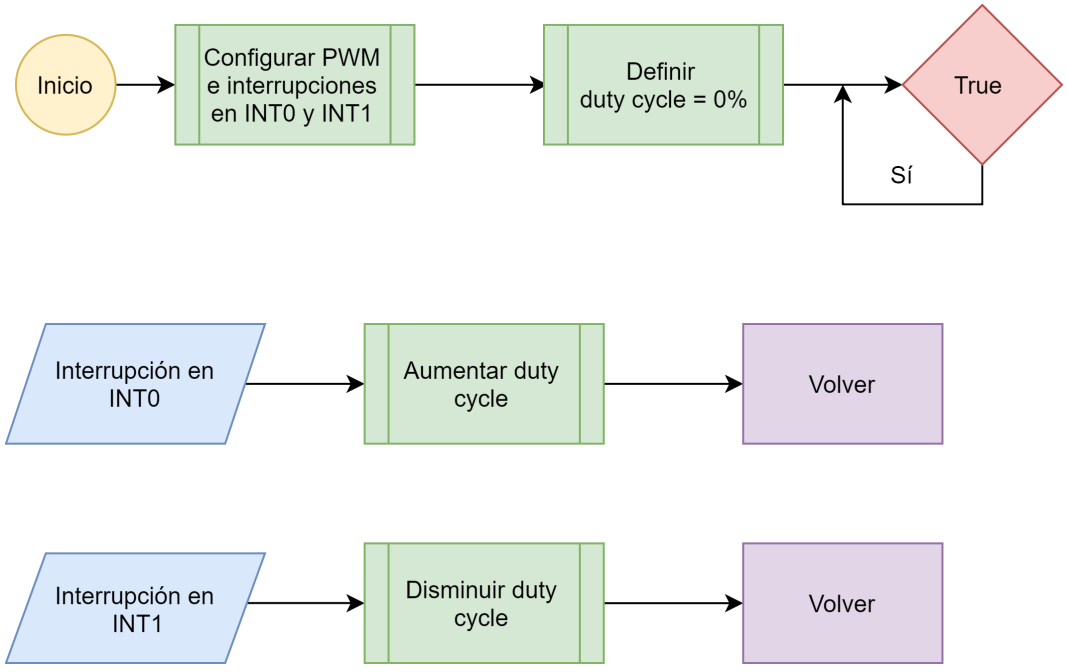

Figura 5: Diagrama de flujo

3.4. Programación en C

A continuación se describe el código en C del proyecto. Se incluyen explicaciones de las herramientas que provee C, en adición a las descripciones de los principales segmentos del código.

3.4.1. Inicialización del PWM

Para incializar al PWM en modo fast PWM se utiliza el siguiente código:

```
// Modo: Fast PWM, top=0xFF
// OC0A: PWM no invertido
// Periodo del timer: 0.032 ms
TCCR1A = (1<<COM1A1) | (1<<WGM11) | (1<<WGM10);
TCCR1B = (1<<WGM12) | (1<<CS10);
TCNT1=0x00;
OCR1A=0x0;
```

Aquí, COM1A1 permite que el puerto de OC1A tome control del puerto correspondiente en el terminal correspondiente del mictrocontrolador. WGM10, WGM11 y WGM12 permiten definir el modo de operación de los timers. CS10, CS11 y CS12 permiten definir al clock, que en este caso no tendrá prescaler aplicado. Por último, OCR1A define cuando cambiará de estado la señal de PWM; en este caso, dado que el timer es de 10 bits, OCR1A será un número entre 0 y 0x03FF.

3.4.2. Inicialización de los puertos

El siguiente código permite inicializar los puertos, estableciendo así interrupciones en INT0 e INT1.

```
  DDRB = 0b00000010; //B1 como salida
DDRD = 0; //D2 y D3 como entradas
EIMSK = (1<<INT0) | (1<<INT1);
EICRA = (1<<ISC00) | (1<<ISC01) | (1<<ISC10) | (1<<ISC11);
sei();
```

3.4.3. Control del brillo

Como se puede visualizar a continuación, el duty cycle es controlado a través de D0 y D1, mediante interrupciones.

```
ISR (INT0_vect)
{
  unsigned char sreg = SREG;

  if(OCR1A < 500){
    OCR1A += 50;
  }
  _delay_ms(10);
```

```
  SREG = sreg;
}

ISR (INT1_vect)
{
  unsigned char sreg = SREG;

  if(OCR1A >49){
    OCR1A -= 50;
  }
  _delay_ms(10);

  SREG = sreg;
}
```

De forma empírica se ha encontrado que la variación de brillo del led se visualiza de mejor forma cuando OCR1A varía entre 0 y 500, con saltos de 50 unidades.

3.5. Lista de materiales

Material	costo ($)
2x pulsador	26.7
4x LED	17.8
Atmega328P	504.7
Arduino nano	539.0
4 x Resistor 220 Ω	14.2
2 x Resistor 1k Ω	3.6
Total	1106.0

NOTA: Se incluyen los costos del programador. Los precios son de Abril de 2021.

3.6. Resultados

En la figura 6 se encuentra la implementación que se ha realizado del circuito explicado en este informe. No han habido inconvenientes en su implementación.

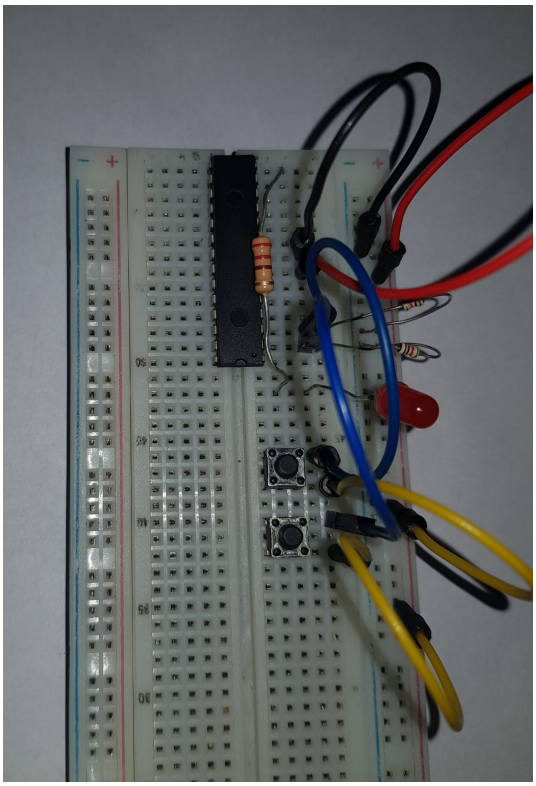

Figura 6: Circuito del proyecto

4. Conclusión

En este informe se ha estudiado la utilización de los timers del microcontrolador y de las diferentes posibilidades que estos proveen, incluyendo la herramienta de PWM. Para ello, se han estudiado dos proyectos que dejan en evidencia estas utilidades de los microcontroladores en forma simple y concisa.

5. Código Fuente

5.1. Código de la primer parte

```
#include <avr/io.h>
#include <avr/interrupt.h>

#define LED PB0

ISR (TIMER1_OVF_vect)    // Timer1 ISR
{
  PORTB ^= (1 << LED);
  TCNT1 = 0;
}
```

```
int main(void)
{
  DDRB = (0x01 << LED);      // Configurar a B0 como salida
  DDRD = 0x00; // Configurar al puerto D como entrada

  TCNT1 = 0; // Inicializar el timer 1 a 0

  TCCR1A = 0x00;
  TIMSK1 = (1 << TOIE1) ;    // Habilitar interrupción por overflow en el timer 1
  PORTB = 0b00000001; // Encender PB0
  unsigned char aux;

  while(1)
  {
    aux = PIND & 0b00000011;
    if(aux == 0b00000000){
      //Encendido fijo
      cli();
      PORTB = 0b00000001;
    }else if(aux == 0b00000010){
      //Prescaler = 64
      TCCR1B = (1<<CS10) | (1<<CS11);
      sei();
    }else if(aux == 0b00000001){
      //Prescaler = 256
      TCCR1B = 1<<CS12;
      sei();
    }else{
      //Prescaler = 1024
      TCCR1B = (1<<CS10) | (1<<CS12);
      sei();
    }
  }
}
```

5.2. Código de la segunda parte

```
#include <avr/io.h>
#define F_CPU 8000000UL
#include <util/delay.h>
#include <avr/interrupt.h>

int main(void){
  DDRB = 0b00000010; //B1 como salida
  DDRD = 0; //D2 y D3 como entradas
  // Modo: Fast PWM, top=0xFF
  // OC0A: PWM no invertido
  // Periodo del timer: 0.032 ms
  TCCR1A = (1<<COM1A1) | (1<<WGM11) | (1<<WGM10);
  TCCR1B = (1<<WGM12) | (1<<CS10);
  TCNT1=0x00;
  OCR1A=0x0;
```

```
  EIMSK = (1<<INT0) | (1<<INT1);
  EICRA = (1<<ISC00) | (1<<ISC01) | (1<<ISC10) | (1<<ISC11);
  sei();

  while(1){}
}

ISR (INT0_vect)
{
  unsigned char sreg = SREG;

  if(OCR1A < 500){
    OCR1A += 50;
  }
  _delay_ms(10);

  SREG = sreg;
}

ISR (INT1_vect)
{
  unsigned char sreg = SREG;

  if(OCR1A >49){
    OCR1A -= 50;
  }
  _delay_ms(10);

  SREG = sreg;
}
```