Fiuba
Facultad de Ingeniería
Universidad de Buenos Aires

(6609) Laboratorio de Microcomputadoras

Proyecto:
(tp8 puerto serie)

Profesor:	**Ing. Jorge A. Alberto**
Cuatrimestre / Año:	1ro/2020
Turno de clases prácticas:	Miércoles
Jefe de Trabajos Prácticos:	Pedro Martos
Docente guía:	

Autores			**Seguimiento del proyecto**								
Nombre	**Apellido**	**Padrón**									
Cristian	Simonelli	87879									

Observaciones:

Fecha de aprobación

Firma J.T.P.

COLOQUIO	
Nota final	
Firma Profesor	

Objetivo:

Realizar una comunicación bidireccional entre el microcontrolador y una pc.

Desarrollo.

Se utilizará la interfaz USART.
Para ello se pueden usar los puertos rx, tx, pero para el caso de arduino se puede utilizar la interfaz USB, sin ningún cambio en la programación del micro controlador.

Registros para el uso de USART en atmega328p.

Baud rate register:
En este registro se carga un valor, que no es necesariamente útil para el programador. Dicho valor se calcula para modo de funcionamiento según la siguiente tabla:

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRRn Value
Asynchronous normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous double speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous master mode	$BAUD = \frac{f_{OSC}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{2BAUD} - 1$

Para nuestro caso (8N1) a 9600 baudos a 16Mhz.

$$Ubrr = \frac{16Mhz}{16*9600} - 1 = 103$$

Los datos se leen y escriben en un mismo registro que cumple ambas funciones. Dicho registro es UDRn.

La forma adoptada en este trabajo es la de polling en dicho registro. Para saber si se pueden escribir datos o si hay datos registros para ser leídos se usa el bit UDREn del registro UCSRnA y para ver si el registro está listo para ser escrito, se utiliza el bite RXCn.

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

La idea es similar para ambos casos, se pollea hasta que el bit indica si el registro está listo o no. De estarlo se lee o escribe según el caso.

Control and status register UCSRnB:

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Para nuestro modo de operación (asincrónico) :

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)[1]

UMSEL01, UMSEL01 = 0.

(Sin paridad)

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, even parity
1	1	Enabled, odd parity

UPM01, UPM00 = 0

(un bit de stop)

USBSn	Stop Bit(s)
0	1-bit
1	2-bit

USBSn = 0

(8 bits de datos)

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

UCSZn2 = 0, UCSZn1 = UCSZn0 = 1.

Los mensajes que se muestran al usuario se guardan en memoria del programa y se acceden a través del puntero Z, el cual es byte addressed y el único cuidado que hay que tener es que los tags a la hora de compilar se definen en worlds (2 bytes), por lo tanto hay que multiplicar por 2.

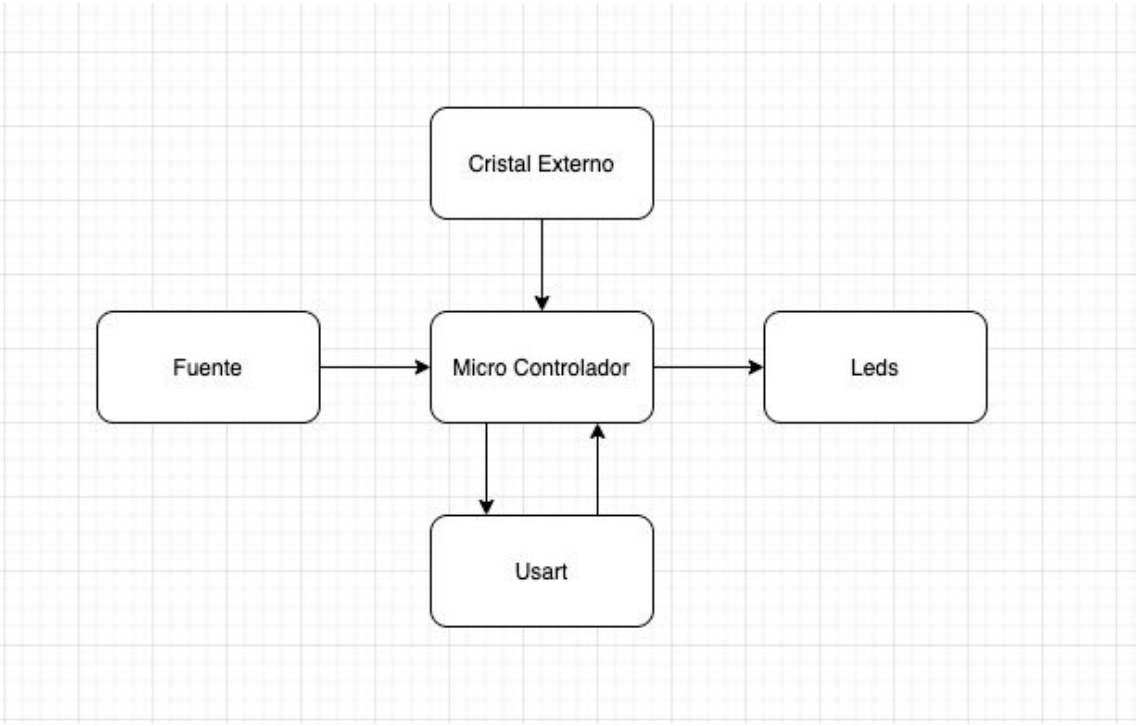
El resto del trabajo fue cubierto en tp anteriores.

- Función de delay, la cual se utiliza para esperar un tiempo antes de enviar los datos de bienvenida por puerto serie.
- Manejo de puertos.

Listado de componentes:

Placa arduino UNO Atmega 328p $659 aprox 10 usd.
4 led (pack de 10) $70.
4 resistencias 220 ohm 1/8w 1% $50.

Diagrama en bloques:



Circuito esquemático:

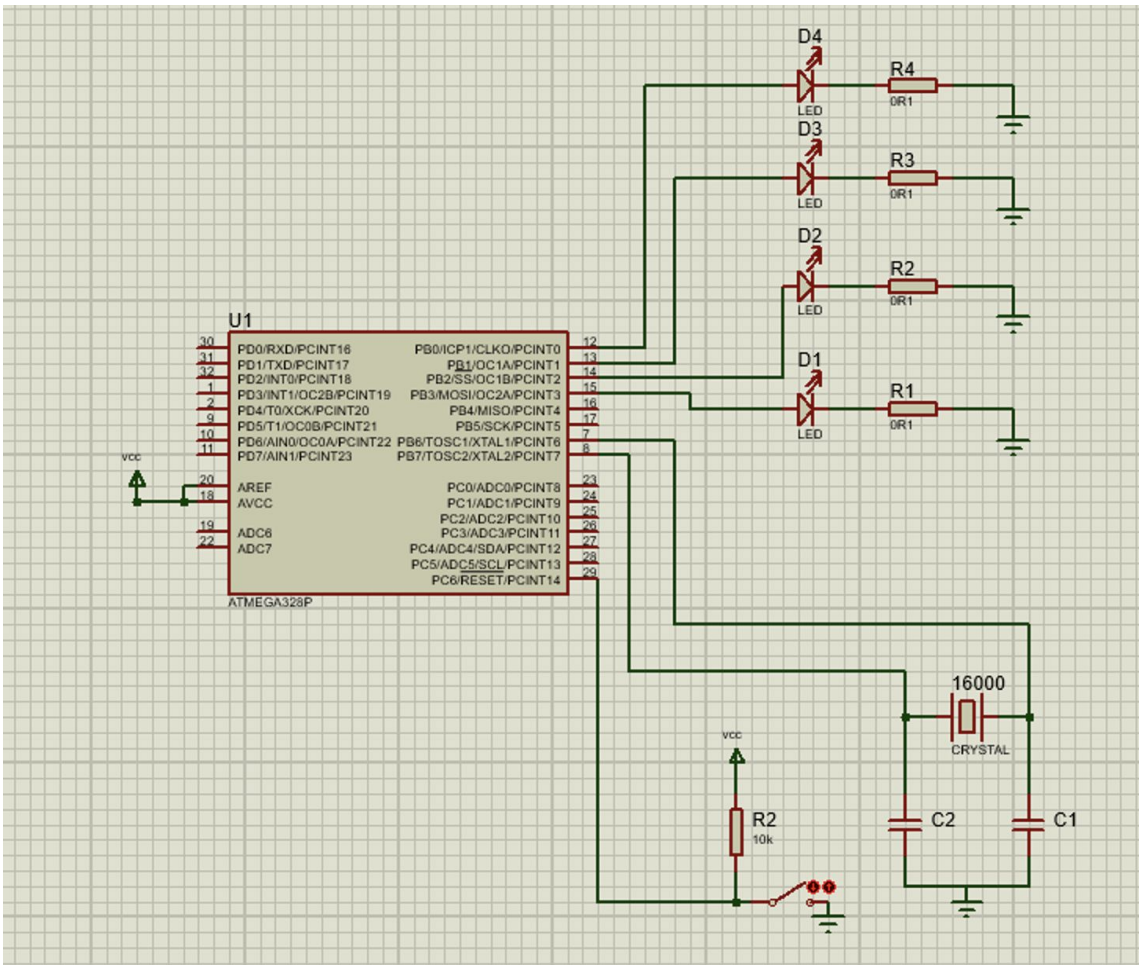
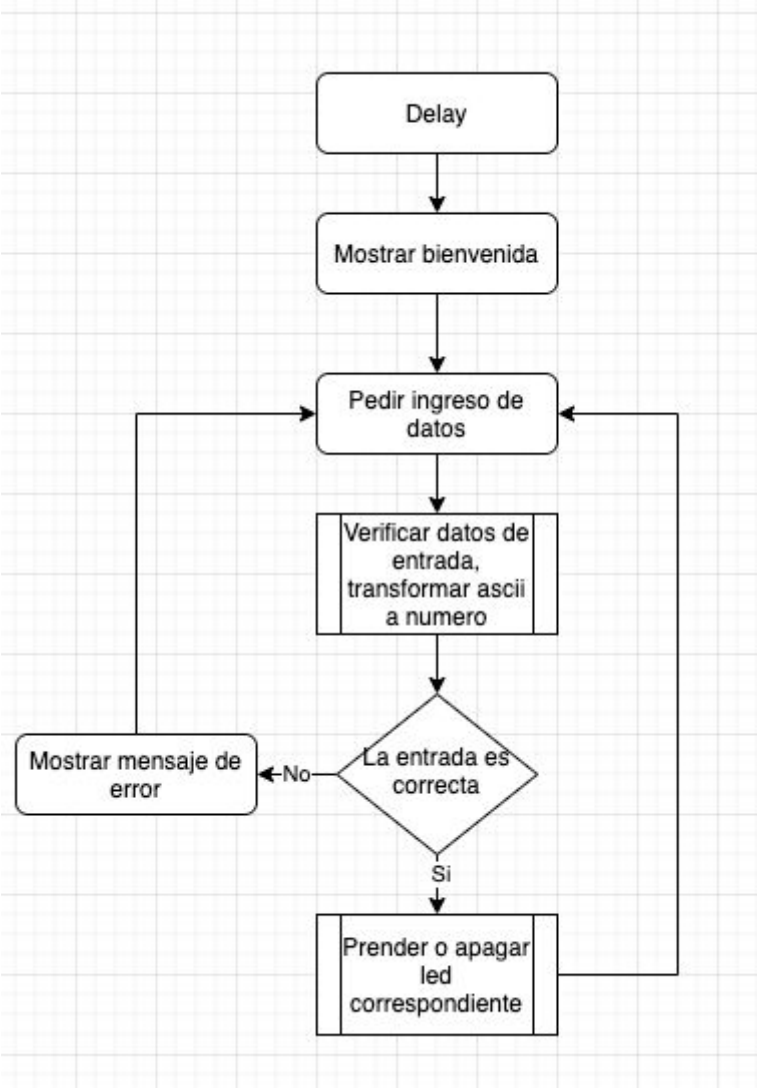
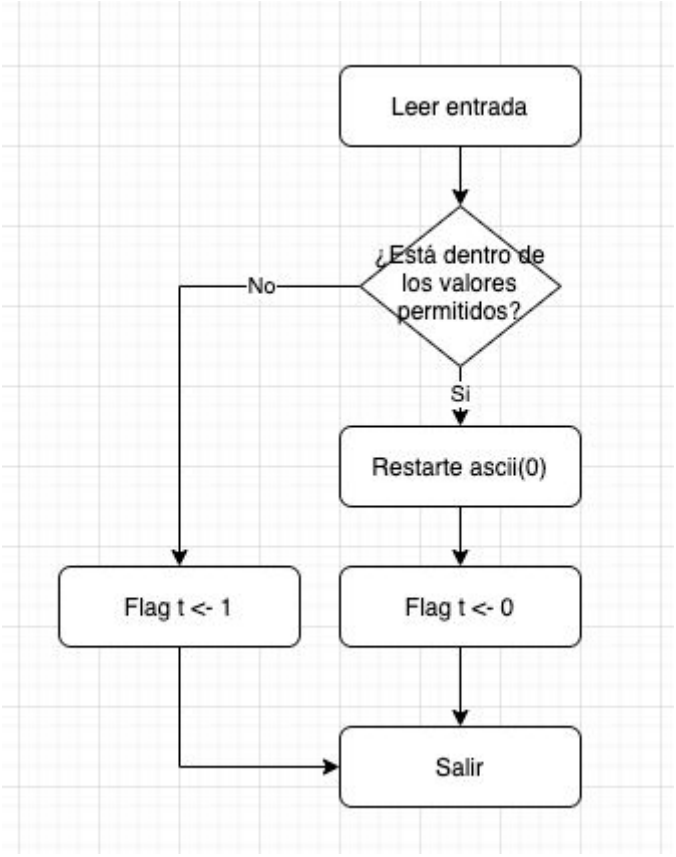


Diagrama de flujos:

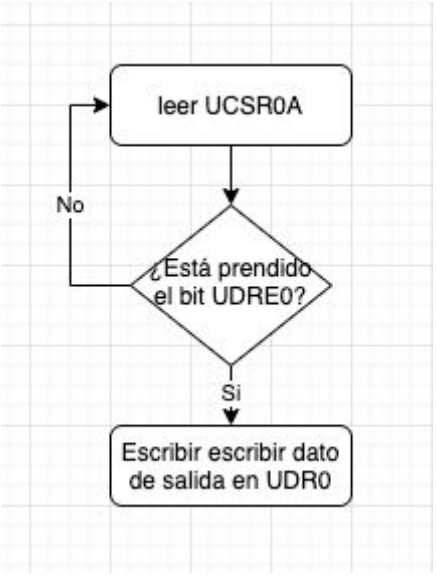
Programa Principal



Verificar Entrada, transformar en ascii



Enviar datos USART



Recibir datos USART.

Código:

```
;
; tp8.asm
;
; Created: 14/08/2020 20:13:45
; Author : cristian
;


.include "m328pdef.inc"

.def dummyreg = r21
.equ FOSC    = 16000000
; Clock prequency
.equ BAUD     = 9600
; Baud/s for usart
.equ UBRRVAL = FOSC/(BAUD*16)-1
; Value in UBRR (datashit)
.equ LOWERINPUTVALUE = '1'
.equ BIGGESTINPUTVALUE = '4'

.cseg
.org 0x0000
                jmp     configuracion


.org INT_VECTORS_SIZE
configuracion:
; ram init
                ldi     dummyreg,low(RAMEND)
; inits stack pointer
                out     spl,dummyreg
;
                ldi     dummyreg,high(RAMEND)
;
                out     sph,dummyreg
;

; port init
```

```
                ldi     dummyreg,0xff
; Port b as output
                out     DDRB,dummyreg
;

; baudrate
                ldi     dummyreg, LOW(UBRRVAL)
; as it was calculated in the
                sts     UBRR0L, dummyreg
;  formula that is in the datasheet
                ldi     dummyreg, HIGH(UBRRVAL)
;
                sts     UBRR0H, dummyreg
;

; Enable receiver and transmitter
                ldi     dummyreg, (1 << RXEN0 | 1 << TXEN0)
; Enable transmission / reception
                sts     UCSR0B,dummyreg
;

; Set frame format: 8data, 1stop bit
                ldi     dummyreg, ( 1 << UCSZ01 | 1 << UCSZ00)
; 8N1
                sts     UCSR0C,dummyreg
;



main:
                call    delay
; Waiting time to connect to a program
                call    delay
;  that establishes a connection with
                call    delay
;  the micro controller
                call    print_greeting_msg
; Pints a gretting
receive:         call    ask_for_input
; Asks for input
```

```
                call    USART_Receive                                   ; Reads input,
lets in r16
                call    trate_input                                     ; Transforms
ascii input in a number, lets that in r17
                brts    error                                           ; input error
                call    toggle_led                                      ; Toggle the
led, reads the number from r17
                jmp     receive                                         ; asks for
another input
error:           call    print_error_input_msg                           ; Shows error
message
                jmp     receive                                         ; Asks for
another input


; Prints the value pointed by z until 0 is reached
print:

                lpm     r16, Z+                                         ; Sets the
value pointed to z to r16, and increments z
                cpi     r16, 0                                          ; Is the
String end reached?
                breq    exit                                            ; Exits
                call    USART_Transmit                                  ; Sends the
value stored in r16
                jmp     print                                           ; loops for
the next character in the string
exit:            ret

; Checks that this is a validate input, and transform ascii to number
; if this fails sets the T flag in status reg, if not, clear T flag in status reg
trate_input:
                cpi     r16, LOWERINPUTVALUE                            ; Verifies
that the input number is bigger than
                brlt    input_error                                     ;   the lowest
accepted
                cpi     r16, BIGGESTINPUTVALUE + 1                      ; Verifies
that the input number is lower than
                brsh    input_error                                     ;   the bigger
accepted
                mov     r17, r16                                        ; copies r16
inbto r17
                subi    r17, '0'                                        ; Transforms
an ascii into a number
                clt                                                     ; Clears t
flag, the input was valid, and could be transformed
                ret
input_error:     set                                                     ; Sets t
flag, the input was invalid
                ret
```

```
; prints greeting message
print_greeting_msg:
                ldi     ZH, high(2*greeting_msg)                        ; points
greeting message
                ldi     ZL, low(2*greeting_msg)                         ;
                call    print                                           ; Prints
                ret

; Asks for a number
ask_for_input:
                ldi     ZH, high(2*ask_msg)                             ; points
asking message
                ldi     ZL, low(2*ask_msg)                              ;
                call    print                                           ; Prints
                ret

; Prints an error
print_error_input_msg:
                ldi     ZH, high(2*error_msg)                           ; Points err
message
                ldi     ZL, low(2*error_msg)                            ;
                call    print                                           ; Prints
                ret

; Transmit the data stored in r16 using USART  (this example is taken from the
datasheet)
USART_Transmit:
                ; Wait for empty transmit buffer
                lds      r17, UCSR0A
                andi     r17, (1 << UDRE0)
                breq     USART_Transmit
                ; Put data (r16) into buffer, sends the data
                sts      UDR0,r16
                ret

; Receibe a byte and stores it in r16   (this example is taken from the datasheet)
USART_Receive:
                ; Wait for data to be received
                lds     r17, UCSR0A
                andi    r17, (1 << RXC0)
                breq    USART_Receive
                ; Get and return received data from buffer
                lds     r16, UDR0
                ret


; turns on the led 1 if it is turned off and vice versa
toggle_led:
                push    r16                                             ; Saves r16
```

```
                push    r18                                     ; saves r18
                ldi     r18, 1                                  ; Sets 1 into
r18
shift:           cpi     r17, 1                                  ; if r17 has
1 stop shifting
                breq    write                                   ; Jmps to
write into port b
                lsl     r18                                     ; left shift
                dec     r17                                     ; decrements
r17 counter
                jmp     shift                                   ; shifts
again
write:           in      r16, PORTB                             ; Reads port
b
                eor     r16, r18                                ; switchs
state on r16
                out     PORTB, r16                              ; loads portb
with the bit switched
                pop     r18                                     ; restore r18
                pop     r16                                     ; restore r16
                ret


; Delay function from tp1
delay:                                                          ; Delay
procedure
                push    r20                                     ; Save the
r20 value in the stack
                push    r21                                     ; Save the
r21 value in the stack
                push    r22                                     ; Save the
r22 value in the stack
                ldi     r22, 82                                 ; 82 * 255 *
255 aprox 8000000/3
loop1:           ldi     r21, 255                               ;
loop2:           ldi     r20, 255                               ;
loop3:           dec     r20                                    ; decrement
r20 by 1
                brne    loop3                                   ; If r20 had
reached 0, z flag would have been seted
                                                                ;   and we
will jump to loop 3
                dec     r21                                     ; The same as
above

                brne    loop2
                dec     r22
                brne    loop1
                pop     r22                                     ; Set r22,
r21, r20 to the same value that
```

```
                pop     r21                                             ; it had
before entere this proc
                pop     r20
                ret                                                     ; go back to
main

greeting_msg: .db   "*** Hola Labo de Micro ***", 10, 13, 0, 0
ask_msg:      .db   "Escriba 1, 2, 3 o 4 para controlar los LEDs", 10, 13, 0
error_msg:    .db   "Error en la entrada", 10, 13, 0
```

Resultado:

Se logra utilizar la interfaz integrada en el microcontrolador.

Conclusiones:

Al usar un Arduino no hace falta utilizar un rs232 to usb ya que viene en la placa.
Tampoco un adaptador de tensión ttl.