Facultad de Ingeniería
Universidad de Buenos Aires

Laboratorio de Microprocesadores - 86.07

Trabajo Práctico Obligatorio N°1

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									
Mariano Federico	Guglieri	99573										

Observaciones:

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

Fecha de aprobación				Firma J.T.P

Coloquio	
Nota final	
Firma profesor	

Índice

1. Objetivos

El objetivo de este trabajo consiste en crear un programa, mediante el uso de registros de timers, que haga parpadear un led en 3 frecuencias distintas o que lo deje encendido fijo, según los valores que haya en las entradas de un puerto.

2. Desarrollo

Para la realización del trabajo se utilizó la plataforma de desarrollo de Atmel, Atmel Studio versión 7.0, en donde se implementó el Software. Para la parte física se requirió de un Arduino UNO, el cual sirvió como programador para el integrado ATMEGA328P, además de suplirlo con la energía necesaria para su funcionamiento. Se utilizó un resistor junto a un led de color verde conectado al puerto b, dos pulsadores conectados al pind0 y pind1, dos resistores de 10kΩ y varios cables para unir el circuito.

En la primera parte del código se declararon las direcciones de la sección de configuración, de la interrupción del timer y del programa principal. En la parte de configuración se configuró el Stack Pointer, los pines de entrada y salida y se habilitó la interrupción por overflow del timer1. El objetivo del programa principal es leer el estado de los pines del puerto d y luego derivar esta información a una subrutina la cual, según en que estado estén los pines, modifica el registro de control del timer1. Si no se presionan los pines, el led permanecerá encendido. Si se presiona sólo el pind0, la interrupción del timer funcionará contando los pulsos de clock dividido por un prescaler de valor igual a 64. Si se presiona el pind1 el prescaler será de 256 y si se presionan los dos al mismo tiempo el prescaler será de 1024.

Se optó por usar la técnica de polling para leer el estado del puerto d, dado que el programa no tiene que ejecutar otras tareas, no interrumpe nada. Haciendo polling se puede enganchar un rebote o un ruido y por lo tanto se podría llegar a configurar mal es preescaler. Si bien la próxima medición puede llegar a corregir el problema, no se debería actualizar el prescaler hasta validar que el dato esta fijo. Es por esto que se hacen 2 lecturas con un delay en el medio y se comparan si ambas fueron iguales.

2.1. Configuración del Timer1

En la sección de configuración, como se mencionó previamente, se configuró el resgistro del timer1 para que se genere la interrupción cuando este produzca un desborde (overflow). Esto se hace poniendo en alto el bit TOIE1 del registro TIMSK1. La dirección de la interrupción por overflow del timer1 se encuentra bajo el nombre OVF1addr en el archivo m328pdef.inc. El comportamiento del timer1 se especifica mediante los registros de control TCCR1A y TCCR1B. Como se opera en modo normal el registro TCCR1A no se utilizó ya que se encuentra de manera predeterminada en cero. El registro TCCR1B se encarga de configurar, entre otras cosas, el uso de prescalers y la detención de la interrupción del timer. Cada vez que se modifica el estado de los pines del puerto de entrada, se debe modificar este registro acorde a lo estipulado por el trabajo.

2.2. Cálculo del período del parpadeo

El timer1 posee un registro de conteo de 16 bits. Por lo tanto, en modo normal, cuenta de 0 a 2^16. Cuando llega al máximo valor posible (en hexadecimal 0xFFFF), este produce un overflow que activa la rutina de interrupción, modificando el estado del pin donde se conectó el LED. Al producirse el overflow, el registro vuelve a contar desde cero. La velocidad con la que el timer va contando es igual a la velocidad del clock que se tenga. En este caso, al utilizar un arduino, la velocidad de clock se encuentra en 16MHz. Si se usa un prescaler, se baja la frecuencia de conteo del registro por un factor de 64, 256 o 1024 según sea el valor del prescaler.

El cálculo de cada cuanto se prende el led será entonces la distancia temporal entre dos overflows:

$$\tau = 2 \cdot \frac{2^{16}}{16MHz/prescaler} \tag{1}$$

Esta cuenta da un período de 0.52s para el prescaler de 64, 2.09s para el prescaler de 256 y 8.39s para el de 1024.

3. Diagrama en bloques

El diagrama de conexiones en bloque del trabajo se presenta a continuación:

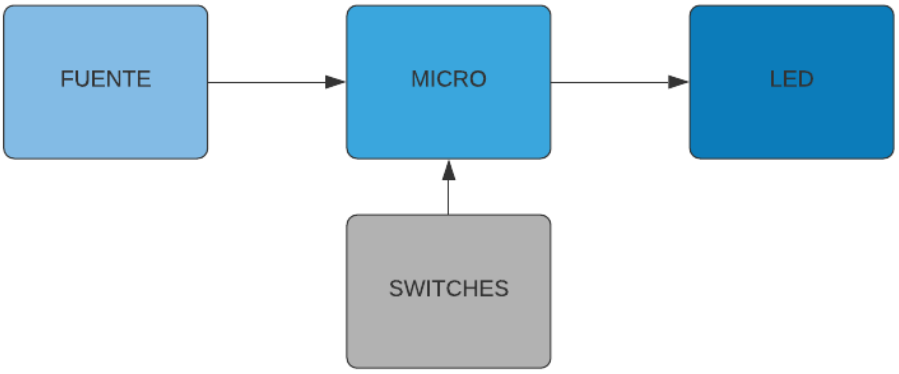


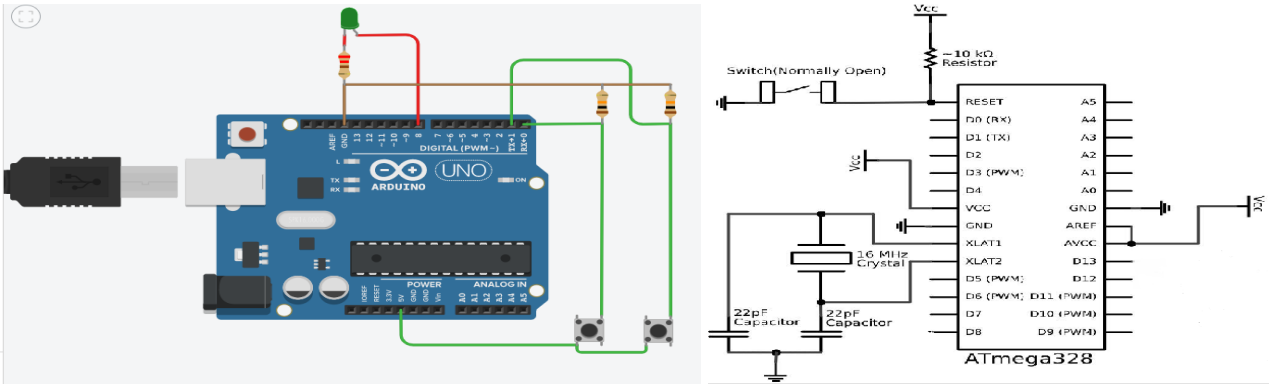
Figura 1: Diagrama de conexiones en bloque

Se observa que el funcionamiento surge primero de la pc donde se encuentra el código que será descargado en el micro y a su vez sirve como una fuente. El micro recibe información del estado de los botones. En base a si se están pulsando o no, el micro modificará el registro de control del timer1, modificando los prescalers. Esto es lo que se ve en el diagrama.

4. Esquemático y listado de componentes

Para el circuito se utilizaron los siguientes componentes:

- 2 resistores de 10kΩ
- 1 resistor de 220Ω
- 1 led de color verde
- 2 pulsadores
- 7 cables para protoboard
- 1 placa arduino con atmega328p
- 1 protoboard



(a) Esquemático del circuito implementado (b) Circuito con cristal de 16MHz

Figura 2: Esquemático del circuito

En total se habrá gastado unos 150 pesos para el circuito. Sin contar la plataforma arduino.

La plataforma arduino le agrega al microcontrolador un cristal de 16Mhz y una resistencia de $10k\Omega$ como se observa en a figura 2b.

5. Diagrams de flujo

A continuación se presentan los diagramas de flujo. El primero consiste en el programa principal más la configuración previa. La configuración de puertos y de la interrupción se puso todo en un mismo bloque ya que esto ya se vio en los trabajos anteriores. La subrutina de polling se encuentra al lado. Esta rutina se encarga de preguntar cuál es el valor del puerto d y en base a eso ejecutar la instrucción correspondiente. Nótese que, si bien se pregunta innecesariamente si el valor del puerto es 3, esto se configuró de esta manera con el propósito de hacer el código más legible y más fácil de agregarle, si se quisiera a futuro, más modos de operaciones. Por último, se presenta el diagrama de flujo de la interrupción del timer. Este consta de prender y apagar el LED.

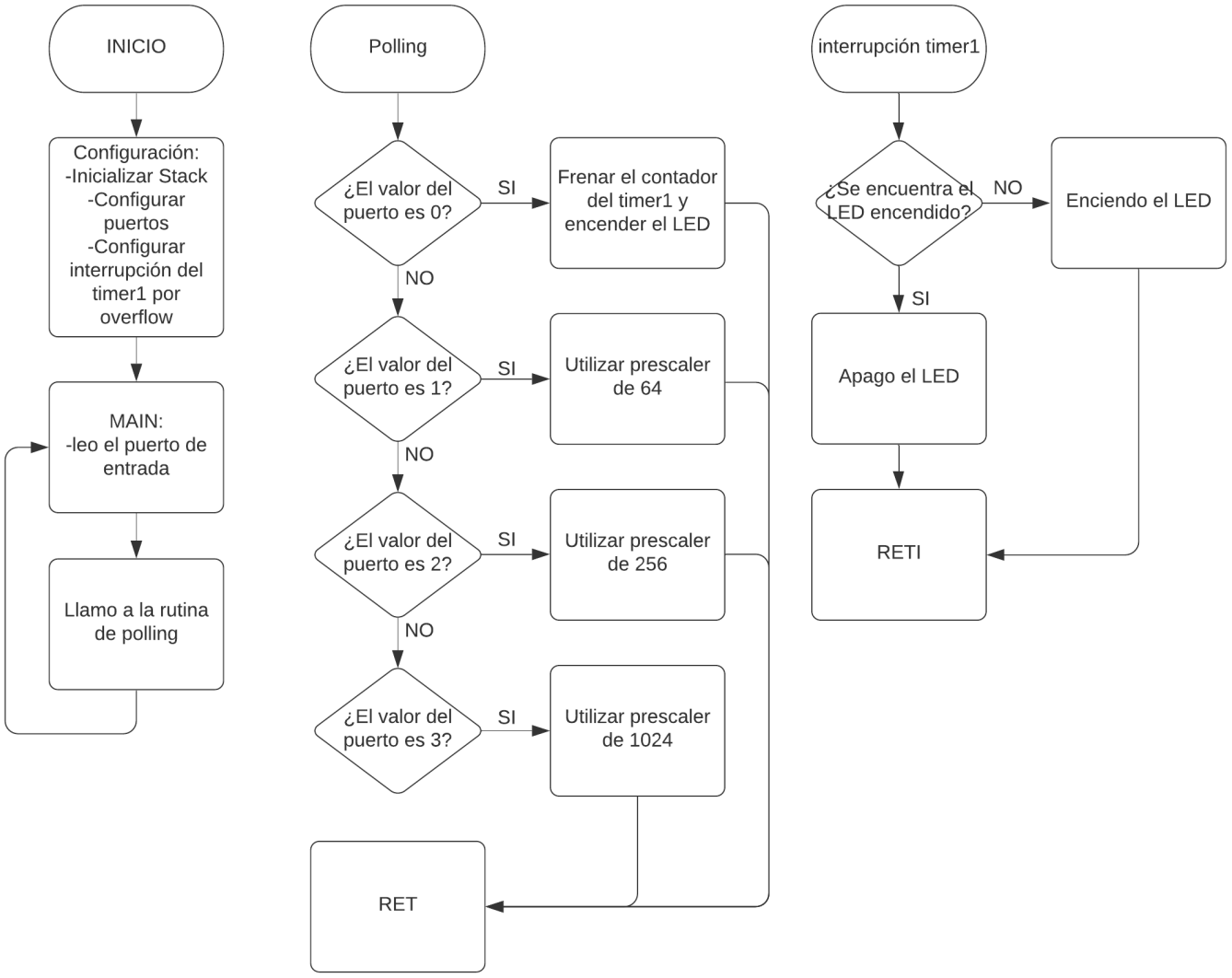


Figura 3: Diagramas de flujo

6. Códigos

A continuación se presenta el código:

```
.include "m328pdef.inc"

.cseg
   ;direcciones de codigo
.org 0x0000
   rjmp  configuracion
.org OVF1addr
   rjmp   TO_1V_ISR
.org INT_VECTORS_SIZE

configuracion:
   ;inicializo el stack
   ldi      R16, HIGH(RAMEND)
   out      SPH, R16
   ldi      R16, LOW(RAMEND)
   out      SPL, R16

   ;declaro el PORTB como salida
   sbi      DDRB, 0

   ;declaro el PORTD como entrada
   cbi      DDRD, 0
   cbi      DDRD, 1

   ;enciendo la interrupcion por overflow del timer1
   ldi      R16, (1<<TOIE1)
   sts      TIMSK1, R16

   sei

main:
   clr      R16
   in    R16, PIND                        ;verifico estado puerto D
   andi  R16, (1 << PIND1 | 1 << PIND0)         ;Se hacen 2 lecturas con un delay en el medio y se
    comparan si ambas fueron iguales
                                      ;De ser as  se entra a la rutina de polling
   rcall delay

   in    R17, PIND
   andi  R17, (1 << PIND1 | 1 << PIND0)

   cp    R16, R17
   breq  polling

   jmp      main

polling:
   cpi      R16, 0               ;lo que se hace aca, es ir comparando los valores del puerto D con
   breq  fijo             ;los 4 estados posibles. Cuando el puerto sea igual a uno, se modifica el
                          ;prescaler acorde a las intrucciones del TP.
   cpi      R16, 1
   breq  clk_64

   cpi      R16, 2
   breq  clk_256

   cpi      R16, 3
   breq  clk_1024

   fijo:
      ldi      R16, 0x00
      sts      TCCR1B, R16
      sbi      PORTB, 0          ;por default, el led se encuentra encendido
   ret

   clk_64:
      ldi      R16, 0X03
      sts      TCCR1B, R16          ;aca se esta modificando el registro de control, utilizando los
    prescalers
   ret

   clk_256:
      ldi      R16, 0X04
```

```
72        sts       TCCR1B, R16
73     ret
74
75     clk_1024:
76        ldi       R16, 0X05
77        sts       TCCR1B, R16
78     ret
79
80  TO_1V_ISR:                    ;rutina de encendido y apagado del led
81        sbis  PORTB, 0       ;se ejecutara cuando el conteo del timer produzca un overflow
82        rjmp  enciendo
83        cbi       PORTB, 0
84  reti
85
86        enciendo:
87        sbi       PORTB, 0
88  reti
89
90  delay:
91        ldi r23, 32
92        loop3: ldi r24, 255
93        loop2: ldi r22, 255
94        loop1: dec r22
95        brne loop1
96        dec r24
97        brne loop2
98        dec r23
99        brne loop3
100 ret
```

7. Resultados

Los resultados fueron los esperados, no hubo mayor complicación para realizar el trabajo. So observó que los periodos de encendido y apagado del led fueron aproximádamente parecido a los calculados. Tampoco hubo problemas de rebote.

8. Conclusiones

Como se ve en este trabajo, no siempre es incorrecto usar una lógica de polling y también puede llegar a ser útil si no se requieren implementar otras tareas simultáneamente.

A partir de los registros de timers se puede observar una manera mucho más útil para generar retardos, puesto que este es producido por una interrupción del micro, este puede seguir ejecutando otras tareas. Es una técnica más eficiente que sólo quemar ciclos de máquina. No hace falta decir que, modificar y calcular el valor del delay es además mucho más fácil con este método. Cabe mencionar que estos registros pueden ser usados de diferentes maneras según se configuren los registros de control, lo que, de nuevo, los hace una herramienta de mucha utilidad.

Haciendo el trabajo se puede concluir que además, es de suma importancia saber la frecuencia del oscilador.