Universidad de Buenos Aires

Facultad de Ingeniería

2do Cuatrimestre 2020

[86.07] Laboratorio de Microprocesadores

Trabajo Práctico Nº 3

Alumno: Agustín Miguel Flouret

Padrón: 102298

Turno: Martes

Docentes:

- Gerardo Stola
- Guido Salaya
- Fernando Cofman

Fecha de entrega: 10 de noviembre de 2020

Índice

1. Objetivo del proyecto

En este proyecto se buscará utilizar el puerto serie para establecer una comunicación serie bidireccional entre un microcontrolador AVR y una computadora de escritorio.

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 el Assembly de AVR, y se ensamblará con Atmel Studio. Además, se utilizara una computadora para comunicarse por USB con el microcontrolador, utilizando el terminal serie incluido en el IDE de Arduino. El hardware externo se detallará en la siguiente sección.

Las tareas a realizar son las siguientes:

- Al encender el microcontrolador, el programa debe transmitir un mensaje que estará cargado en la memoria flash, y que debe mostrarse en el terminal serie.
- Además, al apretar en el terminal serie la teclas '1', '2', '3' o '4', se debe encender el LED 1, 2, 3 o 4 respectivamente. Los LEDs estarán conectados en el puerto C del microcontrolador.
- Primero se realizarán estas tareas sin utilizar interrupciones, y luego utilizando interrupciones relacionadas con el puerto serie.

3. Listado de componentes y gastos

- 1 Arduino UNO - $1009
- 1 protoboard - $300
- 2 LED rojos - $20
- 2 LED verdes - $20
- 4 resistores 220Ω - $28
- **Gasto total: $1377**

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 ampliadas del esquemático.

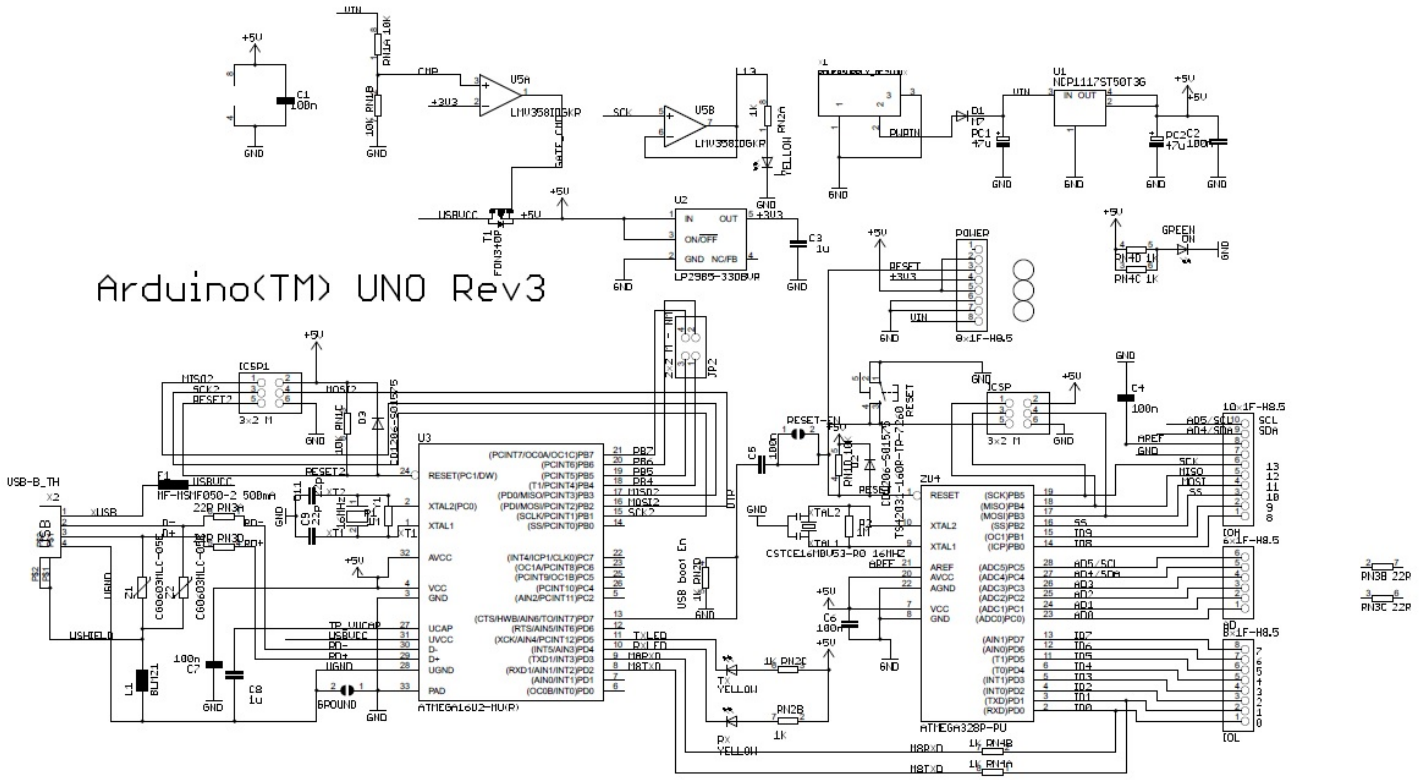

Figura 1: Circuito esquemático de la placa Arduino UNO.

5. Desarrollo del proyecto

Se conectaron 4 LEDs, con resistores de 220Ω en serie, a los pines del puerto C del 0 al 3. Para comunicarse con la computadora se utilizó la salida USB del Arduino, ya que este tiene un conversor USB a serie incorporado. A continuación se muestra el diagrama de conexiones en bloques y el circuito esquemático del proyecto.

5.1. Diagrama de conexiones en bloques

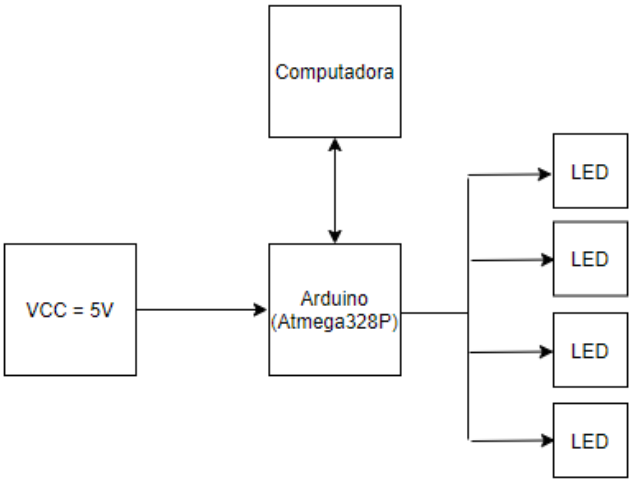

Figura 2: Diagrama de bloques del proyecto.

5.2. Circuito esquemático

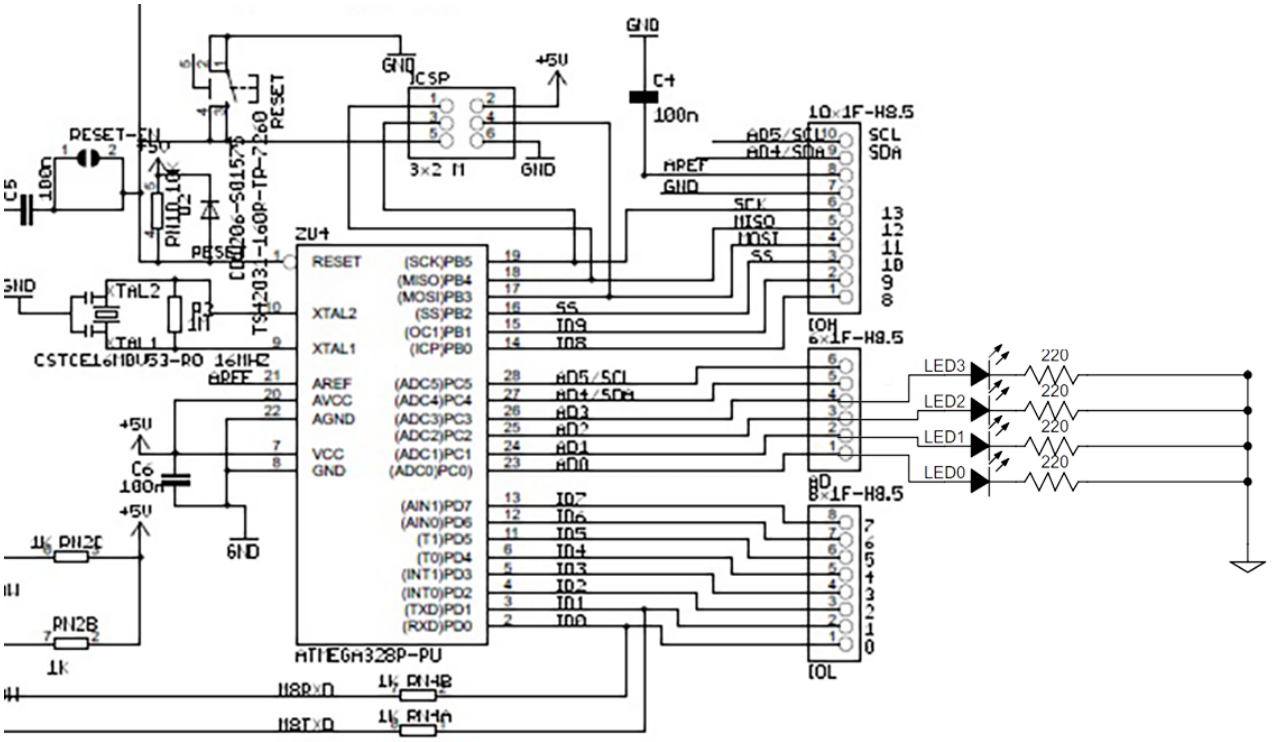

Figura 3: Circuito esquemático que muestra las conexiones con los LEDs.

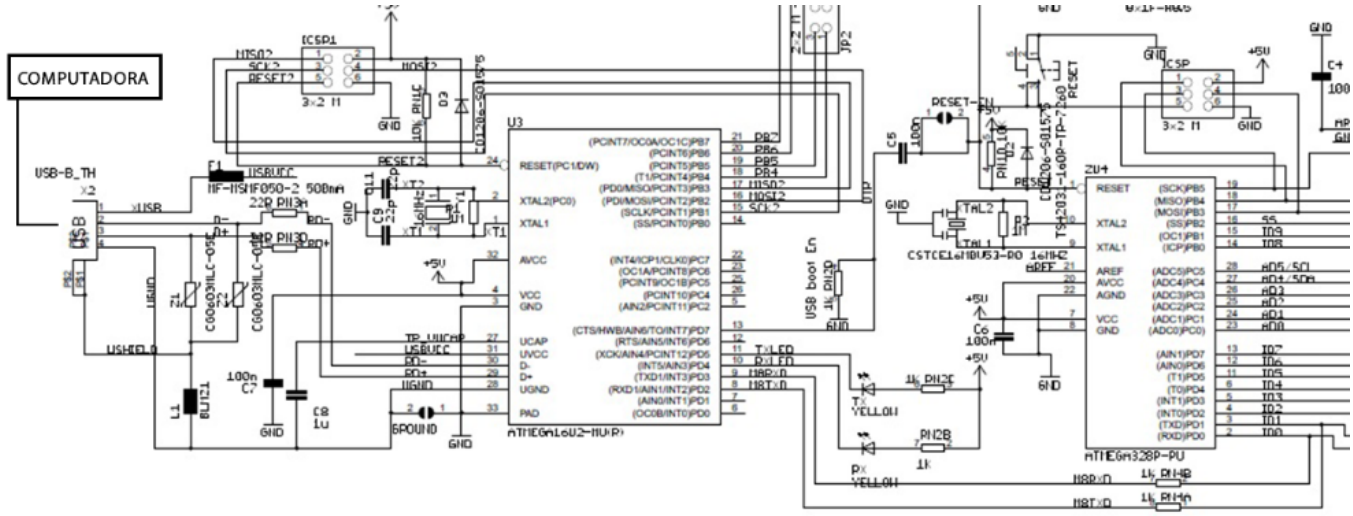

Figura 4: Circuito esquemático que muestra la conexión con la computadora.

5.3. Software

Al iniciar el programa se inicializa el stack y se configura el puerto C (pines 0 a 3) como salida, donde están conectados los 4 leds. En los registros UCSR0B y UCSR0C se configura el puerto serie, para que pueda transmitir y recibir información en formato 8N1 (8 bits de datos, sin bit de paridad y 1 bit de stop). También se llama a una rutina de delay de 2 segundos para poder abrir el terminal serie cuando se enciende el microcontrolador, sin perder el mensaje transmitido. Por último, se carga la dirección de memoria del mensaje a transmitir en el puntero Z.

En el caso del programa sin interrupciones, el programa consultará si el buffer de transmisión UDRE0 está vacío cada vez que se quiera transmitir un byte. En cambio, en el programa con interrupciones, se activa la interrupción UDRIE0 automáticamente cuando este buffer está vacío.

Lo mismo sucede con la recepción. En el programa con interrupciones, la interrupción de transmisión RXCIE0 se activa cuando el buffer de recepción RXC0 tiene datos no leídos, en lugar de consultar continuamente si esto se cumple.

Para invertir el estado del led al recibir el caracter correspondiente, se implementó una rutina de toggle que prende o apaga el led indicado según si ya está prendido o apagado.

A continuación se muestran los diagramas de flujo de los dos programas.

5.3.1. Diagrama de flujo (sin interrupciones)

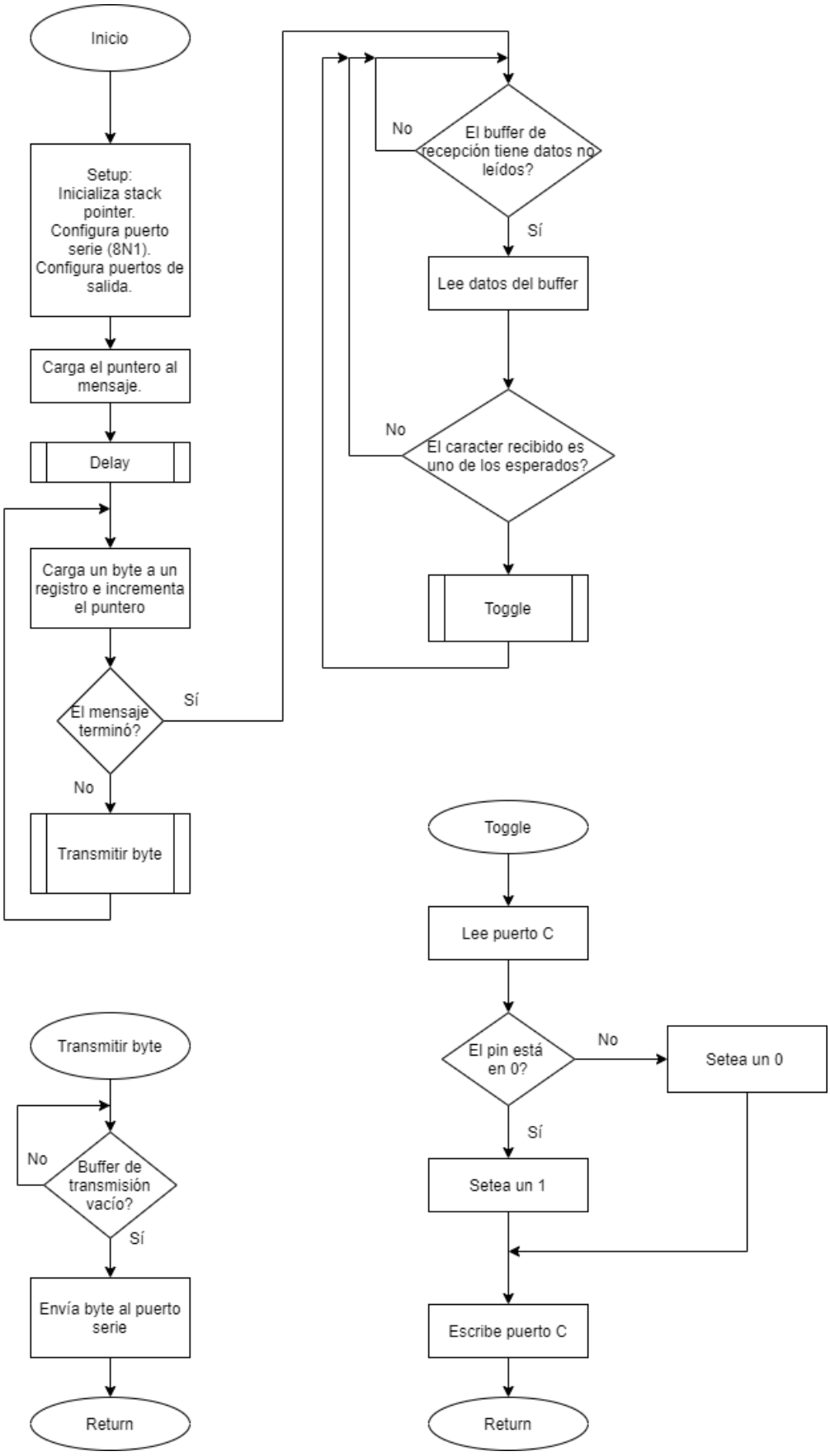

Figura 5: Diagrama de flujo del proyecto, sin utilizar interrupciones.

5.3.2. Diagrama de flujo (con interrupciones)

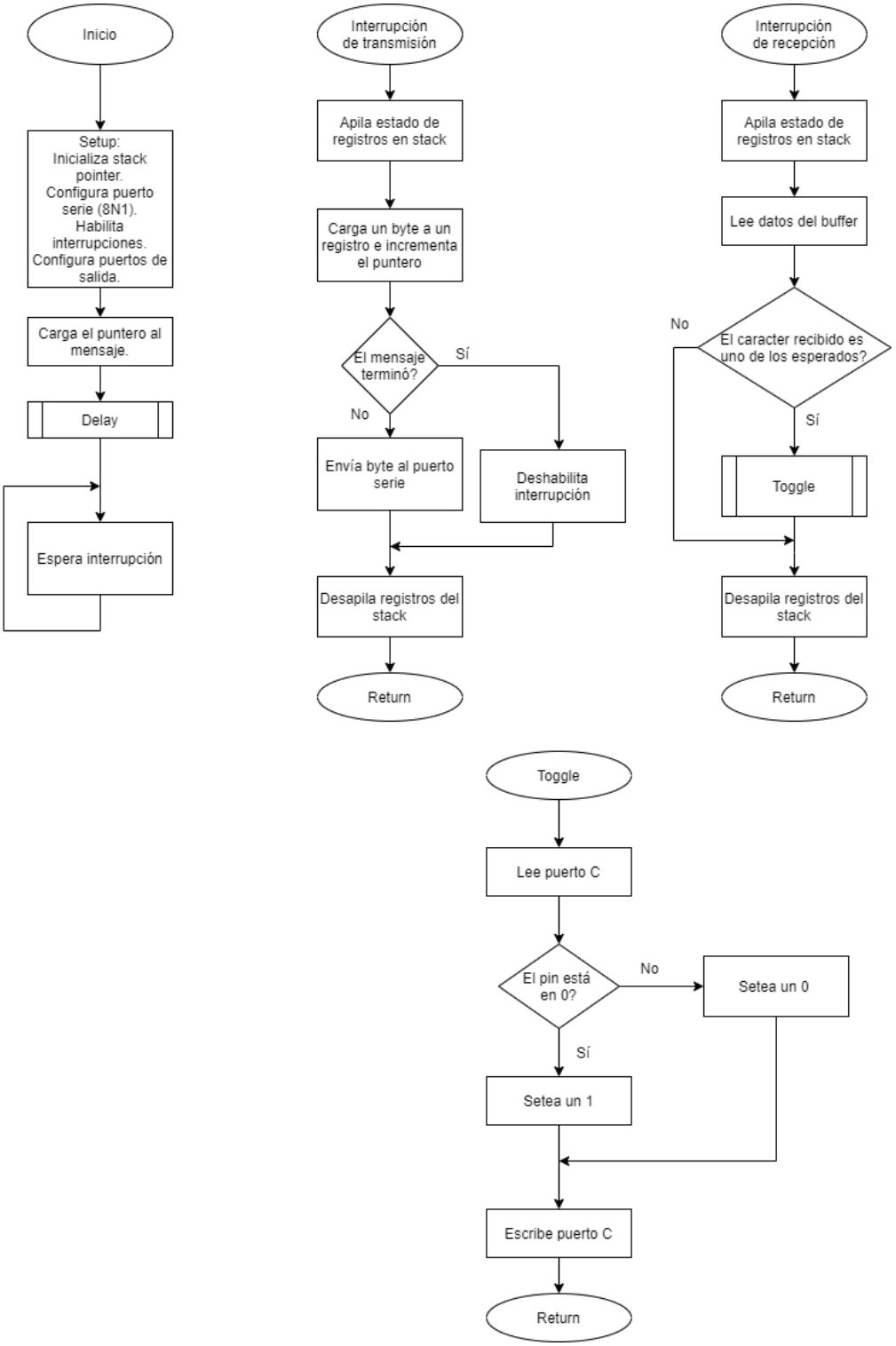

Figura 6: Diagrama de flujo del proyecto, utilizando interrupciones.

5.3.3. Código del programa (sin interrupciones)

```
.include "m328pdef.inc"

.equ    F_CPU = 16000000
.equ    baud = 9600
.equ    bps    = (F_CPU/16/baud) - 1
```

```
6
7  .org 0x0000
8      rjmp setup
9
10 mensaje:
11     .db    "*** Hola Labo de Micro ***", $0D, $0A, "Escriba 1, 2, 3 o 4 para controlar los LEDs",
12     $0A, 0, 0
13
14 setup:
15         ; Inicializa el stack
16     ldi r16, LOW(RAMEND)
17     out SPL, r16
18     ldi r16, HIGH(RAMEND)
19     out SPH, r16
20
21         ; Configura baudrate
22     ldi r16, LOW(bps)
23     ldi r17, HIGH(bps)
24         sts UBRR0L, r16
25     sts UBRR0H, r17
26
27     ldi r16,(1<<RXEN0)|(1<<TXEN0)       ; Habilita transmision y recepcion
28     sts UCSR0B,r16
29
30     ldi r16, (1<<UCSZ00)|(1<<UCSZ01);   ; Configuracion 8N1: 8 bits de datos,
31     sts UCSR0C, r16                     ; sin bit de paridad y 1 bit de stop
32
33         ldi r16, 0x0f
34         out DDRC, r16               ; Configura los pines C0 a C3 como salidas
35
36         ldi zl, LOW(mensaje<<1)     ; Carga el puntero al mensaje constante en zh:zl
37     ldi zh, HIGH(mensaje<<1)
38
39         rcall delay                 ; Delay de 2s para abrir el terminal serie
40
41 transmision_del_mensaje:
42         lpm        r16, z+                 ; Carga un byte del mensaje al registro r16, e incrementa el
43         tst r16                     ; Chequea fin de string
44     breq recepcion              ; En caso de fin, pasa a recepcion
45         call transmitir_byte        ; Transmite el byte
46         rjmp transmision_del_mensaje
47
48 ; Si el buffer de transmision esta vacio, envia el byte almacenado en r16
49 transmitir_byte:
50         lds r17, UCSR0A
51         sbrs r17, UDRE0
52         rjmp transmitir_byte
53         sts UDR0, r16
54         ret
55
56 recepcion:
57         ldi r16, 0b00000001     ; Define una mascara
58         lds r17, UCSR0A
59         sbrs r17, RXC0          ; Chequea si el buffer de recepcion tiene datos no leidos
60         rjmp recepcion
61         lds r17, UDR0           ; Carga los datos del buffer en r17
62
63 ;Compara el byte recibido con cada simbolo, y modifica la mascara
64 ;para invertir el bit correspondiente
65 c1:        cpi r17, '1'
```

```
        brne c2
        call toggle
c2: lsl r16
        cpi r17, '2'
        brne c3
        call toggle
c3: lsl r16
        cpi r17, '3'
        brne c4
        call toggle
c4: lsl r16
        cpi r17, '4'
        brne recepcion
        call toggle
        rjmp recepcion

; Invierte un bit del puerto C utilizando la mascara (r16)
toggle:
        in r18, PORTC   ; Lee puerto C
        mov r19, r18    ; Copia puerto C para modificarlo con la mascara
        and r19, r16
        cpi r19, 0      ; Verifica si el led seleccionado esta prendido o apagado
        breq prender    ; Si esta apagado, va a prender
apagar:
        com r16
        and r18, r16    ; r18 and not r16
        com r16         ; Revierte r16 al estado original
        jmp salida_toggle
prender:
        or r18, r16     ; r18 or r16
salida_toggle:
        out PORTC, r18  ; Escribe puerto C
        ret


; Delay de 48000000 ciclos (2s)
delay:
        ldi r20, 163
        ldi r21, 87
        ldi r22, 3
L1: dec  r22
    brne L1
    dec  r21
    brne L1
    dec  r20
    brne L1
        ret
```

5.3.4. Código del programa (con interrupciones)

```
.include "m328pdef.inc"

.equ    F_CPU = 16000000
.equ    baud = 9600
.equ    bps    = (F_CPU/16/baud) - 1

.org 0x0000
    rjmp setup
```

```
9
10  .org URXCaddr
11          rjmp interrupcion_recepcion
12
13  .org UDREaddr
14          rjmp interrupcion_transmision
15
16  mensaje:
17      .db    "*** Hola Labo de Micro ***", $0D, $0A, "Escriba 1, 2, 3 o 4 para controlar los LEDs",
18      $0A, 0, 0
19
20  setup:
21          ; Inicializa el stack
22      ldi r16, LOW(RAMEND)
23      out SPL, r16
24      ldi r16, HIGH(RAMEND)
25      out SPH, r16
26
27          ; Configura baudrate
28      ldi r16, LOW(bps)
29      ldi r17, HIGH(bps)
30          sts UBRR0L, r16
31      sts UBRR0H, r17
32
33          ; Habilita transmision, recepcion e interrupciones
34      ldi r16,(1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0)|(1<<UDRIE0)
35      sts UCSR0B,r16
36
37      ldi r16, (1<<UCSZ00)|(1<<UCSZ01);   ; Configuracion 8N1: 8 bits de datos,
38      sts UCSR0C, r16                     ; sin bit de paridad y 1 bit de stop
39
40          ldi r16, 0x0f
41          out DDRC, r16               ; Configura los pines C0 a C3 como salidas
42
43          ldi zl, LOW(mensaje<<1)     ; Carga el puntero al mensaje constante en zh:zl
44      ldi zh, HIGH(mensaje<<1)
45
46          rcall delay                 ; Delay de 2s para abrir el terminal serie
47
48          sei                         ; Habilita interrupciones
49
50  esperar_interrupcion:
51          rjmp esperar_interrupcion
52
53  ; Transmite por el puerto serie el mensaje guardado en la memoria flash
54  interrupcion_transmision:
55          push r16
56          in r16, sreg
57          push r16
58          push r17
59          lpm        r16, z+                ; Carga un byte del mensaje al registro r16, e incrementa el
60          tst r16                     ; Chequea fin de string
61      breq finalizar_transmision  ; En caso de fin, termina la transmision del mensaje
62          sts UDR0, r16               ; Transmite el byte
63          rjmp salida_interrupcion_transmision
64  finalizar_transmision:
65          lds r16, UCSR0B
66          cbr r16, (1<<UDRIE0)
67          sts UCSR0B, r16
68  salida_interrupcion_transmision:
```

```
69          pop r17
70          pop r16
71          out sreg, r16
72          pop r16
73          reti
74
75  ; Recibe por el puerto serie un digito ('1', '2', '3' o '4') y togglea
76  ; el led conectado al pin del puerto C 1, 2, 3 o 4, respectivamente
77  interrupcion_recepcion:
78          push r16
79          in r16, sreg
80          push r16
81          push r17
82          push r18
83          push r19
84          ldi r16, 0b00000001     ; Define una mascara
85          lds r17, UDR0           ; Carga los datos del buffer en r17
86  ;Compara el byte recibido con cada simbolo hasta encontrarlo, y desplaza
87  ;la mascara hacia la izquierda para que se corresponda con el bit a modificar
88  c1:        cpi r17, '1'
89          brne c2
90          call toggle
91  c2: lsl r16
92          cpi r17, '2'
93          brne c3
94          call toggle
95  c3: lsl r16
96          cpi r17, '3'
97          brne c4
98          call toggle
99  c4: lsl r16
100         cpi r17, '4'
101         brne salida_interrupcion_recepcion
102         call toggle
103 salida_interrupcion_recepcion:
104         pop r19
105         pop r18
106         pop r17
107         pop r16
108         out sreg, r16
109         pop r16
110         reti
111
112 ; Invierte un bit del puerto C utilizando la mascara (r16)
113 toggle:
114         in r18, PORTC   ; Lee puerto C
115         mov r19, r18    ; Copia puerto C para modificarlo con la mascara
116         and r19, r16
117         cpi r19, 0      ; Verifica si el led seleccionado esta prendido o apagado
118         breq prender    ; Si esta apagado, va a prender
119 apagar:
120         com r16
121         and r18, r16    ; r18 and not r16
122         com r16         ; Revierte r16 al estado original
123         jmp salida_toggle
124 prender:
125         or r18, r16     ; r18 or r16
126 salida_toggle:
127         out PORTC, r18  ; Escribe puerto C
128         ret
```

```
129
130
131 ; Delay de 32000000 ciclos (2s)
132 delay:
133         ldi r20, 163
134         ldi r21, 87
135         ldi r22, 3
136 L1: dec  r22
137     brne L1
138     dec  r21
139     brne L1
140     dec  r20
141     brne L1
142         ret
```

6. Resultados

Se logró obtener los resultados esperados: establecer una conexión serie entre el microcontrolador y la computadora, transmitir un mensaje y que este se muestre en el terminal serie, y controlar los LEDs enviando caracteres desde el terminal.

7. Conclusiones

En este trabajo se logró implementar con éxito lo requerido. La conexión bidireccional permitió enviar mensajes y controlar el microcontrolador con un mismo programa. Se observó la diferencia entre un programa sin interrupciones y un programa que las utiliza. En el primer caso, el programa requiere que se verifique constantemente el estado del buffer, mientras que en el segundo caso no. Sin embargo, la configuración de interrupciones añade complejidad al código.