Trabajo Práctico N°4
Puerto serie

Objetivo

Establecer comunicación bidireccional serie entre un microcontrolador AVR de 8 bits y una computadora
de escritorio.

Descripción

1) Al encender el microcontrolador, el programa deberá transmitir el texto

**** Hola Labo de Micro ****
Escriba 1, 2, 3 o 4 para controlar los LEDs

El texto de arriba deberá mostrarse en el terminal serie

2) Si en el terminal serie se aprieta la tecla '1', entonces se enciende/apaga el LED 1 (toggle). Si se aprieta la tecla '2', ocurre lo propio con el LED 2 y así lo propio para los cuatro LEDs.

Sugerencias:

- revisar la tabla ASCII
- considerar el agregado de un *delay* en el inicio para poder abrir el terminal serie

Para este trabajo práctico la parte A) es el item 1) y la parte B) es el item 2)

Lectura recomendada

"The AVR microcontroller and embedded systems. Embedded system using Assembly and C". Autores:
MUHAMMAD ALI MAZIDI, SARMAD NAIMI, SEPEHR NAIMI
Capítulo 11

Materiales

4 LEDs
4 Resistencias de 220 Ohms
1 Microcontrolador ATmega8 PDIP
Programador USBasp V3.0
Conversor TTL a USB (leer el comentario en referencia a las placas Arduino antes de comprar). La mayoría de las computadoras modernas carecen de puerto serie, en tanto ofrecen conexión USB (Universal Serial Bus). Si bien no es posible conectar directamente un puerto serie a un puerto USB, sí es posible intercalar un hardware que realiza la conversión en ambos sentidos, como indica la ilustración.
El hardware se llama Conversor USB a serie, y en este caso nos conviene obtener uno que tenga salida TTL, no RS232.
En el caso de utilizar una placa Arduino Uno o Arduino Nano (microcontrolador ATmega328p) la funcionalidad de convertir a USB, ya está disponible, dado que se utiliza el mismo hardware para el puerto serie y para reprogramar la placa. En dicho caso, no se necesita comprar el conversor TTL a USB. Solo debe adecuar el circuito de arriba para que funcione con dicho microcontrolador.

Resumen conceptual

El puerto serie es un periférico que necesita configuración antes de utilizarse. La configuración necesaria
tiene que ver con acordar entre ambos extremos de qué modo se van a serializar los datos. Es decir, como
mínimo:

- A qué velocidad transmitir
- De qué modo sincronizamos lo transmitido con lo recibido
- De qué tamaño es la unidad básica de información comunicable, o “carácter”

Para este Trabajo Práctico, adoptaremos una velocidad de 9600 bits por segundo o *baudrate* . Esto significa
que al dar la orden de transmitir o recibir un carácter, sabemos que los dígitos binarios se van a transmitir
a razón de 1/9600 segundos.
Respecto a la sincronización, adoptaremos el mecanismo más común, la comunicación asincrónica. Esto
significa que no se tendrá una señal de clock que acompañe a los datos, sino una estructura de datos
adicional que enmarca la información y permite sincronizar transmisor con receptor. El enmarcado

consiste en que, para cada carácter se antepone un bit de inicio (*start bit*), que siempre vale “0” y luego de
cada carácter se adiciona un bit de fin (*stop bit*) que siempre vale “1” y coincide con el estado en el que
queda el canal cuando no se transmite nada.
En lo que respecta al largo del carácter, adoptaremos 8 bits de datos. En la literatura esto se encuentra
como 8N1: 8 bits de datos, sin paridad y 1 bit de stop. El concepto de paridad tiene que ver con agregar un
noveno bit a modo de redundancia y así detectar algunos errores del lado de receptor. En este Trabajo
Práctico, como mencionamos, no utilizaremos bit de paridad.
Si sumamos el bit de *start* , los 8 bits de datos y el bit de *stop* , entonces, cada byte de información útil se
convierte en 10 bits comunicados. Entonces, el tiempo de transmisión del byte será 10/9600 segundos.
La configuración de registros requeridos para puerto serie está claramente explicada en la bibliografía
mencionada.
Como actividad adicional, una vez realizado el programa requerido en el enunciado, proponemos
implementar la misma funcionalidad pero utilizando la técnica de interrupciones, en este caso en relación
a la recepción de los datos por puerto serie. Es decir: en vez de consultar permanentemente si hay un dato
nuevo en el puerto serie, la recepción de un byte deberá generar una interrupción de puerto serie, que al
atenderse procesará el dato de la forma correspondiente. Compare ambos loops del programa principal
(con y sin interrupciones).