Universidad de Buenos Aires

Facultad de Ingeniería

2do Cuatrimestre 2020

[86.07] Laboratorio de Microprocesadores

Trabajo Práctico Integrador

Alumno: Agustín Miguel Flouret

Padrón: 102298

Turno: Martes

Docentes:

- Gerardo Stola
- Guido Salaya
- Fernando Cofman

Fecha de entrega: 2 de marzo de 2021

Índice

1. Objetivo del proyecto

El objetivo del proyecto es realizar un programa que permita analizar la carga y descarga de un capacitor en un circuito RC utilizando el microcontrolador.

2. Descripción del proyecto

Para realizar el proyecto se utilizará la placa Arduino UNO, que incluye un microcontrolador ATmega328P. La programación del microcontrolador se realizará a través del Arduino, utilizando el software AVRDUDE. El lenguaje de programación que se usará es C, y se compilará con Atmel Studio. El hardware externo se detallará en la sección siguiente.

Las tareas a realizar son las siguientes:

- Generar con Fast PWM una señal de 50 Hz con un duty cycle de 50 %.
- Verificar la frecuencia generada con el modo captura
- Armar un circuito RC, calcular los valores R y C para lograr un tau que sea 3 veces menor que el periodo de la señal de entrada
- Adquirir con el ADC valores de voltaje sobre el capacitor
- Transmitir por via serial los valores obtenidos a una PC
- Graficar los resultados obtenidos

3. Listado de componentes y gastos

- 1 Arduino UNO - $1009
- 1 protoboard - $300
- 1 capacitor 100 nF - $30
- 1 resistor $56k\Omega$ - $7
- **Gasto total: $1346**

4. Circuito esquemático de Arduino UNO

En la figura 1 se muestra el circuito esquemático completo de la placa Arduino UNO que se usará en este trabajo. Para poder observar con mayor detalle las conexiones con otros componentes, en las secciones que siguen se mostrarán solo algunas partes del esquemático en forma ampliada.

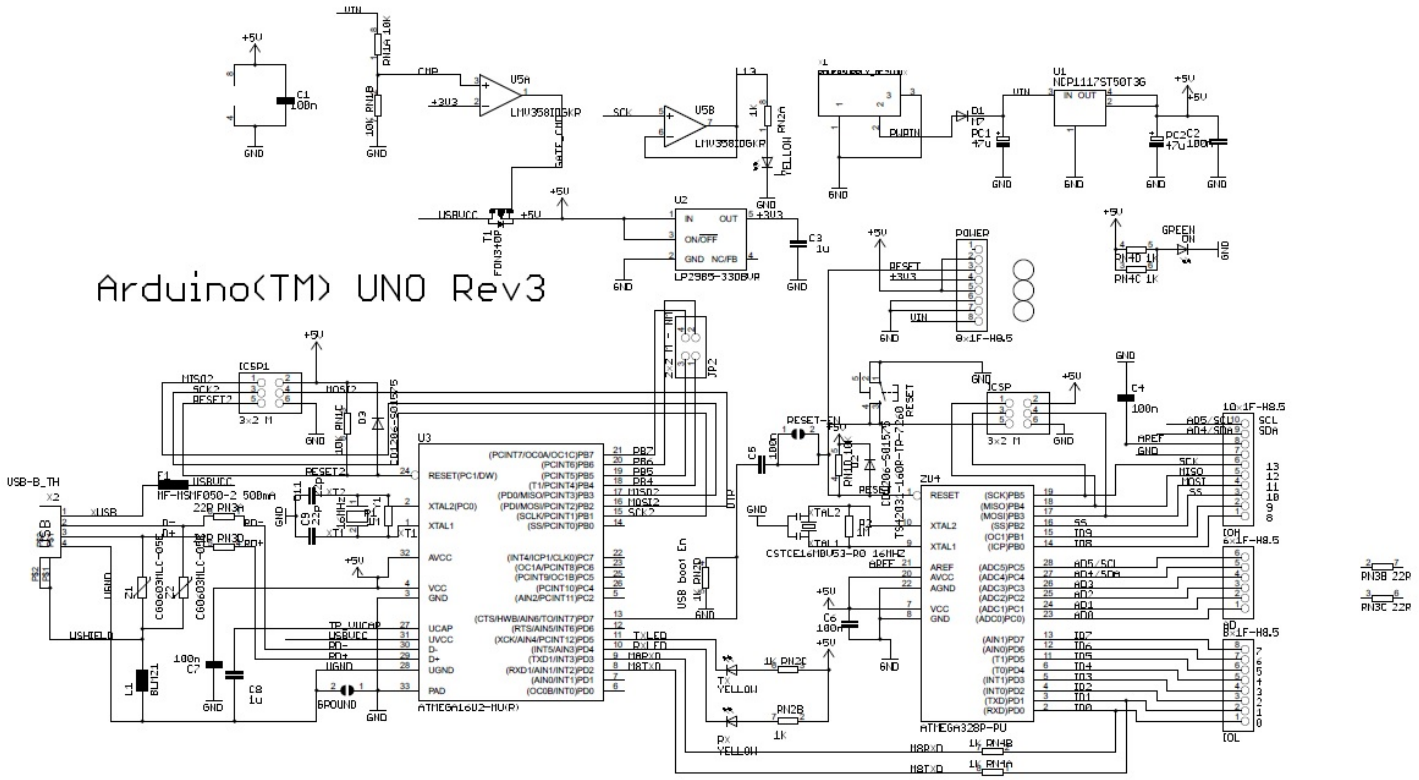

Figura 1: Circuito esquemático de la placa Arduino UNO.

5. Desarrollo del proyecto

En este proyecto se generó una señal cuadrada de frecuencia 61 Hz (período 16.4 ms), que es la mínima frecuencia que puede generar el PWM del Timer0 en este microcontrolador. Para obtener un tau 3 veces menor que el periodo se utilizó un capacitor de $0{,}1\mu F$ y un resistor de $56k\Omega$, resultando en $\tau = 5{,}6ms$.

Para medir el periodo de la señal generada, se conectó la salida del PWM al pin PB0, que corresponde a la entrada de captura del Timer 1. Por otro lado, se midió la tensión sobre el capacitor mediante el ADC, utilizando el pin PC2. La conexión con la computadora por via serial se realizó a través de la salida USB del Arduino UNO. A continuación se muestra el diagrama de conexiones en bloques y el circuito esquemático del proyecto.

5.1. Diagrama de conexiones en bloques

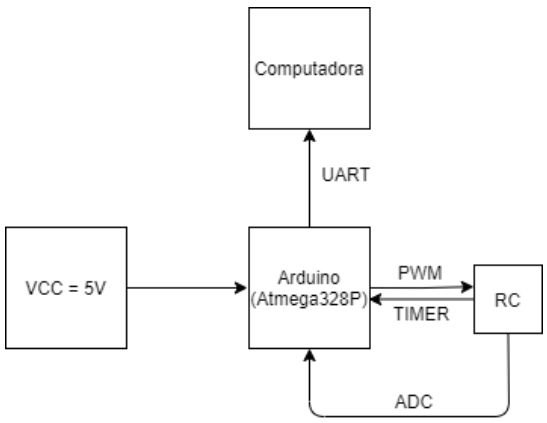

Figura 2: Diagrama de bloques del proyecto.

5.2. Circuito esquemático

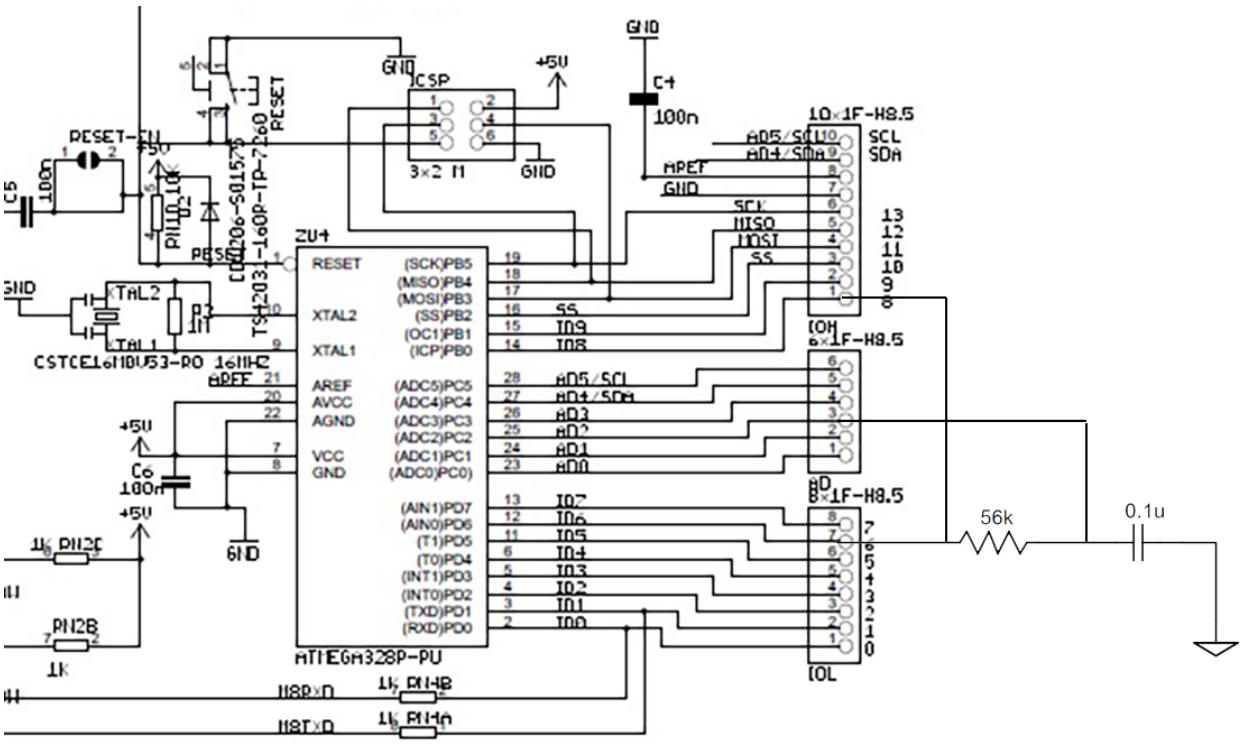

Figura 3: Circuito esquemático que muestra las conexiones con la red RC.

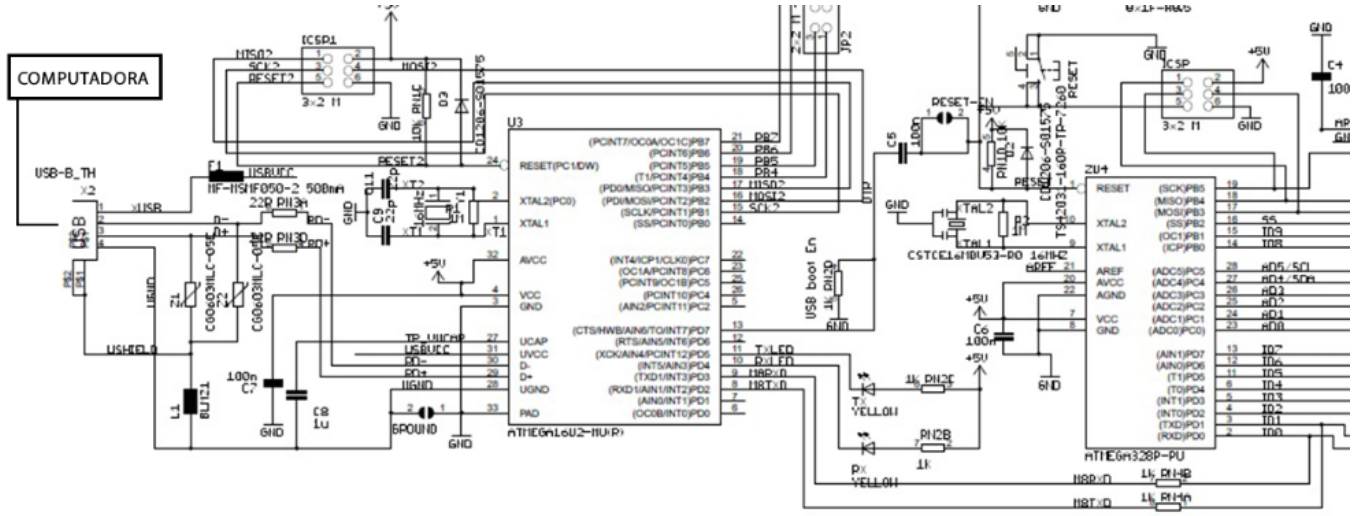

Figura 4: Circuito esquemático que muestra la conexión con la computadora por USB.

5.3. Software

El proyecto está organizado de la siguiente manera: se implementaron las funciones para inicializar y controlar los periféricos en un archivo .c y un archivo .h por cada periférico. Estas funciones son utilizadas en el archivo main.c, donde se encuentra la función main. En la función main se inicializan los periféricos y se realiza una medición del periodo de la señal, transmitiendo el valor por el puerto serie. Luego comienza el loop principal, donde se realiza repetidamente la lectura del ADC y su posterior transmisión por el puerto serie.

Se decidió transmitir el valor de 10 bits del ADC utilizando la función sprintf para convertir a string y transmitiendo cada caracter por el puerto serie. Al ser un número de 10 bits, el número máximo de dígitos decimales es 4 (el valor máximo es 1023). Por lo tanto, por cada lectura del ADC se transmitieron 4 caracteres y un caracter de separación. En total, son 5 caracteres por cada medición del ADC.

Teniendo en cuenta que la frecuencia del clock del Arduino es 16 MHz, y se configura el ADC con prescaler 128, la frecuencia del clock del ADC es 125 kHz. Como cada medición del ADC tiene una duración de 13 ciclos de clock, la frecuencia de muestreo máxima resulta ser aproximadamente 9600 Hz. Sin embargo, está limitada por el baudrate del puerto serie.

El baudrate utilizado es 57600. En la transmisión de cada caracter se utilizan 10 bits (8 bits, 1 bit de stop y 1 bit de start). Por lo tanto, el puerto serie puede transmitir 5760 caracteres por segundo. Como la cantidad de caracteres por muestra es 5, la frecuencia con que se transmite cada muestra es entonces 1152 muestras por segundo.

Esta frecuencia de muestreo permite, según el teorema de Nyquist, muestrear correctamente la frecuencia fundamental de 60Hz y los primeros armónicos impares (180Hz, 300Hz, 420Hz y 540Hz), con lo cual se obtiene una buena reconstrucción de la señal original. En la sección de resultados se muestra un gráfico de la señal obtenida.

5.3.1. Diagrama de flujo

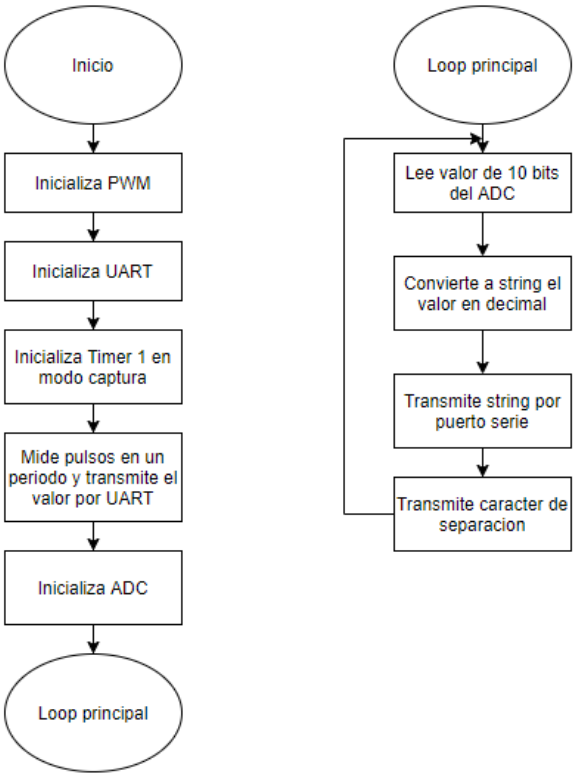

Figura 5: Diagrama de flujo.

5.3.2. Código del programa

main.c

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

```
#include "uart.h"
#include "timer.h"
#include "adc.h"
#include "pwm.h"
#define BAUD 57600


int main(void)
{
        pwm_init();
        uart_init(57600);
        timer_capture_init();

        uint16_t capture_value = timer_measure_period();
        char capture_value_str[6];
        sprintf(capture_value_str, "Medicion modo captura: %05u\n", capture_value);
        uart_send_string(capture_value_str);

        uart_send_string("Valores ADC:\n");

        adc_init();

    while (1)
        {
                int adc_value = adc_read();

                char adc_value_str[5];
                sprintf(adc_value_str, "%04d", adc_value);

                uart_send_string(adc_value_str);

                uart_send_character('\n');
        }
}
```

uart.h

```
#ifndef _UART_H_
#define _UART_H_

void uart_init(uint16_t);
void uart_send_character(char);
void uart_send_string(char*);

#endif
```

uart.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "uart.h"
#define F_CLOCK  16000000

void uart_init(uint16_t baudrate)
{
        //Habilita transmision de datos
        UCSR0B = (1<<TXEN0);
        //Configuracion 8N1: 8 bits de datos, sin bit de paridad y 1 bit de stop
        UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);

        UBRR0L = (F_CLOCK/16/baudrate)-1;
        UBRR0H = ((F_CLOCK/16/baudrate)-1)>>8;
```

```
}

void uart_send_character(char c)
{
        while(!(UCSR0A & (1<<UDRE0)));
        UDR0 = c;
}

void uart_send_string(char* str)
{
        for(int i=0; str[i]!='\0'; i++)
        uart_send_character(str[i]);
}
```

timer.h

```
#ifndef _TIMER_H_
#define _TIMER_H_

void timer_capture_init(void);
uint16_t timer_measure_period(void);

#endif
```

timer.c

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

void timer_capture_init(void)
{
        TCCR1A = 0;
        TCCR1B = (1<<ICES1)|(1<<CS10)|(1<<CS12);//flanco ascendente, 1024 prescaler, sin cancelacion de

}

uint16_t timer_measure_period(void)
{
        uint16_t capture_value;
        while(!(TIFR1&(1<<ICF1)));
        capture_value = ICR1;
        TIFR1 = 1<<ICF1;
        while(!(TIFR1&(1<<ICF1)));
        return ICR1 - capture_value;
}
```

adc.h

```
#ifndef _ADC_H_
#define _ADC_H_

void adc_init(void);
uint16_t adc_read(void);

#endif
```

adc.c

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

void adc_init(void)
```

```
{
        // Configura ADC con entrada simple en PC2 y prescaler CLK/128
        ADMUX = (1<<REFS0)|(1<<MUX1);
        ADCSRA = (1<<ADEN)|(1<<ADSC)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}

uint16_t adc_read(void)
{
        ADCSRA |= (1<<ADSC);
        while(!(ADCSRA&(1<<ADIF)));
        return ADC;
}
```

pwm.h

```
#ifndef _PWM_H_
#define _PWM_H_

void pwm_init(void);

#endif
```

pwm.c

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

void pwm_init(void)
{
        //Configura PD6 (OC0A) como salida
        DDRD = 1<<PD6;
        //Configura PWM en modo Fast PWM non-inverted
        TCCR0A = (1<<COM0A1)|(1<<WGM01)|(1<<WGM00);
        //Prescaler: clk/1024 -> Frecuencia: 16M/256/1024 = 61.035 Hz
        TCCR0B = (1<<CS02)|(1<<CS00);
        //Duty cycle: 50%
        OCR0A = 128;
}
```

6. Resultados

En la siguiente figura se muestran los resultados de una medición de 71 muestras. Se puede observar claramente la forma de onda característica de la carga y descarga de un capacitor en un circuito RC. El valor mínimo (0) representa 0 V, y el máximo (1023) representa 5 V.

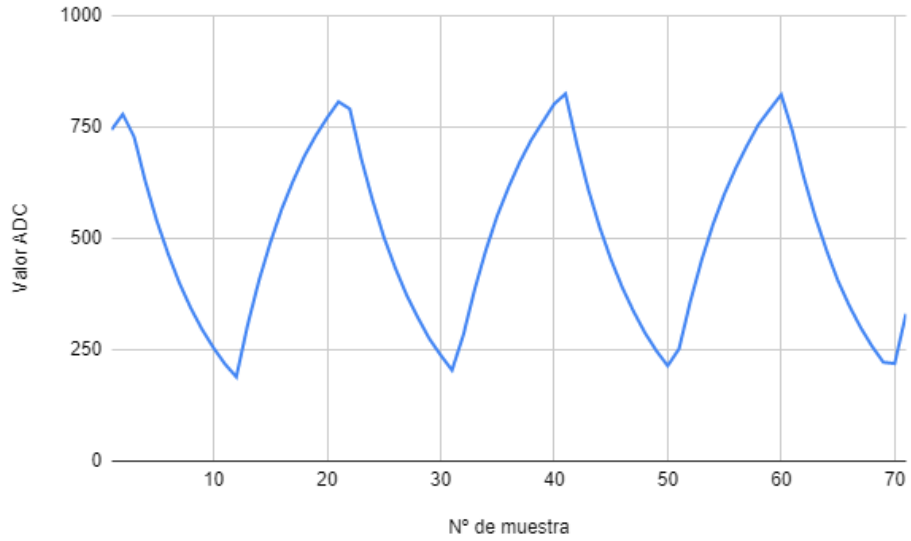

Figura 6: Gráfico de la medición realizada.

Finalmente, se muestra una captura del terminal serie de Arduino, donde se puede ver la medición inicial del período de la señal, así como los valores del ADC. El valor de la medición del timer en modo captura es 256. Dado que se utilizó un prescaler de 1024, esto significa que el contador contó $256 \cdot 1024 = 262144$ pulsos de clock en un período. Para una frecuencia de clock de 16 MHz (período 62,5 ns), la duración del período de la señal es 16,4 ms. Por lo tanto, la frecuencia medida de la señal es 61 Hz, lo cual coincide con la frecuencia real.

```
Medicion modo captura: 00256
Valores ADC:
0782
0813
0757
0652
0563
0485
0417
0359
0309
0266
0230
0199
0313
```

Figura 7: Captura del terminal serie.

7. Conclusiones

En este trabajo se logró implementar con éxito lo requerido. Se pudo utilizar en conjunto todos los periféricos estudiados en los trabajos prácticos anteriores, para implementar un programa que permite medir tensión y transmitir los resultados a la computadora. Si, además, en el futuro se programara un script para graficar los datos recibidos en tiempo real, este programa funcionaría como un osciloscopio para señales de bajas frecuencias.

Se observaron las limitaciones que tiene el microcontrolador para transmitir datos por el puerto serie, lo cual limita la frecuencia de muestreo que se puede obtener en este tipo de aplicaciones.