FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Laboratorio de Microprocesadores - 86.07

Trabajo Práctico N°1:

Parpadeo de un LED

Profesor:			Ing. Guillermo Campiglio							
Cuatrimestre/Año:			1°/2020							
Turno de las clases prácticas			Miércoles							
Jefe de trabajos prácticos:			Ing. Pedro Ignacio Martos							
Docente guía:			-							
Autor			Seguimiento del proyecto							
Francisco	Rossi	99540								

Observaciones:

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

Fecha de aprobación				Firma J.T.P

Coloquio	
Nota final	
Firma profesor	

Índice

1. Introducción

En el siguiente informe se explica el diseño de dos programas escritos en lenguaje Assembler para hacer parpadear un LED conectado al pin *PB0* de un *ATMEGA328p* de dos maneras diferentes:

a. Utilizando todo el puerto.

b. Utilizando solo un bit del puerto.

1.1. Objetivo

El objetivo es controlar un pin de entrada/salida del microcontrolador.

1.2. Descripción

Se conecta un LED con un resistor en serie al pin *PB0*, para controlar el encendido y apagado del mismo mediante las instrucciones de Assembler correspondientes a los casos (a) y (b), en los cuales se controlara en un caso todo el puerto y en el otro un único pin respectivamente. También se realiza el ajuste de la frecuencia de oscilación a partir de pequeños cambios en el código.

2. Materiales

Se utilizaron los siguiente materiales para el proyecto:

a. 1 LED Blanco (20$ (Pesos Argentinos))

b. 1 Resistor de 220Ω (4$ (Pesos Argentinos))

c. 1 Microcontrolador ATmega328p (Utilizando el integrado con el Arduino Uno) (700$ (Pesos Argentinos))

3. Diagrama en Bloques

En la **Fig. 1** se muestra un diagrama en bloques del circuito.

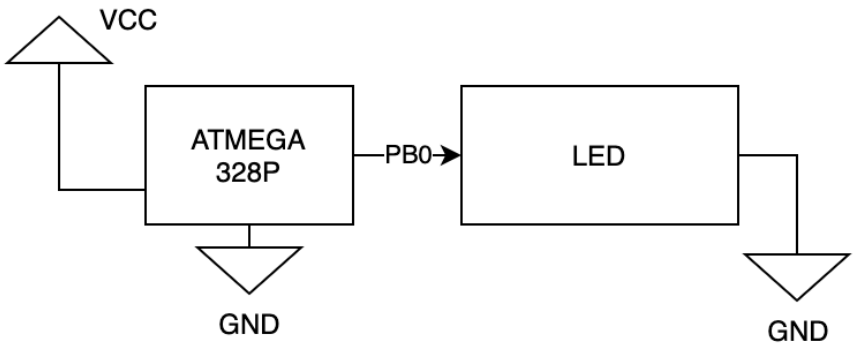

Figura 1: Diagrama en bloques.

4. Esquemático

En las **Fig. 3** y **Fig. 2** se muestra como se conectó el arduino con el LED y el resistor de 220 Ω.

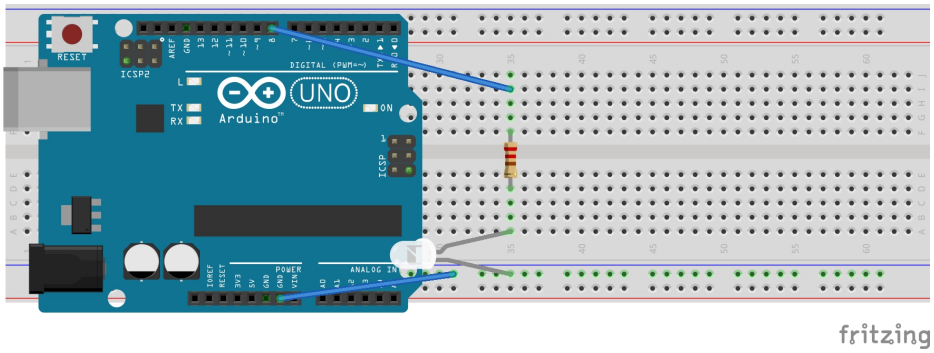

Figura 2: Conección entre arduino y LED.

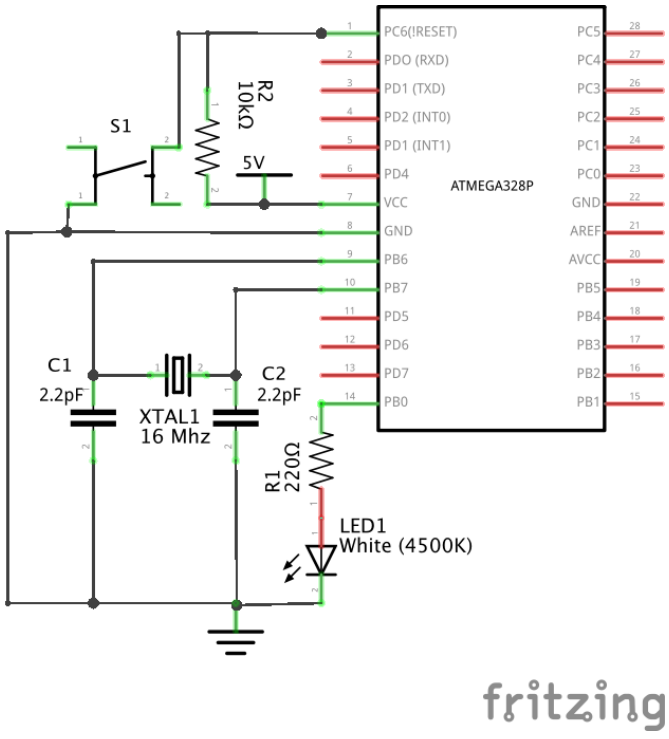

Figura 3: Esquemático del circuito implementado.

5. Diagrama de flujo

En la **Fig. 4** se muestra el diagrama de flujo del programa (b), el caso (a) es identico pero $DDRB = 0xFF$ y LED on representa a todo el puerto B.

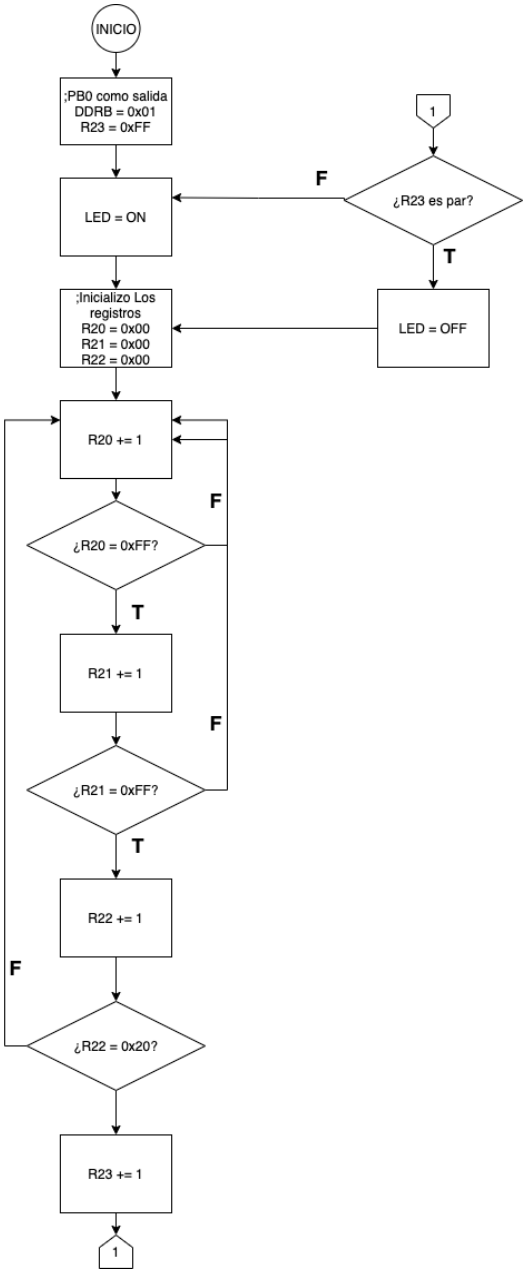

Figura 4: Diagrama de flujo.

6. Códigos

Ambos códigos comparten la mayoría de sus lineas, la diferencia entre ambos es que en (a) se configura todo el *PortB* como salida y se hace conmutar el mismo todo como uno, en este caso podemos conectar el LED a cualquier *PIN* del puerto y este parpadearía. En cambio, en (b) se configura únicamente el *PB0* como salida y se hace conmutar el mismo, por lo tanto, solo conmutaría este *PIN*.

6.1. a) Utilizando todo el puerto.

```
.include "m328pdef.inc"

.cseg
.org 0x0000
      jmp main

.org INT_VECTORS_SIZE


main:

      ldi   r20,0xff
      out   DDRB,r20     ; Configuro puerto B como salida

      ser   r25        ; valores 0xFF y 0x00 para luego conmutar el puerto B
      clr   r26

      ldi   r23,0xff
on:     out   PORTB,r25      ;enciendo TODO el PUERTO B.


delay:
      ldi   r20,0x00
      ldi   r21,0x00
      ldi   r22,0x00

cycle:    inc   r20
      cpi   r20,0xff
      brlo  cycle
      ldi   r20,0x00
      inc   r21
      cpi   r21,0xff
      brlo  cycle
      ldi   r21,0x00
      inc   r22
      cpi   r22,0x20     ; con este valor es facil variar de forma apreciable la frecuencia
                              ;ya que es un multiplicador de todos los incrementos anteriores
      brlo  cycle
      inc   r23

      mov   r24, r23
      andi  r24, 0x01 ; el resultado es 0 o 1, y define si r23 es o no par.
      breq  off      ; Si es el valor de r23 es par entonces que apague el led,
                          ;si no que lo prenda, asi cada vez que incrementa r23 tiene
                          ;un resultado diferente, hace 1 ciclo con LED on y otro con LED off
      RJMP  on       ; reinicio el ciclo

off:    out   PORTB,r26      ;apago el PUERTO B entero
      RJMP  delay
```

6.2. b) Utilizando solo un bit del puerto.

```
..include "m328pdef.inc"

.equ lsb = 0x01

.cseg
.org 0x0000
      jmp main

.org INT_VECTORS_SIZE

main:

      ldi   r20,lsb
      out   DDRB,r20     ; Configuro el PB,0 como salida. No todo el puerto.

      ldi   r23,0xff     ; esto es para incializar r23 en un valor que al incrementarlo
                              ;sea par y cumplir la condicion en el primer breq que debe ser True
on:     sbi   PORTB,0      ;enciendo el LED en PB0.


delay:
      ldi   r20,0x00
      ldi   r21,0x00
      ldi   r22,0x00

cycle:    inc   r20
      cpi   r20,0xff
      brlo  cycle
      ldi   r20,0x00
      inc   r21
      cpi   r21,0xff
      brlo  cycle
      ldi   r21,0x00
      inc   r22
      cpi   r22,0x20     ; con este valor es facil variar de forma apreciable la frecuencia
                              ;ya que es un multiplicador de todos los incrementos anteriores
      brlo  cycle
      inc   r23

      mov   r24, r23
      andi  r24, 0x01 ; el resultado es 0 o 1, y define si r23 es o no par.
      breq  off          ; Si es el valor de r23 es par entonces que apague el led,
                           ;si no que lo prenda, asi cada vez que incrementa r23 tiene
                           ;un resultado diferente, hace 1 ciclo con LED on y otro con LED off
      RJMP  on       ; reinicio el ciclo

off:    cbi   PORTB,0      ;apago el LED
      RJMP  delay      ; reinicio el ciclo
```

El ciclo para realizar los retardos esta basado en incrementos de registros, dependiendo del valor de R23 se determina si el LED se debe apagar o prender. Y se repite el ciclo infinitamente. En la siguiente sección se explicará en profundidad la relación entre el incremento de registro y los tiempos de conmutación del PIN o el Puerto. El código fue pensado para lograr un tiempo en alto y en bajo similares es decir, que el *Duty Cycle* sea lo más cercano posible al 50 %.

6.3. Frecuencia de oscilación del LED

En ambos casos, queremos hacer que la oscilación sea apreciable a simple vista, es decir setear el tiempo de encendido y apagado del LED, lo que es lo mismo que el tiempo de encendido y apagado del *PB0*.

La frecuencia por default del clock del *Arduino UNO* es $f_{ck} = 16\ MHz$, de manera que sabiendo los ciclos de maquina que hay entre encendido y apagado $ciclos_{on}$, $ciclos_{off}$ podemos obtener los tiempos de encendido y apagado como $t_{on} = \frac{ciclos_{on}}{f_{ck}}$, $t_{off} = \frac{ciclos_{off}}{f_{ck}}$ y con esto podemos obtener la frecuencia de oscilación del LED como $f = \frac{1}{t_{on}+t_{off}}$.

Luego, para calcular la cantidad de ciclos en alto y en bajo debemos analizar como vamos a realizar el ciclo en el código, en este caso se realizó incrementando registro de manera que cada registro tiene un *peso*, al igual que el sistema numérico de manera que cuando R20 sea incrementado 255 veces el registro R21 se incrementara una vez y se volverá a repetir el incremento entre 0 y 255 del valor de R20, lo mismo sucederá con R22 y R21 hasta el valor máximo de R22, en este caso se eligió $R22_{MAX} = 0x20 \underbrace{=}_{\text{a decimal}} 32$.

De manera que la cantidad de ciclos totales se podrá calcular como la cantidad de ciclos que se toma en incrementar el registro R20 (CC_{R20}) sumado a la cantidad de ciclos que se toma en incrementar el regitro R21 (CC_{R21}) sumado a la cantidad de ciclos que se toma en incrementar el registro R22 (CC_{R22}) sumado a los ciclos extras.

Teniendo en cuenta la **Tabla. 1** con la cantidad de ciclos que toma cada instrucción.

Tabla 1: Ciclos de maquina por instrucción

Instrucción	Ciclos
SBI	2
CBI	2
LDI	1
INC	1
CPI	1
BRLO	1
MOV	1
ANDI	1
BREQ_FALSE	1
BREQ_TRUE	2
BRLO_FALSE	1
BRLO_TRUE	2
RJMP	2

Podemos representar lo dicho anteriormente de manera matemática como:

$$CC_{R20} = \Big[255 \cdot [cy(INC) + cy(CPI) + cy(BRLO_T)] - cy(BRLO_T) + cy(BRLO_F)\Big] \cdot 255 \cdot R22_{MAX}$$

$$CC_{R21} = \Big[255 \cdot [cy(INC) + cy(CPI) + cy(BRLO_T) + cy(LDI)] - cy(BRLO_T) + cy(BRLO_F)\Big] \cdot R22_{MAX}$$

$$CC_{R22} = [R22_{MAX} \cdot (cy(INC) + cy(CPI) + cy(BRLO_T))] - cy(BRLO_T) + cy(BRLO_F)$$

Obteniendo luego la cantidad de ciclos en los cuales el LED estará encendido y apagado, ordenados por el orden de aparición:

$$cy_{on} = 3cy(LDI) + CC_{R21} + CC_{R22} + cy(MOV) + cy(ANDI) + cy(BREQ_T) + cy(CBI) + CC_{R20} = 8355976$$

$$cy_{off} = 3cy(LDI) + CC_{R20} + CC_{R21} + CC_{R22} + cy(MOV) + cy(ANDI) + cy(BREQ_F) + cy(RJMP) + cy(SBI) = 8355977$$

Luego utilizando la expresión $t_x = \frac{ciclos_x}{f-ck}$ se obtienen lo valores de t_{on} y t_{off}.

Los valores obtenidos se muestran en la **Tabla. 2**.

Tabla 2: Tiempos de encendido y apagado.

	Ciclos	Tiempo (s)
ON	8355976	0,52225
OFF	8355977	0,52225

Con estos resultados obtenemos la frecuencia de oscilación del LED y el *Duty Cycle*:

$$\boxed{Duty\ Cycle = 50\,\%}$$

$$\boxed{f = 0,95740\ Hz}$$

7. Resultados

Se logró diseñar dos programas para el microcontrolador ATMEGA328p que hagan oscilar un LED colocado en el puerto B, con *Duty Cycle* $\approx 50\,\%$ y con frecuencia adaptable cambiando el valor de $R22_{MAX}$ de dos maneras diferentes, una utilizando todo el puerto y otra un único PIN.

8. Conclusiones

Se logró controlar el pin de entrada y salida para hacer titilar el LED tanto individualmente (b), asi como todo el puerto junto (b), podemos encontrar la posible utilidad de los dos casos, por ejemplo en el caso (a) podemos sincronizar todo el puerto para que conmute al mismo tiempo en caso, por ejemplo, de querer conmutar múltiples LEDs de manera sincronizada sin que la corriente máxima que puede ser obtenida del pin sea alcanzada. En el otro caso es de utilidad cuando en los otros pines hay otros periféricos conectamos y no deseamos alterar sus valores.