

# Comunicación serial utilizando puertos I/O rev2

Paul Aguayo S., paguayo@olimex.cl

7 de septiembre de 2005

# Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. La Teoría</b>	<b>3</b>
<b>3. Haciendo que funcione</b>	<b>3</b>
<b>4. Nuestro Código</b>	<b>4</b>

# 1. Introducción

En la mayoría de los proyectos es necesario comunicarse con el resto del mundo. La comunicación serial es relativamente fácil de implementar, y es perfecta para aplicaciones que no requieren de un gran ancho de banda.

En este tutorial se asume que ya tienes tu PIC bajo control, i.e., ya prendiste algún led con tu PIC.

El código necesario es el siguiente:

- Serial\_Calibration.c - Este archivo es utilizado para configurar el timing de la salida serial
- Serial\_Calibration.asm
- Serial\_Calibration.hex
- Serial.c - Envía el carácter 'k' al puerto serial
- Serial.asm
- Serial.hex

Los siguientes pasos son utilizando el P1F16F628 usando el oscilador interno, pero puede ser aplicado a todos los otros PIC's y uC en general.

# 2. La Teoría

El protocolo de comunicación serial RS232 se puede encontrar fácilmente en google, pero otra cosa es hacerlo funcionar.

En este link <http://www.beyondlogic.org/serial/serial.htm> hay una muy buena referencia del protocolo para aquellos que se interesen en entender que están haciendo.

Todo tipo de comunicación serial tiene que realizarse en los tiempos correctos, los cuales se establecen en el protocolo. En este tipo de comunicación no hay una línea clock que cargue los datos. Todo es enviado por una sola línea. Debido a lo anterior ambos dispositivos que deseen comunicarse deben tener la misma velocidad. En este caso la tasa de transferencia será 9600 bps.

$$1/9600 \text{ bit por segundo} = 1.04e-4 \text{ segundos por bit}$$

Esto es equivalente a decir 1 bit cada 104us (micro segundos)



Figura 1: Transmisión serial

La Figura 1 es muy importante. Como se puede ver las señales altas y bajas no corresponden a 1's y 0's respectivamente si no que a -10V corresponde a un 1 y +10V corresponde a un 0 lógico. Ahora se presenta el primer problema. El PIC sólo tiene +Vdd (5V) y nada cerca de -10V ó +10V. Pero aún podemos continuar.

# 3. Haciendo que funcione

Cuando se envía la letra 'r' desde el PIC, el pin serial del computador comienza a enviar una serie de 1's y 0's. En el código ASCII el valor para la letra 'r' es 114 decimal. Esto es 0b.0111.0010 en binario. Entonces de acuerdo a lo anterior cuando el computador envía por el RS-232 la letra 'r' se debe transmitir el complemento de 0b.0111.0010 es decir 0b.1000.1101 para que el PIC vea una 'r'.

## 4. Nuestro Código

Usando el siguiente código, forzamos la comunicación serial incluso usando niveles TTL (0 a 5V)

```
Serial_Out = 1;
rs_wait();
for (j = 0 ; j < 8 ; j++)
{
Serial_Out = !s_char.0;
s_char = rr(s_char);
rs_wait();
}
Serial_Out = 0;
delay_ms(5);
```

Revisemos nuevamente la Figura 1. Asumiendo que partimos con un voltaje bajo ('Mark'). Hay que mantener la línea serial en voltaje alto por al menos un pulso de ancho de manera que el computador pueda ver el bit de partida 'start'. Recuerda, que sólo estamos utilizando 0-5V.

```
Serial_Out = 1;
rs_wait();
```

Ahora transmitimos los 8 bits de datos. La transmisión de los bytes se lleva a cabo desde el bit menos significativo al más significativo (LSB a MSB). Recuerda, el voltaje bajo es 1 y un voltaje alto es 0. Ahora tomamos el LSB, lo invertimos, y lo ponemos en el puerto serial de salida.

```
for (j = 0 ; j < 8 ; j++)
{
Serial_Out = !s_char.0;
s_char = rr(s_char);
rs_wait();
}
```

Ahora que el LSB está en el puerto , lo mantenemos ahí (rs\_wait) durante el tiempo adecuado para 9600bps (104us). Luego enviamos el siguiente bit de datos y así hasta enviar los 8 bits.

```
Serial_Out = 0;
delay_ms(5);
```

Finalizamos la transmisión enviando un bit de parada (stop) poniendo un alto voltaje en la línea. Es necesario que haya un espacio entre la transmisión de cada byte, usualmente funciona con 5ms, lo cual es relativamente un alto tiempo.

```
rs_wait
```

¿Cuánto tiempo es necesario esperar?. Como se dijo anteriormente 104us en total. Si el PIC opera con instrucciones cada 1us, tenemos que esperar 104 instrucciones. La pregunta es ¿Por qué sólo esperamos un ciclo de 6 en la función rs\_wait?

```
void rs_wait(void)
{
int i;
for(i = 0 ; i < 6 ; i++)
}
```

Como se puede ver, hacemos un loop 6 veces. ¿Por qué este valor mágico? el secreto está en el código assembler. Utilizando la función Stopwatch de MPLAB, puedes simular cuanto tiempo toma cada función. Simulando el rs\_wait en MPLAB encontramos los siguientes tiempos:

<b>i</b>	<b>Tiempo de Ejecución Simulado</b>
4	68 us
5	81 us
6	94 us
7	107 us
8	120 us
9	133 us

Puedes ver que usando el oscilador interno del 16F628 puedes obtener un amplio rango de valores aumentando la variable *i* de a 1. Como la función *rs\_wait* tiene más de una instrucción por ciclo, la variable *i* no calza con los microsegundos que necesitamos.

El valor 6 toma 94 us, esto es suficientemente cerca de 104.

Por eso es que escribimos el código *Serial\_Calibration.c* Usando este archivo daremos una rápida y fácil manera de descubrir el tiempo que necesitas. El archivo varía *rs\_wait* desde 1 a 100. Esto te dará efectivamente el timing que necesitas para tu PIC en particular.

- Carga el archivo *Serial\_calibration.hex* en tu 16F628
- Conecta el cable serial desde tu computador al RA1 (Pin18)
- Abre el hyperterminal y configura el puerto con de la siguiente manera 9600 baud 9800-8-N-1
- Alimenta tu PIC y mira la pantalla del hyperterminal

Basados en la posición del carácter 'r' podemos ver el valor de la variable *i* que tenemos que utilizar.

Por ejemplo, se debe ver una línea de caracteres incoherentes en la pantalla del hyperterminal. Si el carácter 'r' aparece en la posición 15, entonces haz el siguiente cambio *i < calibrate\_step* a *i < 15*. Ahora que tienes el valor correcto, deberías poder utilizar la función *rs\_out('e')* (o cualquier otro carácter ASCII) y obtener una 'e' en la pantalla del hyperterminal.

Con la programación adicional de algunas funciones standard de entrada/salida (como *printf* o algo por el estilo) con la llamada a la función *rs\_out* podrías sacar variables por cualquiera de los puertos I/O.

**NOTA:** Las dudas las pueden postear directamente en el foro [www.olimex.cl/phpBB2/](http://www.olimex.cl/phpBB2/)  
**Los archivos necesarios están disponibles en [www.olimex.cl/serialIO.zip](http://www.olimex.cl/serialIO.zip)**