Laboratorio de Microprocesadores - 86.07

Trabajo Práctico Obligatorio N°8

Puerto serie

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								
Nombre	Apellido	Padrón									
Santiago	López	100566									

Observaciones:

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

Fecha de aprobación				Firma J.T.P

Coloquio	
Nota final	
Firma profesor	

Índice

1. Objetivo

Comunicarse a través del puerto serie con una computadora. Enviar mensajes que serán impresos en *stdout* y leer los mensajes enviados por *stdin*, para luego procesarlos y realizar las operaciones acordes.

2. Desarrollo

Para llevar a cabo el programa se definieron una serie de mensajes como tablas ROM, a las cuales se les apuntó el registro Z, para así recorrerlas y transmitir los datos pedidos. Luego, se inició un ciclo sin fin, en el cual se llama a la rutina que espera la recepción de un mensaje, el cual luego es procesado para verificar qué LED es el que debe cambiar su estado.

Para la transmisión y recepción de mensajes se configuró el puerto serie del ATmega328P. Dado que los mensajes son caracteres dentro de la tabla ASCII, los mensajes se configuraron de 8 bits, sumado del bit de arranque y un bit de parada; siendo así un total de 10 bits por mensaje.

En un caso alternativo se configuró el puerto serie para que llame a una interrupción en cuanto haya un mensaje a recibir. Se mencionarán las diferencias entre estas dos alternativas en la sección de Conclusiones.

3. Diagrama de conexiones en bloque

Las conexiones siguieron el esquema de la Figura 1.

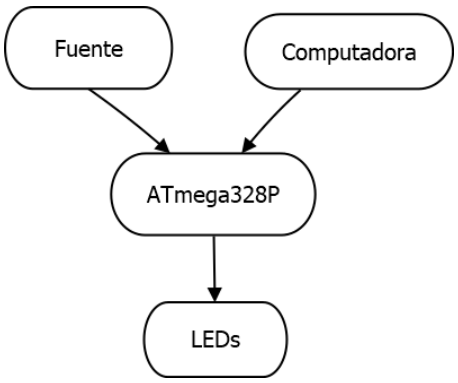

Figura 1: Diagrama de conexiones en bloque.

4. Esquemático

La Figura 2 muestra las conexiones eléctricas efectuadas en el práctico. Los componentes utilizados se encuentran en la sección siguiente.

Para este práctico se decidió incluir el microcontrolador ATmega8U2 ya que cuenta con el conversor TTL USB utilizado para la comunicación.

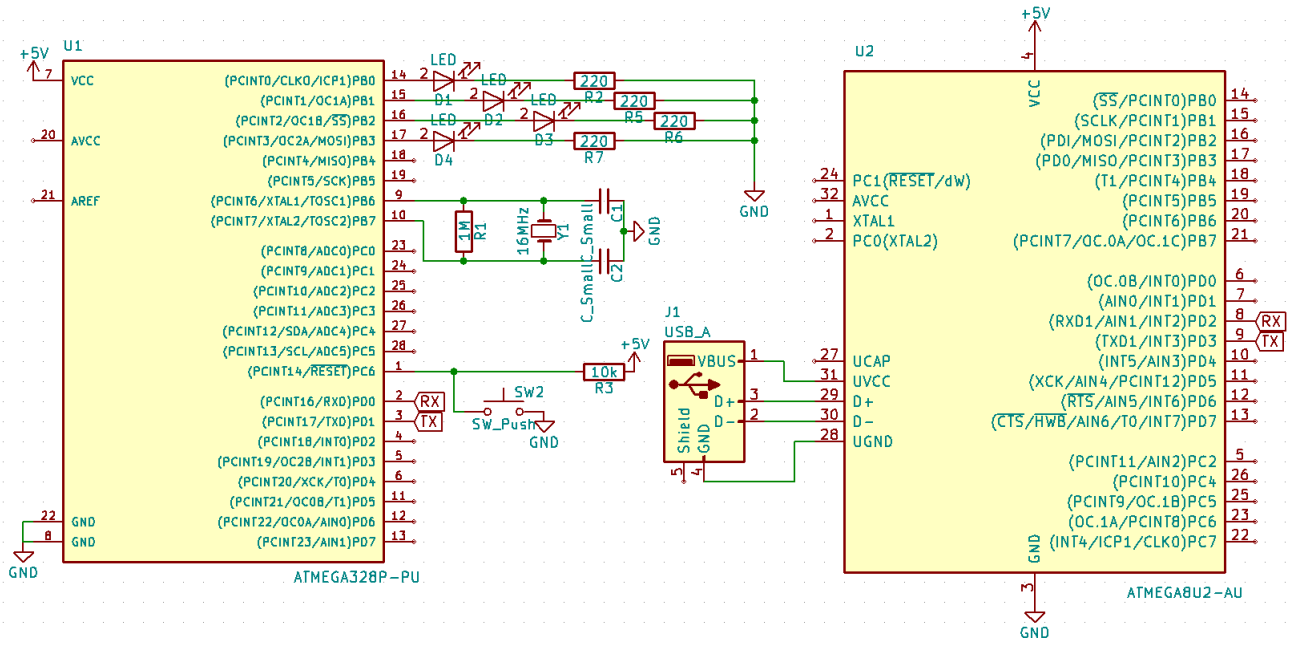

Figura 2: Circuito esquemático.

5. Listado de componentes

Los componentes utilizados fueron los listados a continuación:

- Placa de desarrollo Arduino UNO
- Resistores de 220Ω
- LEDs de color rojo
- Protoboard
- Cables unipolares

6. Diagrama de flujo

Dadas las dos consignas del trabajo práctico, se decidió realizar distintos diagramas de flujo. Por un lado hay dos figuras que cuentan con los diagramas de las implementaciones con y sin interrupciones al recibir datos, y por el otro se cuenta con una figura con rutinas en común entre los dos programas.

En el diagrama de la Figura 3 se encuentran los diagramas del programa sin interrupciones. El de la izquierda es el main del programa, el del centro es la rutina configuración del puerto serie y el de la derecha representa la rutina de recepción de mensajes.

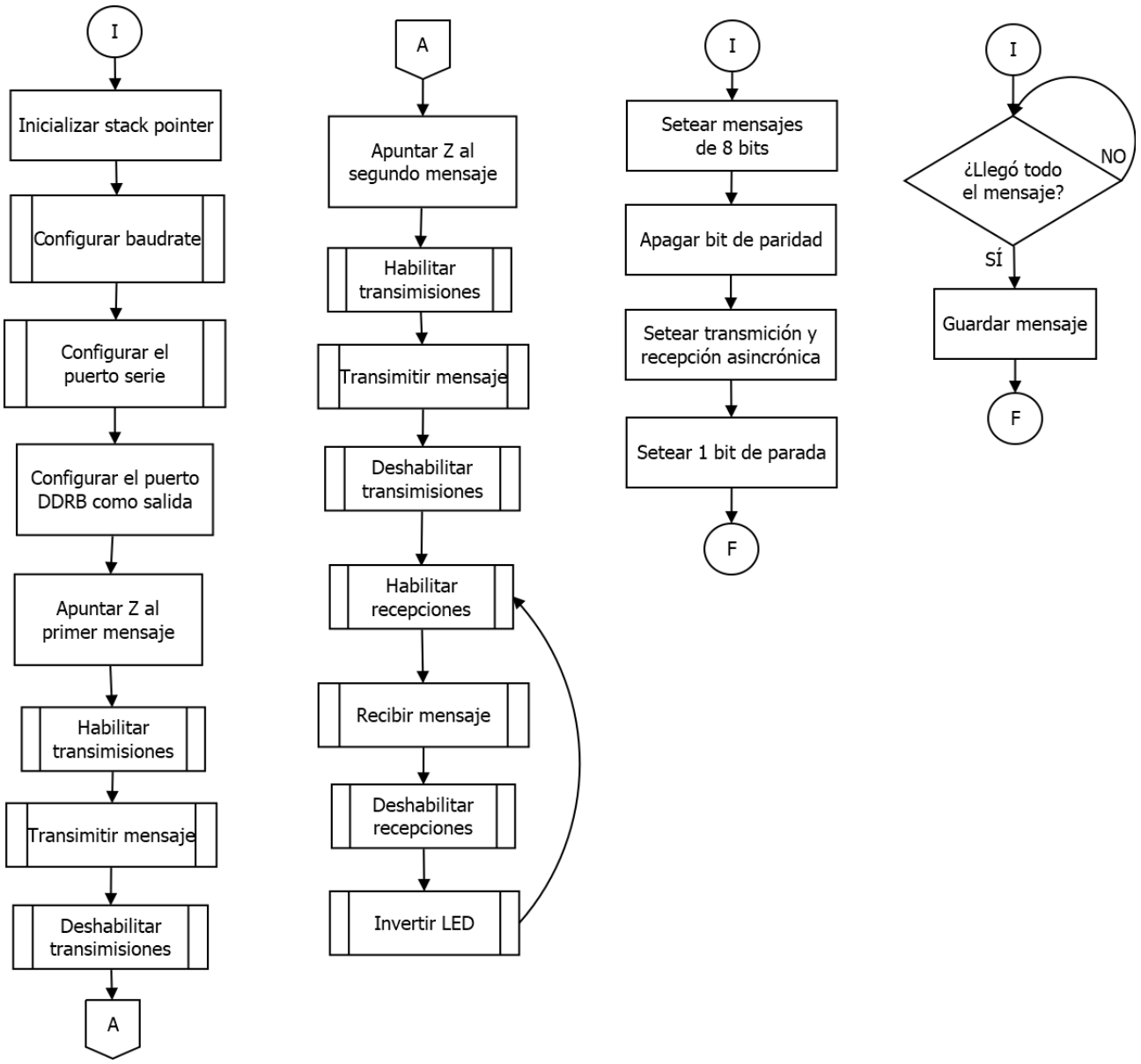

Figura 3: Diagrama de flujo.

En la Figura 4 se encuentran las rutinas en común entre los dos programas. La primera rutina, leyendo de izquierda a derecha, representa la transmisión de un mensaje de n bytes, con el uso de un caracter de fin de mensaje. La segunda rutina cuenta con la configuración del registro UBRR0, con tal de tener un *baudrate* de 9600 bits por segundo. La tercera rutina muestra cómo se alterna el estado del LED indicado por *stdin*.

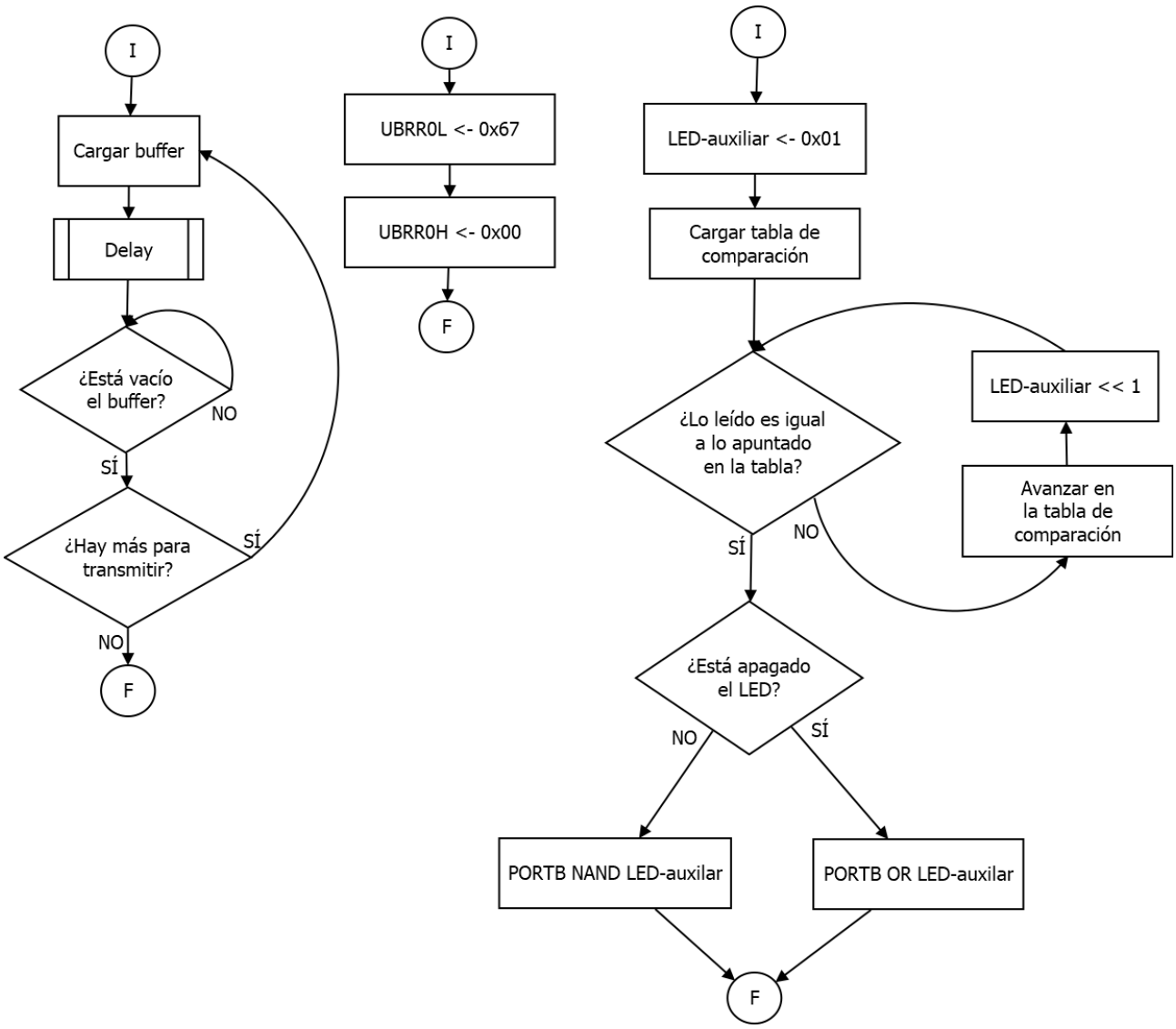

Figura 4: Diagrama de flujo - Subrutinas.

Finalmente, el diagrama de la Figura 5 muestra la variación de las rutinas de la Figura 3 al realizarse el programa con interrupciones al recibirse un mensaje. Se tiene el main por izquierda, la configuración del puerto serie con interrupciones en el centro, y por la derecha, la interrupción generada al recibir un mensaje.

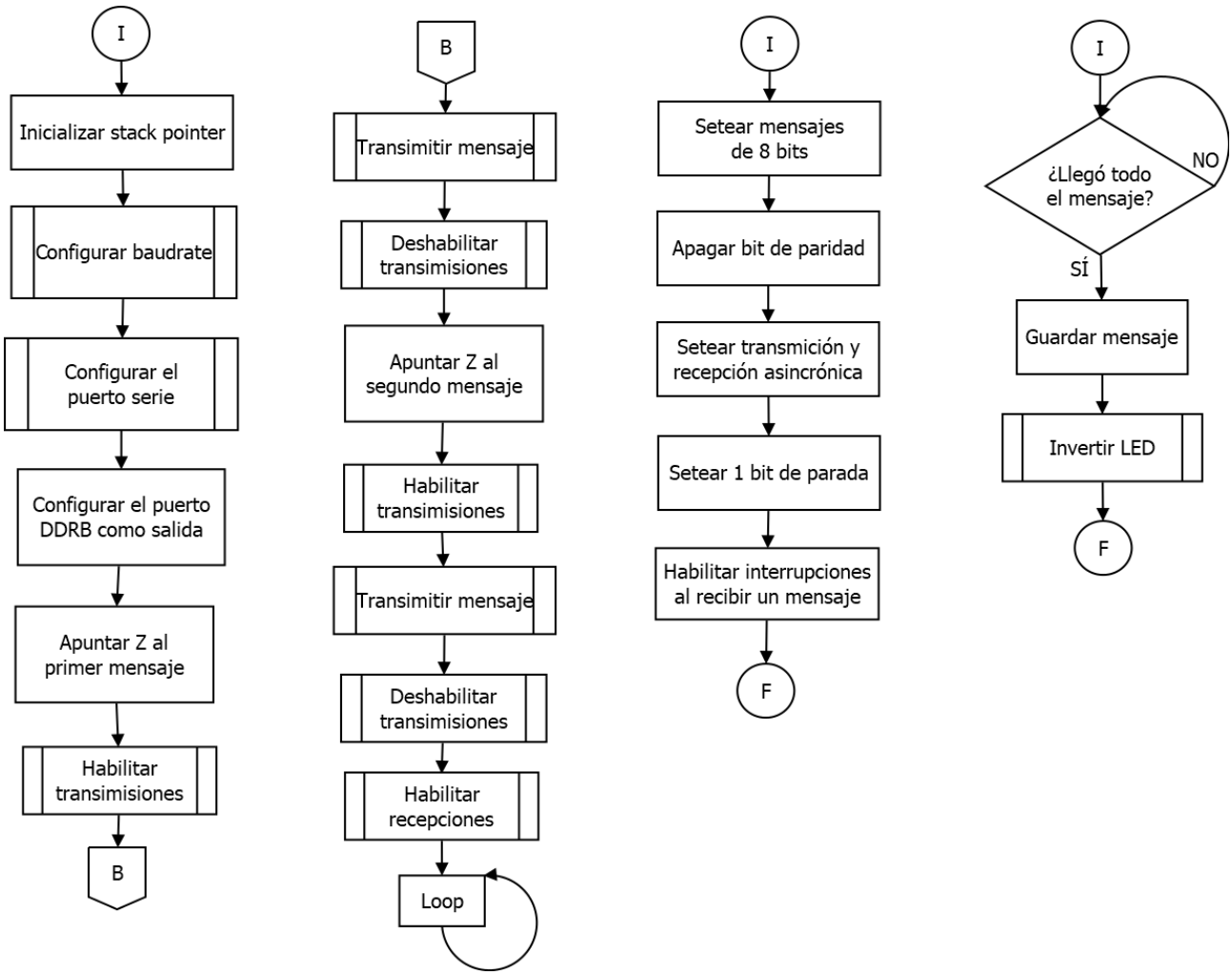

Figura 5: Diagrama de flujo - Con interrupciones.

7. Código fuente

Se presentan dos versiones del programa pedido. En la primera versión se realiza el programa sin el uso de interrupciones del puerto serie, mientras que en el segundo sí se esperan interrupciones para proceder.

```
 1: .include "m328pdef.inc"
 2:
 3: .dseg
 4:
 5: .def conf = r16
 6: .def read = r17
 7: .def aux = r18
 8: .def led = r19
 9: .def write = r20
10: .equ led_port = portb
11:
12: .macro init_sp
13:     ldi conf, low (RAMEND)
14:     out spl, conf
15:     ldi conf, high(RAMEND)
16:     out sph, conf
17: .endmacro
18:
19:
20: .macro setup_output_port
21:     ldi conf, 0xff
22:     out @0, conf
23: .endmacro
24:
25: .macro init_zp
26:     ldi zl, low (@0 << 1)
27:     ldi zh, high(@0 << 1)
28: .endmacro
29:
30: .cseg
31:
32: .org 0x0000
33:     jmp main
34: .org 0x0020
35:     jmp timer_isr
36:
37: .org INT_VECTORS_SIZE
38:
39: main:
40:
41:     init_sp
42:     sei
43:     call setup_baudrate
44:
45:     call setup_serial_port
46:
47:     setup_output_port ddrb
48:
49:     init_zp welcome_message
50:     call transmit_message
51:
52:     init_zp query_message
53:     call transmit_message
54:
55: loop:
56:     call recieve_message
57:     call toggle_leds
58:
59:     jmp loop
60:
61:
62:
63: //subrutinas
64:
65:
66: // BAUDRATE
67:
68: // UBRRn = 16Meg / 9600 - 1 = 103 = 0x0067
69: // UBRR0L <- 0x67
```

```
 70: // UBRR0H <- 0x00
 71:
 72: setup_baudrate:
 73:     ldi conf, 0x67
 74:     sts UBRR0L, conf
 75:     ldi conf, 0x00
 76:     sts UBRR0H, conf
 77: ret
 78:
 79:
 80: // CONFIGURACION PUERTO SERIE
 81:
 82: // UCSZ0  <- 011 ; 8 bits
 83: // UPM0   <-  00 ; sin paridad
 84: // UMSEL0 <-  00 ; USART asincronico
 85: // USBS0  <-   0 ; 1 stop bit
 86: // RXEN0 enable
 87: // TXEN0 enable
 88: // sin 9eno bit
 89: // interrupciones apagadas
 90: setup_serial_port:
 91:     ldi conf, 0x18 ; 0b00011000
 92:     sts UCSR0B, conf
 93:     ldi conf, 0x06 ; 0b00000110
 94:     sts UCSR0C, conf
 95: ret
 96:
 97:
 98: // rutina de transmision
 99:
100: transmit_message:
101:
102: transmit_next:
103:     lpm write, z+
104:     sts UDR0, write
105:
106:     call delay
107:
108: sending:
109:     lds aux, UCSR0A
110:     sbrs aux, UDRE0 ; itero hasta vaciar el buffer
111:     rjmp sending
112:
113:     cpi write, 0x00
114:     brne transmit_next
115: ret
116:
117:
118: // rutina de recepcion
119:
120: recieve_message:
121:
122: reading:
123:     lds aux, UCSR0A
124:     sbrs aux, RXC0  ; espero a recibir la data
125:     rjmp reading
126:
127:     lds read, UDR0
128: ret
129:
130: // DELAY PARA DAR TIEMPO AL BUFFER
131:
132: delay:
133:     ldi conf, 0x01
134:     sts timsk0, conf
135:     ldi conf, 0x01
136:     out tccr0b, conf
137: here:   ; termina la rutina luego de la interrupcion
138:     cpi conf, 0x00
```

```
139:     brne here
140: ret
141:
142:
143: // toggle
144:
145: toggle_leds:
146:     ldi led, 0x01
147:     init_zp compare_table
148:
149: cmp:
150:     lpm aux, z+     ; recorro la tabla para setear que led toca
151:     cp read, aux
152:     breq toggle
153:
154:     lsl led
155:     jmp cmp
156:
157: toggle:
158:     in aux, led_port
159:     and aux, led
160:     cpi aux, 0x00
161:     brne turn_off
162:     in aux, led_port    ; vuelvo a leer para no perder el resto de los estados
163:     or aux, led         ; or para imponer 1
164:     out led_port, aux
165:
166:     ret
167: turn_off:
168:     in aux, led_port
169:     com led
170:     and aux, led        ; and negada para imponer 0
171:     out led_port, aux
172: ret
173:
174:
175: timer_isr:
176:     clr conf
177:     out tccr0b, conf
178: reti
179:
180: //TABLAS
181:
182: .org 0x0500
183:
184: welcome_message: .db "*** Hola Labo de Micros ***", 0x0d, 0x0a, 0x00
185:
186: query_message: .db "Escriba 1, 2, 3 o 4 para controlar los LEDs", 0x0d, 0x0a, 0x00
187:
188: compare_table: .db '1', '2', '3', '4'
```

```
 1: .include "m328pdef.inc"
 2:
 3: .dseg
 4:
 5: .def conf = r16
 6: .def read = r17
 7: .def aux = r18
 8: .def led = r19
 9: .def write = r20
10: .equ led_port = portb
11:
12: .macro init_sp
13:     ldi conf, low (RAMEND)
14:     out spl, conf
15:     ldi conf, high(RAMEND)
16:     out sph, conf
17: .endmacro
18:
19:
20: .macro setup_output_port
21:     ldi conf, 0xff
22:     out @0, conf
23: .endmacro
24:
25: .macro init_zp
26:     ldi zl, low (@0 << 1)
27:     ldi zh, high(@0 << 1)
28: .endmacro
29:
30: .cseg
31:
32: .org 0x0000
33:     jmp main
34: .org 0x0020
35:     jmp timer_isr
36: .org 0x0024
37:     jmp recieve_isr
38:
39: .org INT_VECTORS_SIZE
40:
41: main:
42:
43:     init_sp
44:
45:     call setup_baudrate
46:
47:     call setup_serial_port
48:
49:     setup_output_port ddrb
50:
51:     sei
52:
53:     init_zp welcome_message
54:     call transmit_message
55:
56:     init_zp query_message
57:     call transmit_message
58:
59: loop:
60:     jmp loop
61:
62:
63:
64: //subrutinas
65:
66:
67: // BAUDRATE
68:
69: // UBRRn = 16Meg / 9600 - 1 = 103 = 0x0067
```

```
 70: // UBRR0L <- 0x67
 71: // UBRR0H <- 0x00
 72:
 73: setup_baudrate:
 74:     ldi conf, 0x67
 75:     sts UBRR0L, conf
 76:     ldi conf, 0x00
 77:     sts UBRR0H, conf
 78: ret
 79:
 80:
 81: // CONFIGURACION PUERTO SERIE
 82:
 83: // UCSZ0  <- 011 ; 8 bits
 84: // UPM0   <-  00 ; sin paridad
 85: // UMSEL0 <-  00 ; USART asincronico
 86: // USBS0  <-   0 ; 1 stop bit
 87: // RXEN0 enable
 88: // TXEN0 enable
 89: // sin 9eno bit
 90: // interrupciones por recepcion
 91: setup_serial_port:
 92:     ldi conf, 0x98 ; 0b10011000
 93:     sts UCSR0B, conf
 94:     ldi conf, 0x06 ; 0b00000110
 95:     sts UCSR0C, conf
 96: ret
 97:
 98:
 99: // rutina de transmision
100:
101: transmit_message:
102:
103: transmit_next:
104:     lpm write, z+
105:     sts UDR0, write
106:
107:     call delay
108:
109: sending:
110:     lds aux, UCSR0A
111:     sbrs aux, UDRE0 ; itero hasta vaciar el buffer
112:     rjmp sending
113:
114:     cpi write, 0x00
115:     brne transmit_next
116: ret
117:
118: // rutina de recepcion
119:
120: recieve_message:
121:
122: reading:
123:     lds aux, UCSR0A
124:     sbrs aux, RXC0  ; espero a recibir la data
125:     rjmp reading
126:
127:     lds read, UDR0
128: ret
129:
130:
131: // DELAY PARA DAR TIEMPO AL BUFFER
132:
133: delay:
134:     ldi conf, 0x01
135:     sts timsk0, conf
136:     ldi conf, 0x01
137:     out tccr0b, conf
138:     sei
```

```
139: here:   ; termina la rutina luego de la interrupcion
140:     cpi conf, 0x00
141:     brne here
142: ret
143:
144:
145: // toggle
146:
147: toggle_leds:
148:     ldi led, 0x01
149:     init_zp compare_table
150:
151: cmp:
152:     lpm aux, z+     ; recorro la tabla para setear que led toca
153:     cp read, aux
154:     breq toggle
155:
156:     lsl led
157:     jmp cmp
158:
159: toggle:
160:     in aux, led_port
161:     and aux, led
162:     cpi aux, 0x00
163:     brne turn_off
164:     in aux, led_port    ; vuelvo a leer para no perder el resto de los estados
165:     or aux, led         ; or para imponer 1
166:     out led_port, aux
167:
168:     ret
169: turn_off:
170:     in aux, led_port
171:     com led
172:     and aux, led        ; and negada para imponer 0
173:     out led_port, aux
174: ret
175:
176:
177: timer_isr:
178:     clr conf
179:     out tccr0b, conf
180: reti
181:
182: recieve_isr:
183:     lds aux, UCSR0A
184:     sbrs aux, RXC0  ; espero a recibir la data
185:     rjmp reading
186:
187:     lds read, UDR0
188:
189:     call toggle_leds
190: reti
191:
192: //TABLAS
193:
194: .org 0x0500
195:
196: welcome_message: .db "*** Hola Labo de Micros ***", 0x0d, 0x0a, 0x00
197:
198: query_message: .db "Escriba 1, 2, 3 o 4 para controlar los LEDs", 0x0d, 0x0a, 0x00
199:
200: compare_table: .db '1', '2', '3', '4'
```

8. Costos

A continuación se presenta un listado de los costos de los componentes utilizados en el práctico.

- Arduino UNO - $850
- Resistores - $50
- LEDs rojo - $100
- Protoboard - $240
- Cables unipolares - $150

Sumando un costo total de $1390.

9. Conclusiones

Como se mencionó previamente, el práctico cuenta con dos implementaciones para el mismo problema. Las diferencias entre estas implementaciones no son menores, ya que el uso de las interrupciones permite que el programa siga corriendo hasta el momento en que se recibe un nuevo mensaje, para entonces procesarlo y realizar las operaciones necesarias.

En el caso original, el programa se vió obligado a detenerse por un tiempo indeterminado, hasta que finalmente llegara un nuevo mensaje. A los fines del práctico, esto no fue perjudicial, ya que de no llegar un mensaje, no había nada por realizar, pero para un programa con más funcionalidades esto puede ser una herramienta muy útil para no desperdiciar el tiempo ni recursos del micro en tan solo esperar.