TECNOLÓGICO DE MONTERREY®

Curso:

Laboratorio de Microcontroladores

Práctica #3: Comunicación serial

Manuel Madrigal Valenzuela ID: A01114070

Efraín Duarte López ID: A01113813

Maurcio Capistrán Garza

Noviembre de 2016

Objetivo:

Establecer comunicación serial entre una computadora y un microcontrolador MC9S08QG8 para controlar un motor a pasos. La operación del motor debe ser controlada por secuencia de medio paso. Las acciones que realice el motor estarán dictadas por comandos que se reciban a través del puerto serial. Además, se monitoreará la temperatura del motor la cual será enviada al usuario.

Componentes a usar:

- 1 MC9S08QG8 (microcontrolador)
- 2 ULN2803 (arreglo de transistores Darlington)**[2]**
- Motor a pasos unipolar (PM55L-048)**[3]**
- 1 LM35 (sensor de temperatura) **[4]**
- 1 adaptador de USB a serial (DB9)
- 1 MAX232

Diagrama de conexiones de puerto serial:

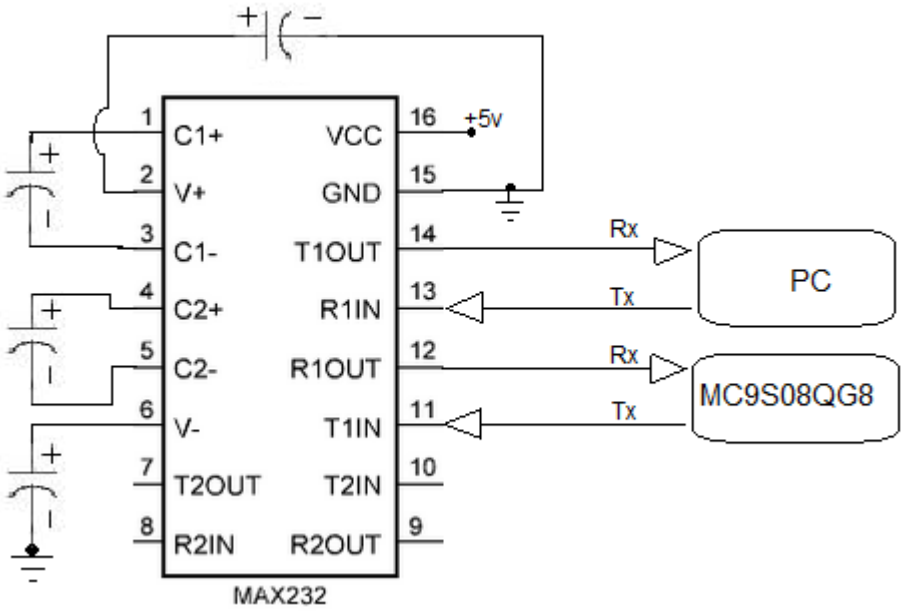
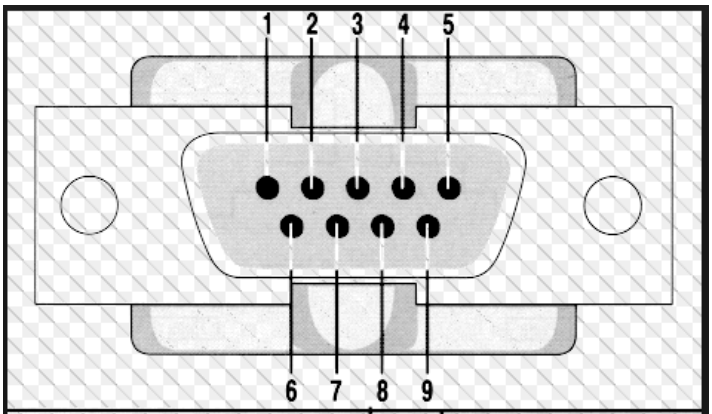


Diagrama de conexión del microcontrolador con la PC a través de un MAX232.

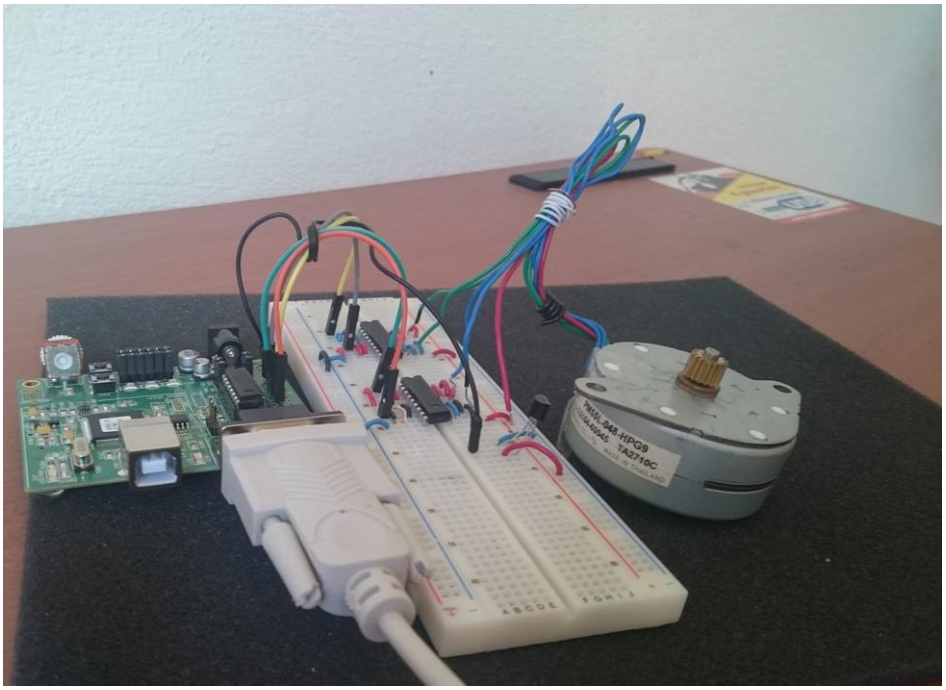


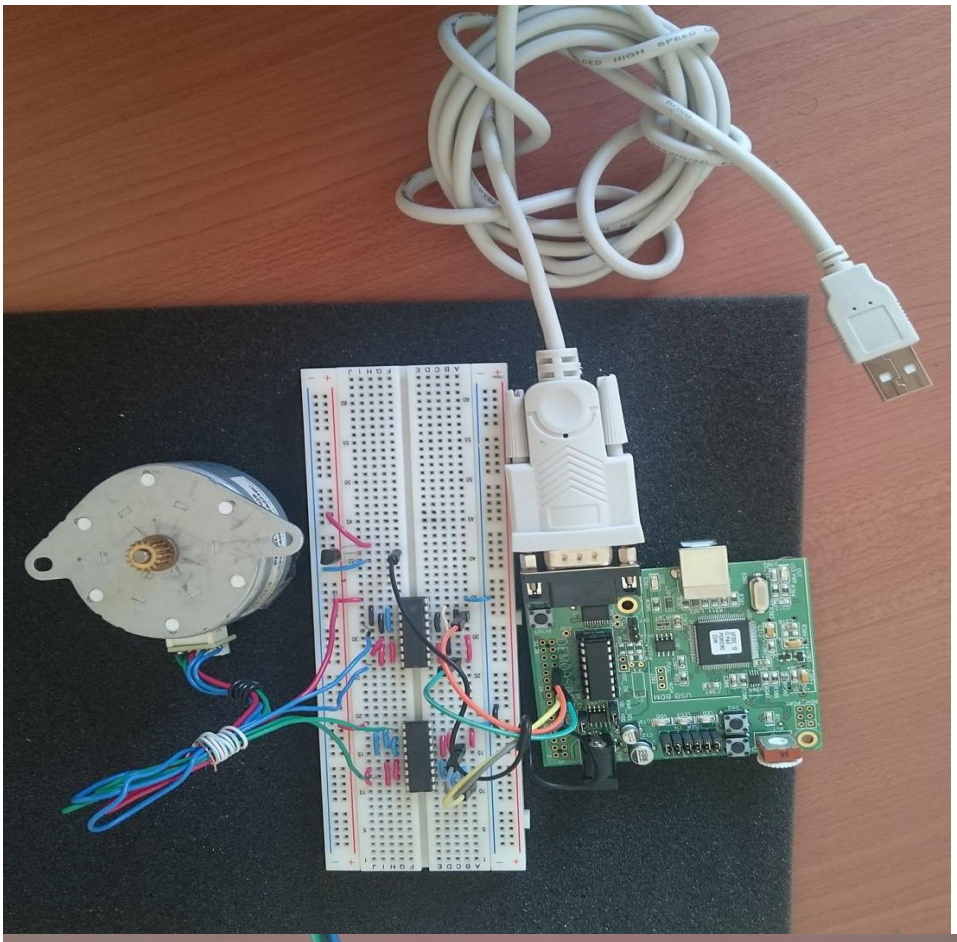
Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

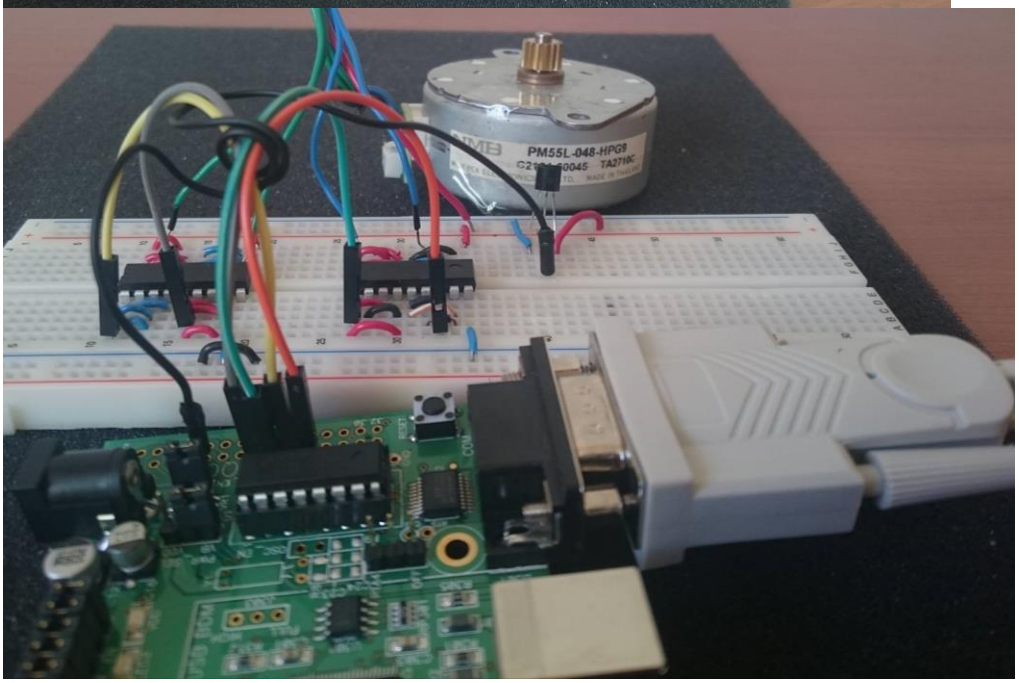
Diagrama de pines del conector DB9.

Los diagramas mostrados anteriormente nos indican como realizar la instalación para poder comunicarnos a través del puerto serial de la computadora al microcontrolador usando un MAX232. Por otra parte, si se tiene el kit de la tarjeta de desarrollo del MC9S08QG8 se puede utilizar el MAX232 que tiene integrado sin la necesidad de hacer la instalación mostrada anteriormente, por que ya está implementada dentro de la tarjeta. Únicamente se tiene que conectar el DB9 (macho) que viene de la PC a la entrada de la tarjeta DB9 (entrada hembra).

Fotos del circuito armado:



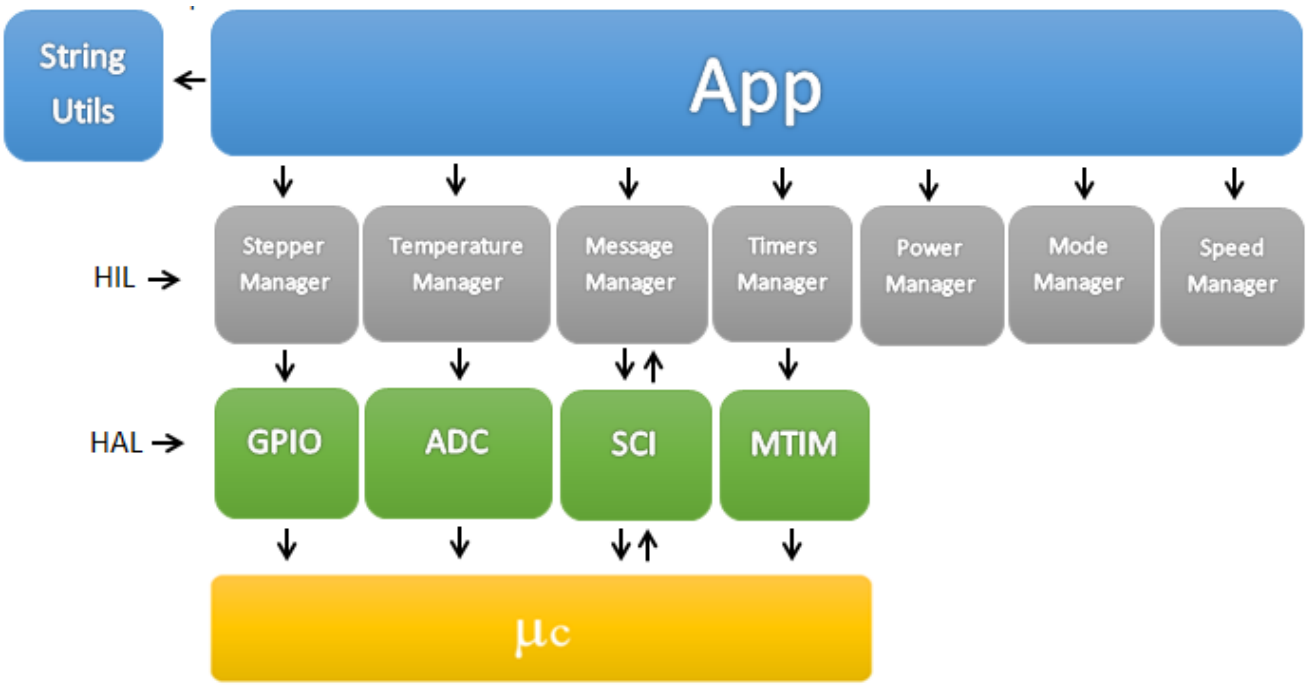




Las imágenes anteriores muestran el circuito montado en un protoboard y el microcontrolador en la tarjeta programable. Además, se muestra el cable DB9 conectado a la tarjeta de desarrollo por el cual se realizó la comunicación entre PC y

microcontrolador. Por otro parte, el sensor de temperatura que está conectado al ADC del microcontrolador se encuentra cerca del motor para detectar la temperatura correctamente.

Arquitectura de software:



Los drivers del SCI (puerto serial) se encargan de configurar el hardware e implementar las interrupciones del SCI para llevar a cabo la comunicación serial de transmisión y recepción.

En el caso de transmisión se inicializa el proceso llamando a la función **SCI_InitTx(void)** la cual se encarga de establecer el baud rate y habilitar la transmisión configurando el registro SCI2 del módulo SCI, haciendo que el pin TxD (pin por el cual transmite datos el microcontrolador) actué como salida. Una vez habilitada la transmisión se verifica que bTxIsBusy tenga valor de "false" para poder llamar a la función **SCI_SendMessage(const unsigned char msg[],u8 size)** la cual se encarga de guardar el mensaje recibido (msg[]) en el buffer para poder enviar el primer carácter y habilitar la interrupción de transmisión. Dentro de la interrupción se verifica que se envíen todos los caracteres restantes, mientras no sea así, la interrupción seguirá habilitada. Cuando termina de enviar todos los caracteres se deshabilita la interrupción y establece que no se está mandando datos por lo tanto asigna a la variable bTxIsBusy el valor de "false".

Por otro parte, el proceso de recepción se inicializa llamando a la función **SCI_InitRx(void)** la cual se encarga de habilitar la recepción y la interrupción de recepción. Dentro de la interrupción se limpia la bandera y llama a la función de apuntador "**typedef void (*FnPtr)(char)"** la cual apunta a una función localizada en el

MessageManager en donde se lleva a cabo todo el proceso de la recepción de caracteres. En esta función se realiza lo siguiente:

- Verifica el tamaño del mensaje que se recibió no sobrepase el límite.
- Ignora los caracteres del mensaje que sobrepasen del límite permitido.
- Se reconstruye el mensaje.
- Prende la bandera de MsgReady para que el usuario sepa cuando leer el dato.

Bitácora de comandos:

Comando	Significado
RPS:nnn.n	Configura la velocidad del motor. Las "**n**" representan las revoluciones por segundo.
DIR:CW	Configura la dirección de giro del motor en sentido horario.
DIR:CCW	Configura la dirección de giro del motor en sentido antihorario.
MOTOR:On	Enciende el motor.
MOTOR:Off	Apaga el motor.
STEP-CW:ggg	Mueve el motor **"g"** grados en sentido horario o antihorario, donde g es un ángulo entre 0 y 999 grados. Este comando es ignorado si el motor está operando en modo continuo.
TEMP-LIMIT:ttt.t	Configura el límite de temperatura del motor a través del puerto serial, donde **"ttt.t"** es la temperatura límite en grados centígrados.

Conclusión:

El funcionamiento del trabajo fue óptimo cumpliendo ampliamente con los requerimientos. Hubo observaciones durante el proceso de esta práctica, entre las principales se encuentran:

- Se puede quebrar el programa si no se verifica que después de un comando enviado a través del emulador de terminal termine con un enter. Es decir, si se envía el comando como esta en la bitácora de comandos agregándole un carácter más al final del que debería de tener y después un enter, el gestor de mensajes lo aceptara mientras no sobrepase el límite de caracteres que se puede enviar, sin embargo al procesarlo no tendrá ninguna relación con los comandos configurados debido a que tendrá un carácter más y el programa no sabrá que hacer. Para evitar esto se verifica que al final de un comando correcto se encuentre un enter en lugar de cualquier otro carácter.
- Si la variable que contiene la cantidad de grados que el motor debe girar se reinicia cada vez que se manda el comando de "STEP-CW: ggg" solamente girara los grados que indicaba el último comando enviado, en lugar de agregarle esa cantidad al valor que ya tenía. Para evitar esto, la variable que contiene la cantidad

de grados se le suma el valor que reciba por un nuevo comando de “STEP-CW: ggg”.

- Se pierden datos de comandos al enviar varios continuos, haciendo que no se ejecuten los comandos en el orden que se enviaron debido a que algunos se pierden en el proceso.

Consideramos que se puede mejorar la implementación del driver de gestión de mensajes para evitar la pérdida de datos al enviar varios comandos continuos, creemos que esto ocurre debido al tiempo que se necesita para almacenar los datos de un buffer a otro buffer.

Bibliografía:

1. .. (2013). Secuencias para manejar motores paso a paso (unipolar) . septiembre 19, 2016, de. Sitio web: http://server-die.alc.upv.es/asignaturas/lsed/2002-03/MotoresPasoaPaso/ftomotpap.htm.
2. .. (2015). ULN2803 Darlington Transistor Arrays. septiembre 19, 2016, de Texas Instruments Sitio web: http://www.ti.com/lit/ds/symlink/uln2803a.pdf.
3. .. (2004). PM55L-048. septiembre 22, 2016, de Minebea Motor Manufacturing Corporation Sitio web: http://www.nmbtc.com/pdf/motors/PM55L048.pdf.
4. .. (agosto, 2016). LM35 Precision Centigrade Temperature Sensors. noviembre 2, 2016, de Texas Instruments Sitio web: http://www.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=lm35&fileType=pdf