UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Monitor gráfico de entradas/salidas de un microcontrolador

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Villalba Felip, Ricardo

Tutor: Martí Campoy, Antonio

2015/2016

Resumen

La programación de un microcontrolador es relativamente sencilla gracias a las facilidades que nos ofrecen los entornos de programación existentes. Sin embargo, una vez escrito el código y transferido al dispositivo, sería de mucha utilidad poder comprobar de una forma gráfica que el microcontrolador actúa o no según lo deseado.

En el proyecto planteamos el desarrollo completo de una interfaz gráfica que nos permite conocer el estado de las entradas y salidas del microcontrolador mientras se está ejecutando un programa realizado por el usuario. Mediante la creación de dos aplicaciones, una para el dispositivo microcontrolador y otra para el PC, que se comunican entre ellas vía USB utilizando un canal serie virtual (EIA-232) con un protocolo también definido en este trabajo, conseguimos con resultados positivos los objetivos de este proyecto.

Este trabajo de ingeniería puede resultar muy interesante tanto para el desarrollo como en la puesta en marcha de sistemas empotrados, también llamados sistemas embebidos, y que son extensamente utilizados actualmente.

Palabras clave: microcontrolador, EIA-232, msp430, sistemas empotrados.

Abstract

Microcontroller programming is relatively simple thanks to the facilities provided by the existing programming environments. However, once the code is written and transferred to the device, it would be very useful to check graphically if the microcontroller behaves like we desire or not.

In the project we propose the complete development of a graphical interface that allows us to know the status of the microcontroller inputs and outputs while it is running with a user program. By creating two applications, one for the microcontroller device and another one for the PC, which communicate with each other via USB using a virtual serial channel (EIA-232) with a defined protocol also in this work, we get positive results with the objectives of this project.

This engineering work could be very interesting in both development and implementation of embedded systems, which are widely used today.

Keywords: microcontroller, EIA-232, msp430, embedded systems.

Agradecimientos

A la Universidad Politécnica de Valencia, por ser una entidad con mayúsculas.

A todos los profesores que he conocido; por su labor, interés y dedicación. En especial, a Robert Fuster, Mª Isabel Giménez, Elena Vázquez, F. José Romero, Carlos Herrero y J. Vicente Benlloch, porque fueron mis primeros profesores, me ofrecieron confianza y me quitaron el miedo a empezar una carrera.

También a Toni, mi tutor, porque sin haber tenido una sola clase con él, parece que sea mi *profe* particular de toda la vida. Espero saber reflejar su esfuerzo en este trabajo.

A todos los compañeros que he tenido el placer de conocer en las distintas estaciones por las que he pasado al realizar este gratificante viaje.

A Gabriela y Gema, mi esposa e hija. Han sido el motor que me ha empujado a comenzar esta aventura, el corazón que no ha dejado de palpitar para recorrerla por completo, y la sonrisa que ha hecho suaves las pendientes que hemos superado.

Tabla de contenidos

Índice de figuras

1. Introducción

1.1 Motivación

Como es bien sabido, los microcontroladores (µC o MCU) son circuitos integrados que incluyen en un solo chip CPU, memoria, y periféricos como los puertos digitales de entrada/salida y temporizadores, entre otros. Debido a su elevado nivel de integración, coste reducido y bajo consumo, una gran mayoría de los equipos electrónicos actuales incluyen µCs: equipos de comunicaciones, electrodomésticos, vehículos, etc.; por ejemplo, un automóvil moderno incluye decenas de ellos, desempeñando todo tipo de tareas de control, monitorización y seguridad. En la industria, igualmente, se ha introducido a todos los niveles, desde un simple sensor hasta una compleja máquina herramienta o robot industrial.

Llamamos sistemas empotrados a los circuitos electrónicos que contienen un sistema computador, por ejemplo un microcontrolador, y que forman un conjunto desarrollado para realizar una aplicación concreta: mando a distancia, sensor de caudal, semáforo etc. Para el desarrollo de estos sistemas se realizan proyectos en los que se programa el microcontrolador de forma que actúe según las necesidades de la aplicación.

En la fase de desarrollo de un producto así como en la puesta en marcha, se realizan test de funcionamiento para comprobar si el sistema reacciona según lo esperado. Cuando en un test existe un fallo de secuencia, sin causa aparente, no es sencillo determinar dónde se ha producido ni cuál es el origen del problema. Se necesita una información del estado actual de cada una de las señales del µC para determinar tanto el problema como la posible solución al mismo. Además, en la mayoría de casos la persona que instala el equipo no es la misma que lo ha programado y en caso de disponer de esta información, puede enviarla para su estudio al centro de desarrollo.

La motivación de este proyecto es la creación de una ayuda visual que proporcione la información necesaria para:

- Comprobar el estado de las entradas y salidas (E/S) del microcontrolador.
- Reducir el tiempo de desarrollo del producto.
- Colaborar con la puesta en marcha y los test de funcionamiento.

Se ha elegido para desarrollar el trabajo un sistema empotrado de la plataforma de Texas Instruments para experimentación, la tarjeta *Experimenter's Board*, que incluye el microcontrolador MSP430FR5739.

1.2 Objetivos

El principal objetivo del proyecto es conseguir monitorizar el estado de los puertos y canales de E/S de este microcontrolador de una forma visual en un PC, y que resulte cómoda y sencilla de entender. La consecución de este objetivo depende de algunos objetivos parciales, como son:

- Comprender y programar la configuración interna de los microcontroladores de la familia MSP430 de Texas Instruments.
- Estudiar y diseñar un protocolo de comunicación que permita la transferencia de los datos entre el ordenador y el microcontrolador.
- Crear una librería para el µC que realice la monitorización y envíe la información al PC.
- Crear una aplicación para el PC que sea capaz de representar gráficamente la información recibida del microcontrolador.

1.3 Requerimientos

1.3.1 En la aplicación del microcontrolador

Diseñar el código de la librería para el microcontrolador de manera que el usuario pueda incluirla en su proyecto de forma sencilla.

La inclusión de esta librería debe interferir mínimamente en el funcionamiento del programa de usuario.

Comprobación de la comunicación entre el microcontrolador y el PC.

Adquirir y transmitir a intervalos periódicos el estado de las entradas y salidas del microcontrolador.

Recibir órdenes de *start* y *stop* para iniciar o detener la transmisión de datos hacia el PC.

Añadir algún tipo de seguridad en la comunicación, como longitud de trama o *checksum.*

1.3.1 En la aplicación para el PC

Representar gráficamente el elemento analizado, la tarjeta *Experimenter's Board*.

Adaptar de forma automática la representación gráfica a la configuración programada en el µC mediante símbolos o colores.

Posibilidad de configurar la comunicación (puerto, velocidad, etc.) fácilmente.

Incluir en la interfaz elementos para activar y desactivar la comunicación.

Representar de una manera gráfica y sencilla los estados de las entradas y salidas digitales.

Representar las entradas analógicas de forma gráfica y también numérica.

Permitir el ajuste en la representación de los valores analógicos, por ejemplo con un offset y factor de amplificación.

Incluir un sistema de grabación de datos para registrar los estados de las entradas y salidas del microcontrolador en un fichero.

2. Hardware

2.1 Componentes

Disponemos para este trabajo de un microcontrolador de la familia MSP430™ (*Mixed-Signal Proccessors*) de Texas Instruments del que veremos sus principales características. Este chip va montado en una tarjeta electrónica diseñada por la misma marca, que incluye una serie de dispositivos que simplificarán y complementarán el desarrollo del proyecto.

También se dispone de un cable de comunicación estándar USB a USB *mini* para comunicar el microcontrolador con el PC.

2.2 El microcontrolador MSP430FR5739

Texas Instruments ha desarrollado una amplia gama de microcontroladores con características variadas según el tipo de aplicación a la que son destinados. En nuestro caso se trata de un procesador de 16 bits, basado en la arquitectura RISC (*Reduced Instruction Set Computer*), y con un consumo ultra bajo. Está optimizado para prolongar el tiempo de uso con batería en aplicaciones portables gracias a los 7 modos de trabajo disponibles para bajo consumo y al uso de una memoria no volátil (FRAM) que combina la flexibilidad y rapidez de la SRAM y la estabilidad y fiabilidad de la memoria FLASH.

En el anexo A se pueden apreciar sus características con todo detalle. En cuanto a la funcionalidad, incluye 5 puertos de E/S digitales, 5 temporizadores, 3 canales DMA, 16 comparadores y 14 canales para entradas analógicas como se observa en la figura 2-1.

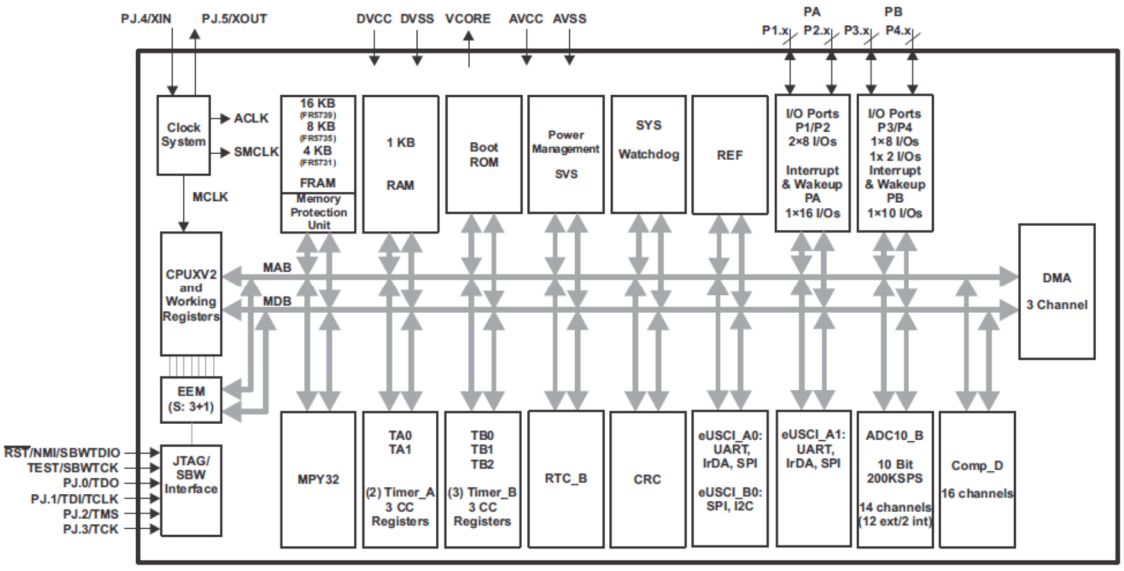


Figura 2-1. ***Bloques funcionales***

2.3 La tarjeta MSP-EXP430FR5739

Texas Instruments también ha creado los denominados *LaunchPad development kits* y *Experimenter boards* que son placas de circuito impreso con uno de sus microcontroladores y algunos componentes añadidos para realizar prototipos y poder crear aplicaciones embebidas. Vamos a enfocar nuestro trabajo en torno a una de estas tarjetas para experimentación. En concreto, la tarjeta MSP-EXP430FR5739 que ya dispone, además del microcontrolador del mismo nombre, de estos componentes:

- Acelerómetro de 3 ejes.
- Resistencia NTC *(Negative Temperature Coefficient).*
- 8 indicadores *led*.
- 2 pulsadores.
- Conexión USB.

La utilizamos exactamente igual que en una aplicación de usuario, pero aprovechando algunas de estas características para no tener que realizar montajes hardware añadidos.

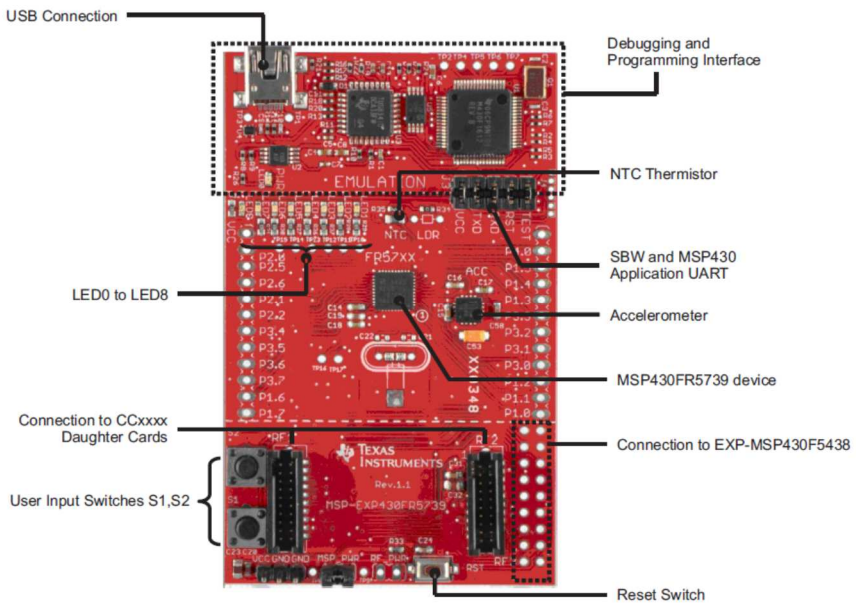


Figura 2-2. ***Experimenter's Board***

Como información al lector, esta tarjeta lleva una conexión para módulos de expansión inalámbricos (Wifi, Bluetooth...) que aumentan la conectividad con el entorno. En este proyecto la comunicación se realizará a través de cable, pero no se descarta un uso futuro de estos módulos para instalaciones en altura o de difícil acceso. El precio de venta público de esta tarjeta es de 35$.

3. Diseño de la solución

3.1 Primeros pasos

Para conseguir los objetivos del proyecto necesitamos fijar las bases de trabajo:

- El entorno de programación a utilizar en la parte del microcontrolador.
- El entorno de programación a utilizar en la parte del PC.
- La comunicación entre ambos.

3.1.1 Entorno de programación del µC

Existen varios entornos disponibles para la programación de estos dispositivos, entre los que cabe señalar:

IAR Systems, *Energía*, *Code Composer Studio*™ y *MSP430-GCC-OPENSOURCE*.

Después de revisar las características de cada uno, instalar y probar su funcionamiento, se ha elegido ***Code Composer Studio v6*** (CCS) en su versión gratuita por admitir hasta 16 kilobytes de código de programa (el máximo disponible en nuestra tarjeta) y por estar integrado con el IDE de Eclipse.

IAR Systems ofrece una versión gratuita, pero con una limitación de tamaño de código de 8KB.

Energía es un entorno idéntico al de *Arduino*, que es muy apropiado para iniciarse en la programación de microcontroladores pero poco útil, en mi opinión, para aplicaciones profesionales.

MSP430-GCC-OPENSOURCE notifica que se puede esperar un 15% de incremento en el tamaño del código compilado, además de un ligero aumento del tiempo de ciclo en las aplicaciones, con respecto al entorno *Code Composer Studio*™.

3.1.2 Entorno de programación del PC

La programación en el PC se ha optado por realizarla con el lenguaje de programación *C#* de la plataforma *.NET* en el entorno de programación *Visual Studio*, aunque en este caso no hay grandes diferencias en cuanto a diseño y manejo con otras plataformas como *NetBeans* o *Eclipse*. *C#* deriva de *C* y *C++*. Se eliminan los punteros, las macros, la herencia múltiple y los ficheros de cabecera. La gestión de memoria es automática, lo cual quiere decir que el desarrollador sólo ha de preocuparse de crear un objeto cuando lo necesite pero no de eliminarlo, ya que esta tarea queda a cargo del *garbage collector*.

C# es un lenguaje elegante, sencillo, orientado a componentes y ofrece un manejo de tipos seguro.

La aplicación realizada es principalmente gráfica y dinámica, lo cual es una oportunidad para el autor del proyecto de experimentar y aprender con otro entorno y lenguaje de programación distinto a los ya utilizados en las asignaturas del grado en Ingeniería Informática.

3.1.3 La comunicación

La tarjeta lleva implementada una conexión hardware USB para programar y depurar. Sobre esta conexión se crea un puerto *com* virtual que permite realizar la comunicación como si fuese una conexión serie real. Al conectar el cable USB al ordenador, si ya tenemos instalado el entorno CCS, automáticamente lo reconoce y se instala el driver apropiado. Y una vez instalado se genera un puerto de comunicaciones (COMxx) en el PC, donde xx es el número de puerto, que puede variar según el ordenador al que esté conectado. Este es un punto a tener en cuenta en nuestra aplicación para el PC, en la que se debe poder configurar el puerto de comunicación serie. En el lado del microcontrolador, este com virtual se convierte en un com físico que se conecta a los pines de transmisión y recepción de la UART, uno de los dispositivos internos encargado de las comunicaciones serie.

3.2 Planteamiento

En principio, el planteamiento funcional parece sencillo. El µC ejecuta su aplicación (del usuario) leyendo las entradas y activando o desactivando salidas y cuando recibe una petición del PC le envía los datos del estado de sus puertos de E/S. El PC, a su vez, actualiza estos datos, los transforma de manera adecuada y los presenta en pantalla.

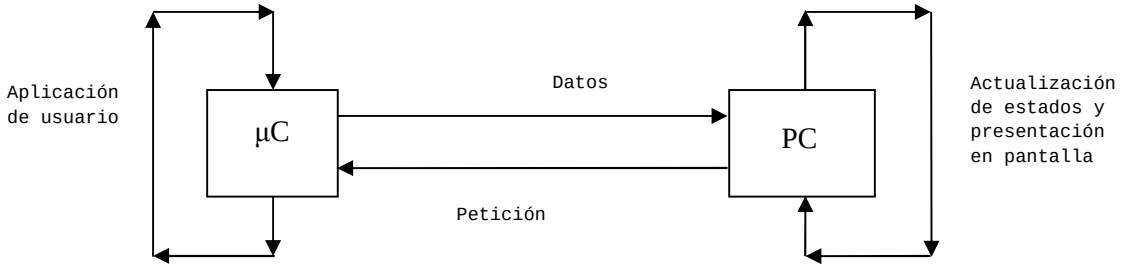


Figura 3-1. ***Esquema funcional inicial***

Esta es la idea básica del proyecto, pero no hay que olvidar que todo esto se va a realizar mientras está funcionando un programa de usuario, al que no debe afectar. Para ello se ha creado una librería (a partir de ahora *monit*) que será incluida en el código de programa del microcontrolador, y que será llamada solo una vez al inicio del mismo.

En primer lugar, la visualización de los cambios en las E/S debe ser lo más próxima posible al momento en que se sucede, por tanto, el PC debe estar continuamente refrescando los datos en pantalla, y el µC tiene que transmitir estos datos de forma periódica, varias veces por segundo. Para reducir la sobrecarga en la ejecución de código, tanto en el microcontrolador como en el PC, y al mismo tiempo reducir la carga en el canal serie virtual, no es necesario que el PC envíe continuamente la petición de datos. Basta con emitir una señal de *start* y el microcontrolador enviará sistemáticamente paquetes de información con un periodo que haga inapreciable a simple vista, el desfase entre el dispositivo y el PC.

En el µC se utiliza un temporizador que genera una interrupción cada 80 ms, y en cada interrupción es cuando se realiza la transmisión de la información, siempre que el PC haya enviado la orden de *start*.

Cuando dejamos de visualizar la aplicación, no es necesario que el dispositivo siga enviando los datos, así que también tenemos la opción *stop* para detener la transmisión.

4. El protocolo

4.1 Órdenes y datos

Tal y como se ha comentado anteriormente, se necesita un pequeño protocolo para la comunicación de los 2 equipos, microcontrolador y ordenador. El dispositivo debe atender las peticiones del PC consumiendo el menor tiempo posible, es decir, solo va a recibir unos pocos comandos, respondiendo a cada uno de ellos de manera inmediata. La solución más sencilla y eficaz adoptada es transmitir un solo byte por cada orden. La definición de estos se puede ver en la figura 4-1.

Comando	Descripción	Byte enviado	ASCII
Test	Comprobación de la comunicación	0x30	"0"
Conf	Petición de configuración	0x31	"1"
Start	Inicio de transmisión de datos E/S	0x32	"2"
Stop	Fin de transmisión de datos E/S	0x33	"3"

Figura 4-1. ***Tabla de comandos PC→µC***

La orden *Test* se envía desde el ordenador cuando se inician las comunicaciones, y provoca como respuesta un paquete de datos conteniendo la identificación del microcontrolador y 62 caracteres ASCII, desde el código 64 (‘@’) hasta el 126 (‘~’), siendo comprobados en la aplicación del PC para confirmar el correcto funcionamiento de la comunicación.

Si el resultado es satisfactorio, el siguiente paso es pedir la lectura de la configuración del microcontrolador. El comando para esta función es *Conf*, que transmite el código ASCII ‘1’, y genera la lectura de los registros de configuración de los 5 ports de E/S del microcontrolador y que es enviada al PC para la presentación gráfica en pantalla.

Y una vez recibida la configuración, solo queda enviar el comando *Start* para que el dispositivo comience a recoger periódicamente la información de sus E/S y la devuelva al PC en un formato que veremos a continuación.

Bien a petición nuestra, o por finalización de la aplicación, se emitirá desde el PC el comando *Stop*, que interrumpirá el envío de los datos sin interrumpir el funcionamiento del programa de usuario del microcontrolador.

Como vemos, las órdenes que envía la aplicación desde el PC obtienen mensajes de respuesta del microcontrolador. Estos mensajes de respuesta tienen un formato determinado, que analizamos en el siguiente apartado.

4.2 Formatos del mensaje

Se ha diseñado un formato *ad hoc* para los mensajes de respuesta que enviará el dispositivo hacia el ordenador. Todos los códigos que se envían se traducen a caracteres ASCII, esto es, si tenemos que enviar un byte con el valor 0x2C lo que se transmite realmente es el carácter '2' y el carácter 'C'. Por tanto, se están enviando dos bytes para transmitir el valor de uno solo. Tradicionalmente en la industria se hace así para poder usar consolas de comunicación y que un operador humano pueda entender los mensajes.

El primer byte de cada mensaje indicará si se trata de una respuesta a alguno de los comandos de comunicación y configuración, o por el contrario si se trata de un paquete de datos ("@" o "#" respectivamente). Tenemos entonces dos tipos de mensajes de respuesta del dispositivo microcontrolador, que nombraremos como *mensaje de órdenes* y *mensaje de datos*.

4.2.1 Mensaje de órdenes

Este mensaje lo genera el microcontrolador después de una orden emitida desde el PC. Va precedido por el carácter '**@**', y si el mensaje es el resultado de un comando *Test* o *Conf*, a continuación le seguirán 4 bytes que serán el nombre del comando al que se responde. Y después le seguirá el mensaje de respuesta. Si ha habido algún error de comunicación, o el comando no es reconocido, estos 4 bytes ASCII serán "Err_", indicando así el fallo ocurrido.

1 byte	4 bytes	0 … 76 bytes	1 byte
Tipo msj. '@'	Orden Test Conf Err_	Mensaje	EOL (0x0A)

Figura 4-2. ***Formato del mensaje de órdenes***

El último byte enviado es el carácter *End Of Line* (EOL), que indica el final del mensaje. La longitud de cada uno de los mensajes puede ser distinta y aunque hay otras maneras de tratarlo, la forma escogida para evitar equivocaciones en la lectura es capturar el *frame* completo hasta recibir este byte.

La orden *Start* enviada desde el PC no genera un mensaje de respuesta, pero sí que provoca que el microcontrolador comience a enviar paquetes de datos hacia el ordenador con la información relativa al estado de sus entradas y salidas. Estos paquetes tienen un formato definido, que podemos ver a continuación.

4.2.2 Mensaje de datos

Si el mensaje es de datos, el byte inicial será '#' seguido de los 5 bytes con el status de los ports de entradas y salidas digitales, y a continuación los bytes que indican el valor de las entradas analógicas, si las hubiere.

1 byte	10 bytes	0 … 60 bytes	2 bytes	1 byte
Tipo msj. '#'	E/S digitales	Entradas analógicas	Check Frame	EOL

Figura 4-3. ***Formato del mensaje de datos***

Después de la información de las E/S del microcontrolador, se envía un valor que indica la cantidad de bytes del frame, es decir, la longitud del mensaje, sirviendo así como chequeo de seguridad de la integridad del paquete.

Podemos ver algunos ejemplos:

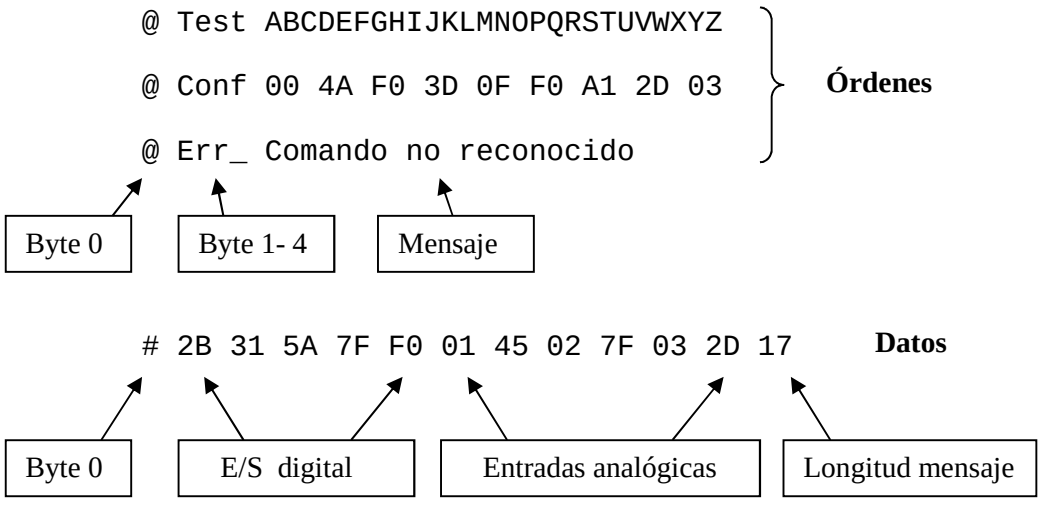


Figura 4-4. ***Formato de los mensajes µC → PC***

En el caso de las entradas analógicas, se utilizan dos bytes; uno para indicar el número de canal y otro para el valor de dicho canal. Recordemos que realmente se enviarán 4 bytes, por la conversión a caracteres ASCII.

La identificación del tipo de paquete en el byte 0 nos permite discriminar rápidamente el mensaje de datos para atenderlo directamente en una rutina dedicada.

4.3 Tiempos

En esta sección se calcula la cantidad de tiempo necesario para enviar un *frame* completo desde el microcontrolador hacia el PC. La configuración de la comunicación se ha hecho con unos valores por defecto: 9600 baudios, 8 bits de datos, sin paridad y con 1 bit de stop.

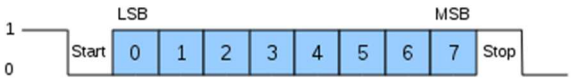


Figura 4-5. ***Secuencia de bits en EIA-232***

Al utilizar un bit de start, los 8 bits de datos y un bit de stop, significa que por cada byte enviado consumimos 10 bits en la transmisión.

Si tomamos como ejemplo el paquete de datos de la figura 4-4, contamos veinticinco caracteres a enviar y uno de final de línea (EOL) , es decir, veintiséis bytes, el tiempo necesario para transmitirlos es el expresado en la fórmula de la figura 4-6, siendo *n* la cantidad de bits a enviar y V_{tx} la velocidad de transmisión.

$$26\ bytes \times 10\ bits = 260\ bits$$

$$t_{tx} = n\frac{1}{V_{tx}} = \frac{260\ bits}{9600\ baud} = 0.027\ s$$

Figura 4-6. ***Tiempos de transmisión***

Por tanto, enviar este *frame* tiene un coste de 27 milisegundos . Como la longitud del mensaje es variable, dependiendo de la cantidad de canales analógicos configurados, los tiempos de transmisión pueden oscilar entre 13 ms y un máximo de 74 ms, tiempos menores que el periodo de interrupción y actualización de datos del microcontrolador.

No obstante, en aplicaciones donde exista un uso exhaustivo del microcontrolador y los tiempos de ciclo estén muy ajustados, se puede incrementar la velocidad de transmisión hasta un máximo de 115200 baudios, reduciendo así hasta en 10 veces el tiempo anteriormente calculado.

5. Aplicación en el µC

5.1 Diagrama de flujo

Se ha desarrollado un diagrama de flujo donde se expone de una manera compacta el funcionamiento del programa en el microcontrolador.

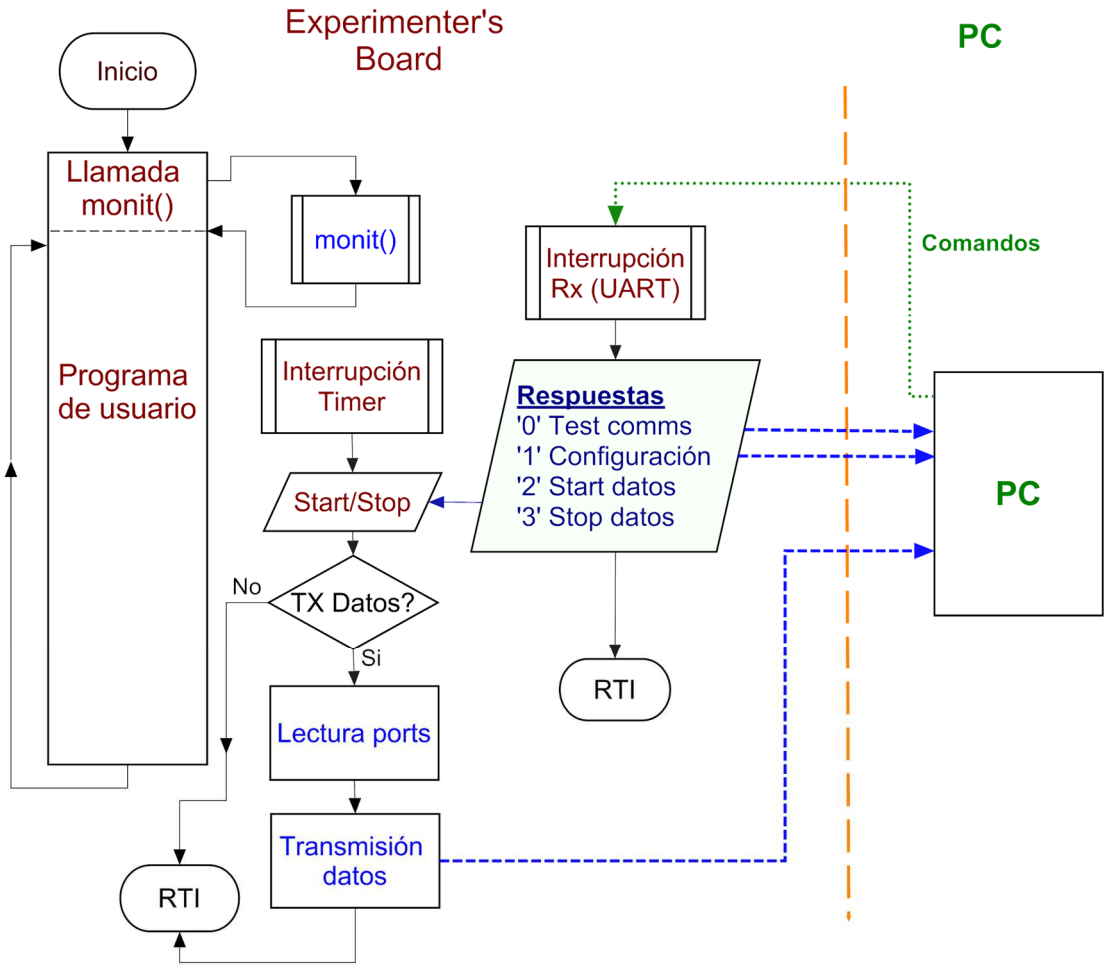


Figura 5-1. ***Diagrama de flujo***

5.2 Configuración del microcontrolador

Nuestro monitor tiene, necesariamente, que inmiscuirse en la actividad del microcontrolador, pero de manera que sea mínima la interferencia que pueda provocar en el funcionamiento del programa de usuario. Se necesita tomar el control de dos elementos funcionales para poder observar y exportar la configuración y los cambios en algunos registros. En concreto, hace falta controlar uno de los cinco temporizadores existentes y uno de los dos módulos USCI (*Universal Serial Communication Interface*) de los que dispone el microcontrolador.

El temporizador (*timer*) es necesario para provocar una interrupción periódica que dirija el flujo de programa hacia los procedimientos desarrollados en la librería *monit*, encargada de recopilar la información del estado de las entradas y salidas del µC. Y para transmitir esta información al exterior se debe utilizar el módulo USCI, que se encarga de adaptar la comunicación al estándar comentado en el apartado 3.1.3.

5.3 Clock (señal de reloj)

La familia MSP430 se diseñó pensando en bajo coste y sobre todo en bajo consumo. Este es el motivo por el cual todos sus µCs tienen varias opciones de configuración del sistema de reloj; sin componentes externos, con uno o dos cristales de cuarzo, o con resonadores externos. No nos vamos a extender en esta parte ya que el bajo consumo no es el objetivo de este trabajo. Pero sí que vamos a comentar que el chip dispone de hasta cuatro fuentes de *clock*:

- XT1CLK Oscilador al que se le pueden conectar cristales externos, tanto de baja frecuencia (32768 Hz) como de alta (de 4 a 24 MHz), para conseguir mucha precisión.
- VLOCLK Oscilador interno de muy baja frecuencia, típicamente 10 KHz.
- DCOCLK Oscilador interno controlado digitalmente. Tres frecuencias seleccionables por software. Utilizaremos esta fuente para nuestro trabajo.
- XT2CLK Oscilador opcional de alta frecuencia con conexión de elementos externos.

Estas fuentes, si se usan, pueden ser canalizadas para crear las cuatro señales de *clock* del sistema:

- ACLK *Clock* auxiliar. Puede ser seleccionado por módulos periféricos individuales, y procede de alguna de las cuatro fuentes anteriores (seleccionables por software).
- MCLK *Clock* principal. Es usado por la CPU y el sistema. También puede venir de cualquiera de las fuentes.
- SMCLK *Clock* del subsistema principal. Puede ser seleccionado por módulos periféricos individuales, y puede venir de cualquiera de las cuatro fuentes anteriores.
- MODCLK Módulo de reloj interno usado para casos especiales o de seguridad.

El usuario puede usar las combinaciones que le interesen de las señales, respetando siempre la señal SMCLK que es con la que se han hecho los cálculos para la periodicidad de las interrupciones de nuestra programación, además del cálculo de velocidad de transmisión.

En futuras ampliaciones se podría estudiar la posibilidad de seleccionar este valor desde la aplicación del PC.

Los registros involucrados en la configuración del reloj son siete, y podemos ver la descripción completa en el documento *datasheet slau272C* suministrado en la web de Texas Instruments *www.ti.com.* Si no se modifican estos registros, obtenemos su valor por defecto, en el que queda configurado con el oscilador interno DCOCLK que suministra esta señal al reloj del sistema MCLK y a los periféricos SMCLK con una frecuencia de 8MHz, que pasa través de un divisor por ocho, con lo que la frecuencia de trabajo es de 1MHz.

5.4 El *timer* y las interrupciones

El temporizador o *timer* es el elemento que nos va a proporcionar una interrupción cada cierto tiempo que aprovecharemos para que el μC ejecute el código que explora los registros y los envía al PC. Sus características son:

- Temporizador/contador de 16 bits
- Entrada de *Clock* seleccionable (ver apartado anterior)
- Siete registros configurables
- Salidas configurables para PWM
- Registro del vector de interrupciones

Dispone de 4 modos de funcionamiento, y el modo elegido en nuestra aplicación es *up mode.* En este modo el *timer* lo que hace es contar los pulsos del reloj (previamente seleccionado) y cuando llega a un *preset* o valor establecido en el registro TAxCCR0, vuelve a empezar desde cero. Y se activa el flag de interrupción que, si están permitidas las interrupciones, será atendido y se ejecutará el código previsto.

En el programa (figura 5-2) se ve reflejada la configuración del *timer*. Por ejemplo, ID_3 significa que los bits del campo ID, bits 6 y 7, toman el valor 3 (los dos bits a uno), y según las definiciones de la figura 5-3, nos indica que utilizamos el divisor de reloj por ocho.

```
// Configurar Timer
TA0CCTL0 = CCIE;                        // Timer A0 interrupción permitida
TA0CTL   = TASSEL_2 + ID_3 + MC_1;      // SMCLK, divisor x8, upmode
TA0CCR0  = 10000;                       // Preset para 12.5 Hz
```

Figura 5-2. ***Configuración Timer A0***

El registro de control para la configuración del temporizador es TA0CTL, y como se puede observar en la figura 5-3, con los bits 8 y 9 seleccionamos qué señal de reloj vamos a utilizar, con los bits 6 y 7 seleccionamos el divisor para adaptar la velocidad de conteo a nuestras necesidades, y con los bits 4 y 5 seleccionamos el modo de funcionamiento (*up mode* en nuestro caso).

Registro TAxCTL

15	14	13	12	11	10	9	8
Reservado						TASSEL	

7	6	5	4	3	2	1	0
ID		MC		Reservado	TACLR	TAIE	TAIFG

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit resets TAxR, the timer clock divider logic, and the count direction. The TACLR bit is automatically reset and is always read as zero.
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

Figura 5-3. ***Registro de control TA0CTL***

5.5 El módulo ***USCI***

Grosso modo, la comunicación serie se basa en descomponer un dato en bits y transferirlos de uno en uno desde el emisor al receptor, donde se vuelve a recomponer para poder operar con él.

En este caso, el microcontrolador ya va equipado con un módulo que va a realizar todo este trabajo. Pero tenemos que configurar algunos parámetros porque hay varias opciones de funcionamiento. En primer lugar, el tipo de comunicación; síncrona (SPI) o asíncrona (UART). En nuestro trabajo se usará la comunicación asíncrona. Después los parámetros, como son la velocidad, la cantidad de bits a enviar en cada paquete de información, y algunas características de la comunicación como el número de bits de stop y la paridad.

No se va a describir con detalle el nivel físico de la comunicación, porque se trata de un canal serie virtual integrado en una comunicación USB, y no ha lugar la exposición de los detalles del estándar EIA-232.

Como dispone de dos módulos, utilizaremos uno de ellos, quedando el otro a disposición del usuario.

La configuración del módulo USCI para comunicación, en el programa, es la que vemos en la figura 5-4.

```
// Configurar USCI en modo UART0 9600 baud, 8 bits, sin paridad, y 1 stop bit

UCA0CTLW0 = UCSWRST;                  // Este bit a 1 para permitir cambios
UCA0CTLW0|= UCSSEL_2;                 // UCSSEL = 2 Selección del clock -> SMCLK
UCA0BRW= 0x0006;                      // 9600 baud según tabla
UCA0MCTLW = 0x2081;                   // Modulación UCBRSx =  según tabla
                                      // UCBRS0 = 0x20, UCBRF0 = 8, UCOS16 = 1
// Bits 15-8   Valor UCBRSx   valor 20h
// Bits 7-4    Valor UCBRFx   valor 8
// Bit 0       Valor UCOS16   valor 1 - Modo Oversampling

UCA0IE |= UCRXIE;                     // Permitir interrupciones de RX
UCA0CTL1 &= ~UCSWRST;                 // Liberación del Reset de la UART
```

Figura 5-4. ***Configuración USCI***

Se puede observar que se activa el bit UCSWRST al inicio y se desactiva al final. Esta es la señal *UART Reset* que debe estar activa para poder realizar modificaciones. En la segunda línea se selecciona el clock al que funcionará el módulo de comunicación. Las dos siguientes son valores que adaptan la velocidad del reloj a la velocidad de transmisión, 9600 baudios, y que se pueden calcular mediante un procedimiento indicado en los *datasheet*, o también tomando los valores de unas tablas suministradas en los mismos. Y por último, al registro UCA0IE se le activa el bit UCRXIE (*interrupt enable*), para que se genere una interrupción cada vez que se reciba algún carácter (comando desde el PC).

5.6 Las entradas y salidas

El microcontrolador dispone de puertos de entradas y salidas que se configurarán en el programa de usuario, y que deberemos leer para poder representarlos en la aplicación del PC.

Cada puerto del µC es configurable, tiene ocho bits, y algunos de estos bits están conectados físicamente a los pines de la tarjeta, y se pueden programar tanto entradas como salidas individualmente por cada bit del puerto. En cada bit del registro PxDIR (x será el número de puerto) asignaremos la función deseada, un '1' si queremos una salida, o un '0' si queremos que sea una entrada. Una vez definidos los bits del puerto, podremos acceder al registro PxIN para leer el estado de las entradas, o colocar valores en el registro PxOUT para activar salidas.

Muchos de los pines del µC están multiplexados con otras funciones de periféricos, como entradas analógicas, entradas de reloj externo, etc. Para seleccionar cual es la función deseada disponemos de dos registros PxSEL0 y PxSEL1.

PxSEL1	PxSEL0	I/O Function
0	0	General purpose I/O is selected
0	1	Primary module function is selected
1	0	Secondary module function is selected
1	1	Tertiary module function is selected

Figura 5-5. **Selección de función de E/S**

Para configurar una entrada analógica en el bit 3 del port 1, por ejemplo, necesitaremos colocar un '1' en el bit 3 del registro P1SEL0 y otro '1' en el bit 3 del registro P1SEL1, ya que la entrada analógica es una función terciaria.

5.7 Las entradas analógicas

Se configuran con los registros SEL0 y SEL1, como hemos visto en el apartado anterior, pero la lectura de sus valores no es tan sencilla como leer el valor en un registro determinado para cada entrada. Solo existe un conversor analógico-digital (ADC) y los canales de entrada están multiplexados y solo dispone de un registro donde almacenar el resultado de las mediciones. En otros modelos hay un registro por cada canal.

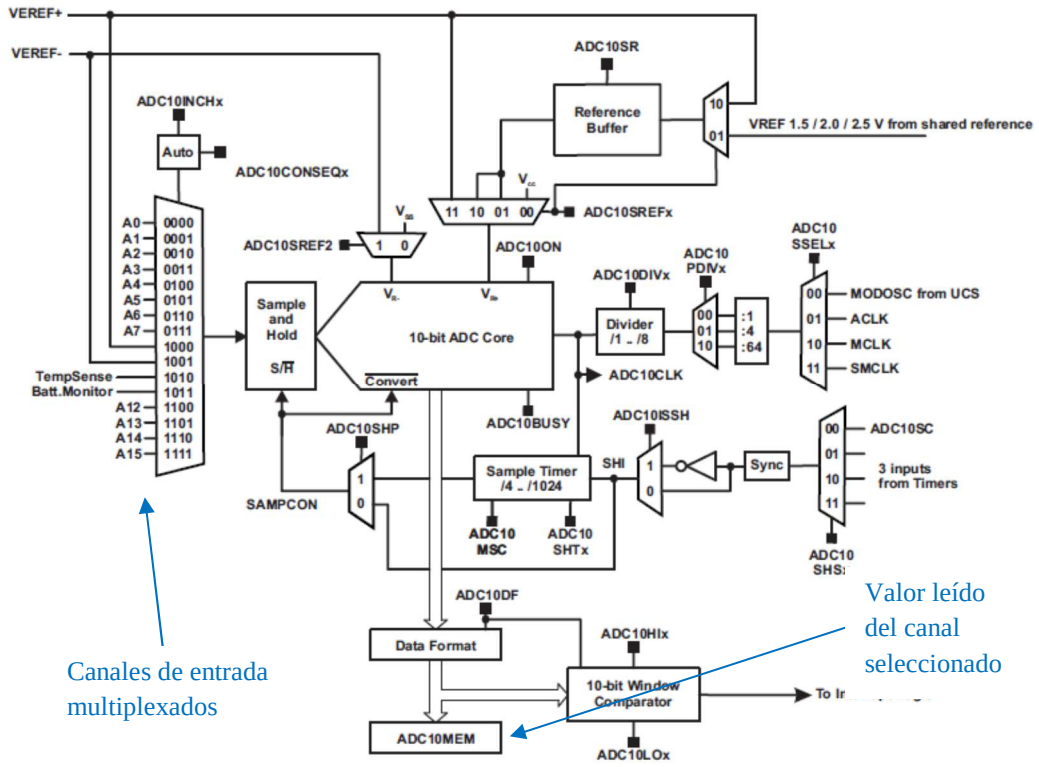


Figura 5-6. ***Diagrama interno del conversor A/D***

Este registro (ADC10MEM) como vemos en la figura 5-6, es actualizado en cada lectura de los distintos canales de entrada. Esto ha supuesto una barrera en nuestra aplicación, ya que no se encuentra la manera de leer este dato en el momento en que se produce, y asociarlo al canal correspondiente. Y tampoco podemos estar leyendo este registro continuamente ya que el bit de interrupción, que avisa al usuario que hay una lectura realizada, se borra al leer este registro y produciríamos una interferencia en el funcionamiento de su programa.

Así que, solo en caso de uso de las entradas analógicas, necesitamos la colaboración del usuario, realizando una segunda llamada a nuestra librería cada vez que quiera actualizar la información sobre un canal analógico. Esta vez la llamada es *monit_analog(canal,dato).* Vemos un ejemplo de llamada en la figura 5-7.

```
canal = ADC10MCTL0 & 0x0F;    //Que canal se está leyendo?
dato  = ADC10MEM0 ;           //Dato leído
ADC10IE = ~ADC10IE0;          //Disable ADC conv complete interrupt
ADC10CTL0 &= ~ADC10ENC & ~ADC10SC;   //Stop conversión
monit_analog (canal, dato);   //Paso de datos a monit
```

Figura 5-7. ***Lectura de entrada analógica***

5.8 Instrumentación del código del usuario

Como ya se dijo en un principio, el primer paso es iniciar nuestra rutina *monit* al inicio del programa *main* de usuario. Y como se ve en la figura 5-8, también hay que incluir el fichero de cabecera *monit.h*, donde se declaran las variables y las rutinas utilizadas.

```
/*  main.c  */
#include "monit.h"
int main(void) {

// Llamada a la rutina inicial del 'Monitor E/S'
monit();

// ******  PROGRAMA DE USUARIO   ***************

// Configurar Ports
P1OUT   &= 0x00;              // Resetear Port1
P2OUT   &= 0x00;              // Resetear Port2
...
```

Llamada a *monit* al inicio de programa

Figura 5-8. ***Llamada a librería***

Y ahora vemos, en la figura 5-9, el procedimiento *monit* completo. En realidad, lo que aquí se puede ver es la anulación del *watchdog*, la inicialización de la UART, y la inicialización del *timer*. El trabajo real se hace en los procedimientos que atienden a las interrupciones.

```
void monit(void){

// CONFIGURACION INICIAL PARA EL FUNCIONAMIENTO COMO MONITOR DE E/S
WDTCTL = WDTPW | WDTHOLD;     // Anular watchdog timer

//Clock                       // Por defecto 8MHz
//CSCTL0 = 0XA500;            // PASSWORD PARA PODER CAMBIAR EL RELOJ  0XA500;

// Configurar UART 0
P2SEL1 |= BIT0 + BIT1;        // Configura UART pins TX and RX
P2SEL0 &= ~(BIT0 + BIT1);

UCA0CTLW0  = UCSWRST;         // Este bit a 1 permite cambios => Reset UART
UCA0CTLW0 |= UCSSEL_2;        // UCSSEL = 2 --> SMCLK  Selección del clock
UCA0BRW    = 0x0006;          // 9600 baud según tabla
UCA0MCTLW  = 0x2081;          // Modulación según tabla
// UCBRS0 = 0x20, UCBRF0 = 8, UCOS16 = 1
//Bits 15-8    Valor UCBRSx   valor 20       ver tabla
//Bits 7-4     Valor UCBRFx   valor 8        ver tabla
//Bit 0        Valor UCOS16   valor 1 - Modo Oversampling

UCA0CTL1 &= ~UCSWRST;         // Liberación del Reset de la UART
UCA0IE |= UCRXIE;             // Permitir interrupciones de RX

// Configurar Timer
TA0CCTL0       = CCIE;        // Timer A0 interrupción permitida
TA0CTL         = TASSEL_2 + MC_1 + ID_3;     // SMCLK, upmode, /8
TA0CCR0        = 10000;       // Preset para aprox. 80ms = 12.5 Hz

startData=0;                  // De momento no se envían datos
}
```

Figura 5-9. ***Procedimiento de inicialización monit***

El temporizador *Watchdog* (WDT) es un temporizador de 16 bits. La principal función de éste módulo WDT es llevar a cabo el reinicio controlado del sistema después de que se haya

producido un problema en el software, por ejemplo, un bloqueo o bucle infinito. Si el intervalo de tiempo seleccionado expira, se genera un reinicio del sistema.

Cuando se recibe un carácter del PC por el puerto serie, se activa la interrupción de RX de la UART. En la rutina de la interrupción se detecta el carácter recibido y en función de este se llamará a la subrutina adecuada para comprobar la comunicación, comenzar o terminar el envío de datos al PC.

Periódicamente el flujo de programa pasará por la interrupción del timer, y allí hará la llamada al procedimiento *status()*, que es realmente donde se consulta el estado de cada puerto, se convierte el dato a código ASCII y se escribe en el buffer de transmisión de la UART o módulo de comunicación.

El tratamiento de la interrupción es tan sencillo como vemos en la figura 5-10. Se consulta la variable *startData* que nos informa si hay que enviar los datos al PC, y si es así, se llama a *status()*.

```
// **********  Timer A0 interrupt ******************
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
        PJOUT ^= BIT0;          // Toggle PJ.0 (LED)
        if (startData==1) status();
}
```

Figura 5-10. ***Interrupción del timer***

Y en la llamada a *status()* se convierte el dato a formato hexadecimal (ASCII) y se envía al PC por el canal serie.

Para convertir un número a formato hexadecimal, se ha encontrado una manera sencilla y rápida que lo que hace es asignar todos los valores posibles de un dígito hexadecimal a un array de caracteres, `BIN2HEX[]`. Al acceder a la variable colocando como índice el dato a convertir, el resultado es el carácter de la conversión realizada. En la figura 5-11 vemos un ejemplo.

```
void convertASCII(void){
char BIN2HEX[]="0123456789ABCDEF";

byteLow  = BIN2HEX[ dato & 0x0F];
byteHigh = BIN2HEX[(dato & 0xF0)>>4];
```

Figura 5-11. ***Ejemplo conversión byte a ASCII***

Si hay que enviar los datos al ordenador, el primer carácter que se envía es ‘#’ indicando al PC que viene a continuación un paquete de datos.

Todas las funciones anteriores están en la librería *monit.c*, que junto con *monit.h* es todo lo que hay que añadir al programa de usuario.

Con esto damos por finalizada la aplicación en el microcontrolador. Se ha probado el funcionamiento tanto de la comunicación como del manejo de los datos con una aplicación gratuita que emula un terminal. Se puede ver a continuación.

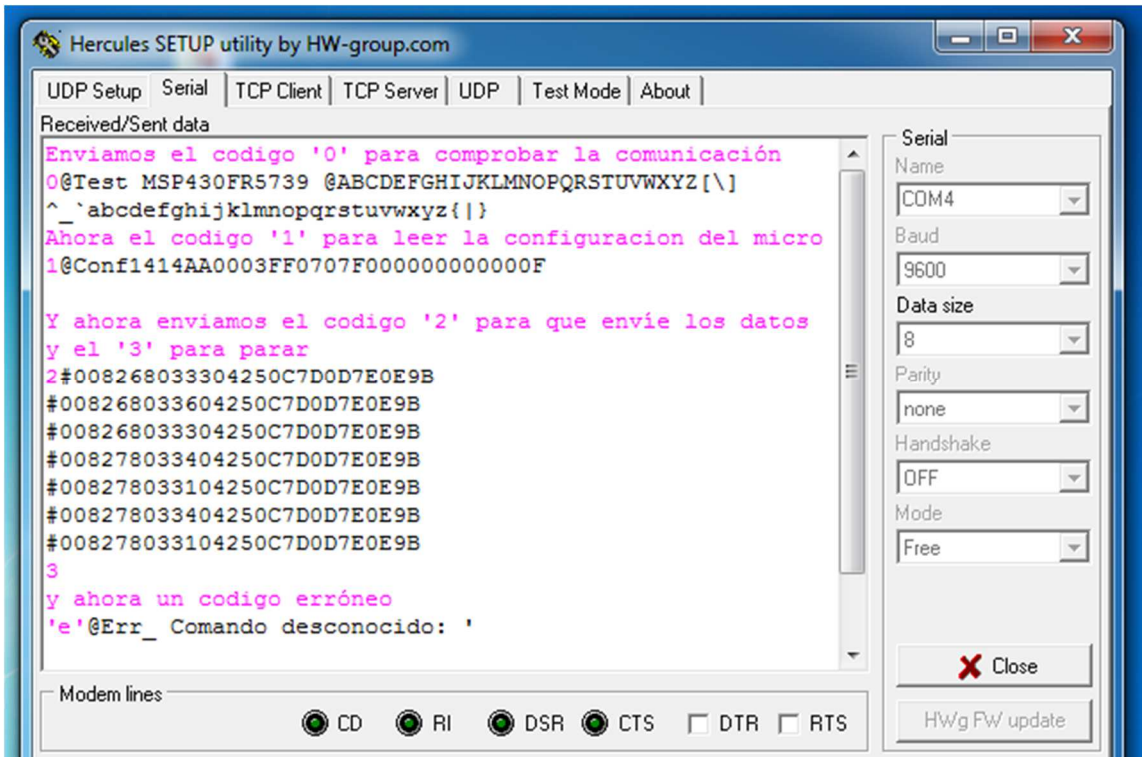


Figura 5-12. ***Pruebas de funcionamiento***

6. Aplicación en el PC

6.1 Visión general

Vamos a dividir el desarrollo de la aplicación en tres grandes bloques:

- Inicialización de objetos y gráfico inicial
- Comunicaciones, configuración y funcionamiento
- Tratamiento de datos y gráficos dinámicos

La aplicación en el ordenador se ha desarrollado partiendo de un formulario inicial sobre el que se han ido incrustando todos los elementos gráficos. Al tener una fase de inicialización y otra de funcionamiento continuo, se ha optado por obtener dos pantallas distintas utilizando la estructura de pestañas (*tabs*).

6.2 Posibilidades y problemas, *brainstorm*

La aplicación debe tener la posibilidad de configurar y chequear las comunicaciones; selección de puertos, baudios, avisos de *online*, *offline*, etc.

Se deben representar todos los elementos que lleva incorporados la tarjeta, como son las conexiones al exterior (tiras de pines en los lados), pulsadores y los diodos *led*.

Cada elemento debe identificarse con un nombre y un índice para poder tener un acceso rápido y eficaz, y además poder tratar un conjunto de elementos usando bucles que simplifiquen la programación.

La distribución de los pines de los 2 conectores que lleva la tarjeta a ambos lados de la misma no está ordenada y esto dificulta la asignación de los nombres, la creación de etiquetas etc.

Los datos que envía el µC van a llegar de manera continua. Hay que capturarlos y tratarlos inmediatamente, sin dejar de actualizar en pantalla los datos ya recibidos.

Una entrada o salida digital (bit) se representa con un objeto (rectángulo) de color, pero para la representación de una entrada analógica (valor) hay que buscar una representación visual clara, además de la numérica.

Cada uno de los puntos anteriores se ha planteado antes de empezar a programar, intentando así localizar posibles puntos conflictivos que puedan retrasar o incluso paralizar el desarrollo del proyecto. Han sido pequeños obstáculos que han ofrecido la posibilidad de ingeniar distintas soluciones que se han ido adaptando a la programación de la aplicación.

Haremos una breve descripción de algunas de las soluciones adoptadas mientras se va viendo el funcionamiento de la aplicación.

6.3 Gráfico inicial

Si vamos a representar el funcionamiento de un sistema con una serie de componentes y características propias y queremos visualizar el funcionamiento del mismo de una manera similar a la real, vamos a presentarlo como una imagen animada del dispositivo, y que además nos dé la información que necesitamos de forma clara.

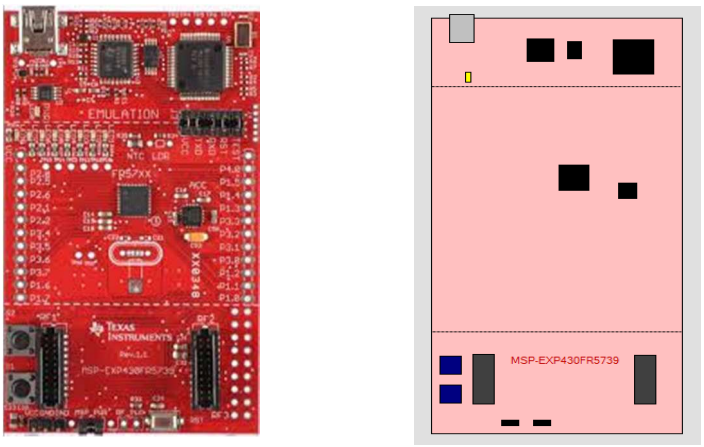


Figura 6-1. ***Gráfico inicial***

Sobre este gráfico inicial (figura 6-1) se van añadiendo todos los elementos dinámicos que se construyen en tiempo de ejecución.

Por cada elemento que representamos necesitamos un objeto, y este debe tener relación con los bits de los puertos a los que se está accediendo, es decir, hay que establecer un vínculo directo entre el objeto y el bit al que representa. Esto se consigue con las matrices o arrays de objetos. El array de objetos es de dos dimensiones, para poder situarlo según el *puerto* y el bit al que pertenece.

Por ejemplo, si se activa el bit 2 del puerto 3 y queremos acceder al objeto del array *R* que lo representa para colorearlo o iluminarlo, nos dirigiremos a *R[port][bit]*, en este caso sería *R[3][2].Backcolor = Color.Red* .

Además de lo indicado en los dos párrafos anteriores, el número y tipo de objetos puede cambiar en cada ejecución, ya que el usuario puede configurar los puertos de forma distinta. La mejor solución encontrada para este problema ha sido crear en tiempo de ejecución y al inicio de la aplicación, un array de objetos vacío, al que llamaremos *port[][]*. Y ahora creamos una instancia del objeto *led*, que será un rectángulo de color azul. Una vez creada esta instancia es tan sencillo como igualar la referencia del objeto *led* a la del *port[3][2]*. Podemos volver a inicializar la instancia *led* tantas veces como necesitemos e ir asignándola a los elementos del array.

Vemos un ejemplo claro en la figura 6-2. Supongamos que necesitamos crear 8 *leds* y asignarlos al puerto 3, en los bits 0 al 7. Basta con crear las 8 instancias del objeto *led* e ir asignándolas a cada nuevo elemento de la matriz.

```
        // ******  Gráficos  **********

private RectangleShape[][] port;
private RectangleShape led;
...
...
// Posición inicial del primer led
posX = 125; posY = 85;

// Bucle para generar 8 leds
For (n=0;n<8;n++)
{
        // Creamos un rectángulo base para los leds
        led = new RectangleShape(posX, posY, 8, 6);
        led.BackColor = Color.Blue;
        // Y una vez creado asignamos esa referencia al port.
        port[3][n] = led;
        // Ya está. Tenemos un led llamado port[3][n].
        // Incrementamos posY y generamos otro led port[3][n+1]
        posY += 30;
}
```

Figura 6-2. ***Asignación de objetos a una matriz***

La distribución de los pines de conexión de cada una de las E/S en la tarjeta tampoco está ordenada. Hay 12 pines de conexión a cada lado de la tarjeta, y cada uno está conectado a un bit de uno de los puertos del microcontrolador. El primer pin *Vcc* es la tensión de alimentación, no está conectado a ningún bit de E/S.

Figura 6-3. ***Puntos de conexión de la tarjeta***

Para tener enlazado cada pin con su nombre y localización, se ha creado una tabla índice con tres arrays que se pueden recorrer en orden y nos suministran tanto el puerto como el bit al que pertenece cada pin.

```
string[]     labelTextIzq = {"Vcc","P2 .0","P2 .5","P2 .6","P2 .1","P2 .2","P3.4"};
private int[] asigPortIzq = { 0, 2, 2, 2, 2, 2, 3 };    // Asignaciones ports y bits
private int[] asigBitIzq  = { 0, 0, 5, 6, 1, 2, 4 };    // al grafico de la tarjeta
```

Figura 6-4. ***Asignación puntos de conexión***

Y así, en un bucle con *i=3* tendremos que el *pin 3* se llamará *P2.6*, pertenece al *port 2* y es el *bit 6*. Este sistema se ha utilizado para asignar las conexiones de la parte izquierda y también de la derecha.

6.4 Comunicaciones y configuración

El primer paso en la aplicación es ajustar los parámetros de comunicación. Como ya se indicó en el apartado 3.1.2, al utilizar un puerto de comunicaciones (*COMxx)* virtual, en cada ordenador el puerto puede ser distinto. Con este sencillo procedimiento (figura 6-5) averiguamos los puertos activos y se añaden a un *ComboBox (cBoxPort)* que nos permitirá elegir el puerto deseado.

```
// Revisa los puertos (Com) activos
Private void miraComms ()
{
String[] comPorts = SerialPort.GetPortNames();
cBoxPort.Items.Clear();
cBoxPort.Items.AddRange(comPorts);
}
```

Figura 6-5. ***Detección puertos activos***

Una vez seleccionado el puerto podemos activar el pulsador *test* para realizar una prueba de comunicación. A partir de aquí, activamos los eventos del puerto serie. Esto significa que en cuanto se reciba cualquier dato se ejecutará un procedimiento donde se tomarán decisiones sobre la información recibida. El manejo de eventos en la recepción del canal serie se activa según podemos ver en la figura 6-6.

```
miPort.DataReceived += new SerialDataReceivedEventHandler(miPort_Recepcion);
 …
 …
private void miPort_Recepcion(object sender, SerialDataReceivedEventArgs e)
{
 try
 {
  if (strBufferIn.Substring(0, 1) == "#"){ this.Invoke(new EventHandler(RXDatos)); }
  if (strBufferIn.Substring(0, 1) == "@"){ this.Invoke(new EventHandler(RXComandos));}
  miPort.DiscardInBuffer();
 } catch (TimeoutException exTimeout)
    {
     onLine(false); MessageBox.Show("Pérdida comunicación. Revisa conexiones..." );
    }
}
```

Figura 6-6. ***Eventos en la recepción***

Y aquí se ve también la manera en que se distingue si la recepción es de un comando o de datos (“@” o “#”), tratándose en procedimientos distintos.

Si el test de comunicación es correcto, pasamos a la siguiente fase en que le pedimos al µC que nos transmita la configuración que tiene programada de sus E/S. El formato en que se visualiza lo vemos en la figura 6-7.

Se aprecia una barra de progreso que indica la realización de la lectura de la información, y además cada bit de cada puerto con una indicación. Estas indicaciones nos dicen si el bit está configurado como entrada (‘I’), como salida (‘O’) o si está configurado como entrada analógica (‘A’).

Figura 6-7. ***Lectura de la configuración***

Y una vez leída la configuración, ya disponemos de toda la información necesaria para dibujar la tarjeta con todos los detalles, colores, y elementos que van a intervenir en su funcionamiento.

```
// Dibuja en pantalla la configuración actual del microcontrolador
private void pintaConf()
{
 for (int p = 1; p < 6; p++)
 {
  for (int b = 0; b < 8; b++)
  {
   int col = cnfPort[p, b]; // col es el color del pin según dato en cnfPort[p,b]
   if (labelPort[p][b] != null)           // si tiene etiqueta
   labelPort[p][b].BackColor = colores[col]; //se colorea con el color correspondiente
  }
 }
}
```

Figura 6-8. ***Color de etiquetas según el tipo de E/S***

El procedimiento *pintaConf()* de la figura 6-8 recorre los cinco puertos y los ocho bits de cada puerto para colorear la etiqueta de cada bit según esté configurado como entrada, salida o entrada analógica.

Podemos seleccionar la pestaña *Board* y aparece la tarjeta completamente dibujada, con los ocho *led* en la parte superior, las doce señales en la parte izquierda, y las otras doce de la parte derecha.

Todas las señales, representadas por rectángulos de color blanco, van acompañadas de una etiqueta en la que se lee el nombre del puerto y bit al que pertenece. Y además, cada etiqueta tiene un color determinado, que indica el tipo de señal que es; entrada, salida o entrada analógica.

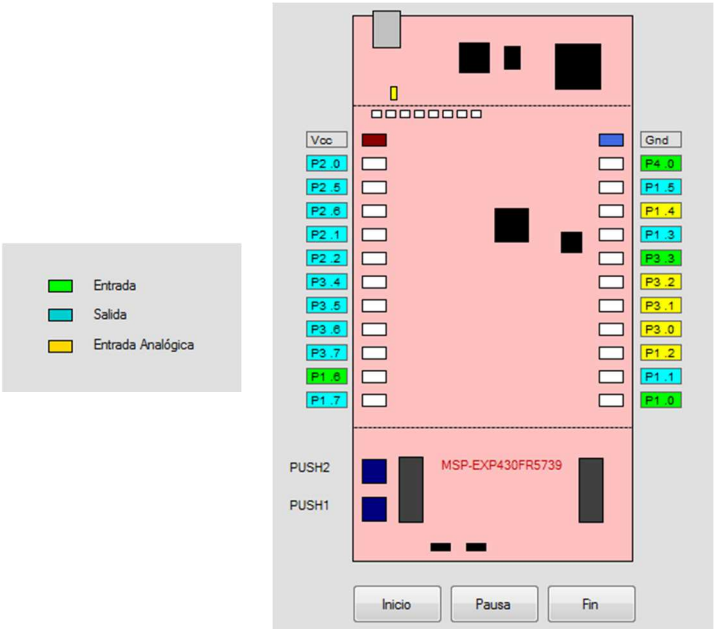


Figura 6-9. ***Presentación de la tarjeta ya configurada***

Los botones debajo de la tarjeta son para iniciar o parar la transmisión de datos y visualizar o detener la visualización de los estados de las señales en la tarjeta.

Pero también hay que representar las entradas analógicas. Se han encontrado en internet algunas librerías para este tipo de indicadores, pero hemos preferido utilizar algo más estándar. Los elementos *ProgressBar* se utilizan a menudo para este tipo de indicaciones; *vu-meter*, llenado de depósitos, indicadores de aceleración, etc.

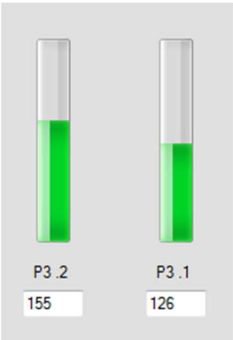


Figura 6-10. ***Representación de las entradas analógicas***

La información queda muy clara; número de puerto y bit, valor gráfico y valor numérico de cada entrada, según vemos en la figura 6-10.

6.5 Gráficos en movimiento

Para poner el sistema a trabajar solo necesitamos pulsar el botón *Inicio*. Entonces el PC envía la orden *Start* (ASCII '2') y el µC comienza a transmitir paquetes de datos con la información del estado actual de cada una de sus entradas, salidas y entradas analógicas.

En cuanto llega un paquete de datos al PC se llama al procedimiento *checkFrame()*, que comprueba si el frame recibido tiene la longitud que indica su último byte. Si no es correcta, el paquete se descarta y se incrementa un contador de fallos de comunicación, quedando registrado el evento. Inmediatamente, en unos pocos milisegundos se recibirá el siguiente mensaje de datos que volverá a ser chequeado, y el sistema seguirá funcionando sin interrupciones.

Se ha descartado el uso de un control de flujo como *stop & wait* por varios motivos; para no generar más tráfico, no sobrecargar el microcontrolador y también porque la posibilidad de llenar el buffer es prácticamente nula ya que los mensajes de datos son muy pequeños en comparación con el tamaño del buffer de recepción del sistema (4096 bytes). Y en el supuesto caso de *buffer overflow* saltará una excepción que detendrá la aplicación.

Si el paquete de datos es correcto, se descomponen en bytes los cinco primeros valores, que corresponden con los cinco puertos de E/S digitales, y cada número de puerto y el valor del mismo se pasan como parámetros al procedimiento *pinta* que de una manera sencilla localiza el valor de cada bit y colorea el objeto de la matriz *port[p][b]* en rojo o blanco si el valor del bit es '1' o '0' respectivamente.

```
private void pinta(int p, int value)
{   // Colorea cada pin (E/S) según el valor actual
 try
 {
  for (int b = 0; b < 8; b++)          //Bucle de los 8 bits
  {
    var bit = (value & (1 << b)) != 0; //Extrae el bit b
    if (bit)                           //Si está a '1'
    {
      port[p][b].BackColor = Color.Red;//pinta rojo
    }
    Else                               //y si está a '0'
    {
      port[p][b].BackColor = Color.WhiteSmoke; //pinta blanco
    }
  }
 }catch (Exception ex) { }
}
```

Figura 6-11. ***Procedimiento para colorear cada bit de un puerto***

En cuanto a las entradas analógicas, el procedimiento es parecido al anterior. Se le pasa el parámetro *bar* (número de barra) y el valor a representar. Este valor puede ser modificado por dos factores (*offset* y *ganancia*), y el valor resultante se escribe en el cuadro de texto además de representarse gráficamente en el *ProgressBar*.

```
private void pinta_ana(int bar, int valor)
{
valor = (valor + offsetBar[bar]) * factorBar[bar];
if (valor>255) valor=255;            //Límite máximo
if (valor < 1) valor = 0;            //Límite mínimo
txtVertProgBar[bar].Text = ""+valor; //Valor numérico
vertProgBar[bar].Value = valor;      //Valor gráfico
}
```

Figura 6-12. ***Procedimiento para representar valores analógicos***

Para modificar la representación del valor de cada canal analógico simplemente hay que posicionar el cursor encima de una *ProgressBar* y pulsando el botón izquierdo del ratón, aparecen en pantalla 2 elementos de la clase *NumericUpDown* en los que podemos seleccionar el valor deseado tanto de *offset* como de ganancia. De esta manera se puede adaptar la dinámica de funcionamiento del indicador analógico a los criterios o valores deseados.

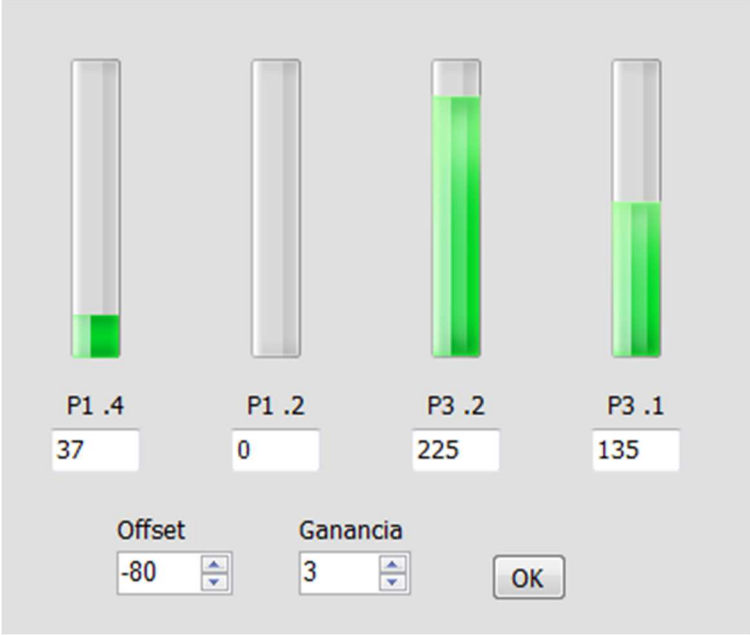


Figura 6-13. ***Detalle del ajuste de offset y ganancia***

La modificación de estos valores es solo a nivel de presentación en pantalla. No altera los valores leídos, si se quiere guardarlos en fichero como veremos en el siguiente apartado.

6.6 Registro de datos

Se ha programado la opción de grabar secuencialmente en un archivo los datos recibidos en el PC. Con esto disponemos de un fichero con la información necesaria para representar todo lo ocurrido en el sistema en un tiempo determinado, pero con la movilidad que permite el almacenamiento en archivos. En el fichero quedarán registrados todos los cambios ocurridos en formato ASCII. Y así se puede, por ejemplo, analizar detenidamente para localizar cambios o comprobar sincronizaciones entre señales.

El fichero se genera siempre que tengamos activa la marca del *CheckBox* que hay debajo de los pulsadores y en el momento en que se pulse *inicio*. Y se cierra con los datos registrados cuando se pulse *pausa*.

Como se ha dicho, el formato en que se guardan los datos es ASCII, tal y como los envía el microcontrolador, siendo el primer carácter el símbolo '#' seguido de los valores de cada uno de los 5 puertos digitales y a continuación el número de canal y el valor de cada uno de los canales analógicos usados en la aplicación.

En la figura 6-14 vemos un ejemplo de captura de datos en el fichero, abierto con cualquier editor de texto. En la fila 8 se puede ver la detección (provocada) de un fallo de trama.

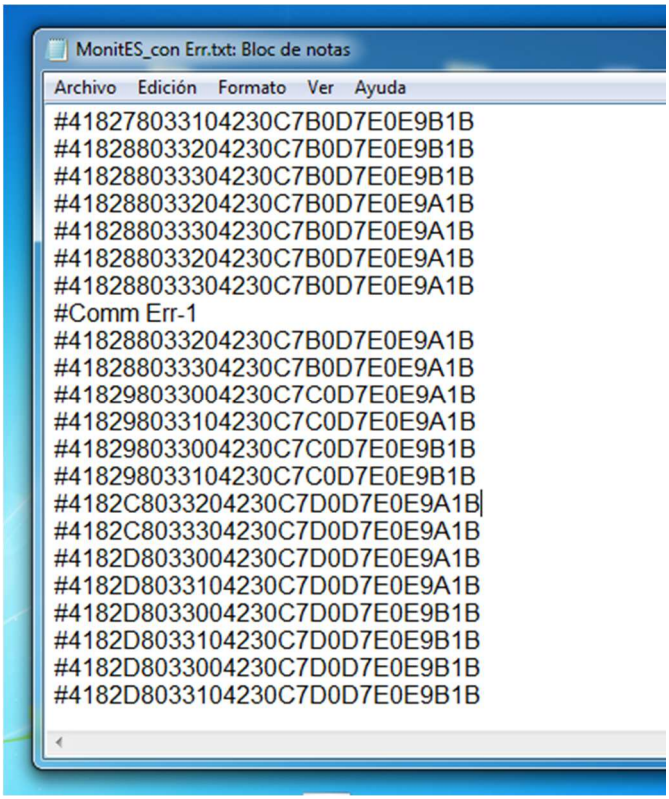


Figura 6-14. ***Contenido del fichero de datos***

Ahora podemos ver el diagrama de flujo completo, donde se interrelacionan las dos aplicaciones y donde se aprecia el funcionamiento global del sistema.

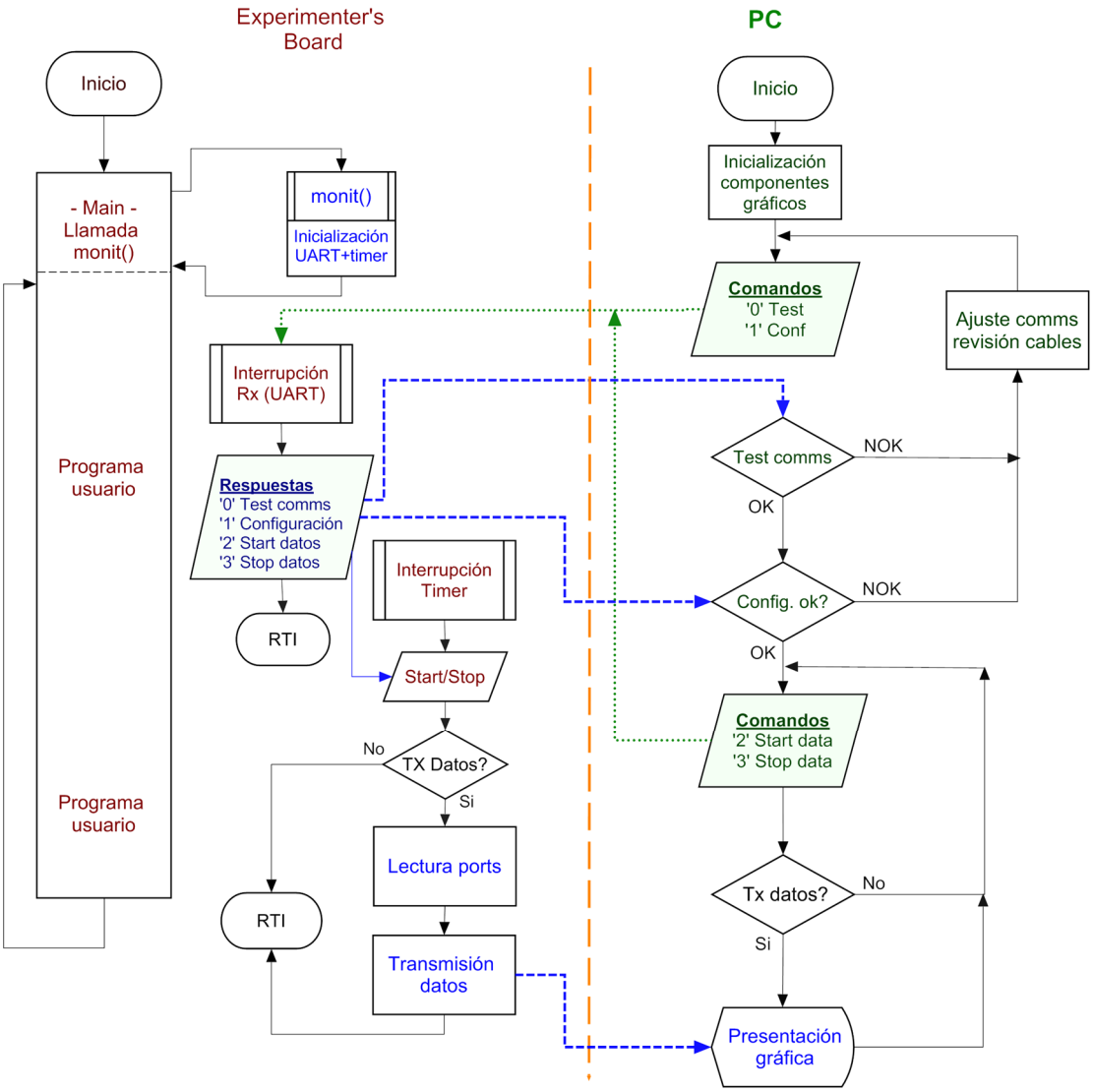


Figura 6-15. ***Diagrama de flujo global***

7. Resultados

Finalmente, el resultado de la aplicación gráfica en funcionamiento junto con la aplicación del microcontrolador la podemos ver en la figura 7-1.

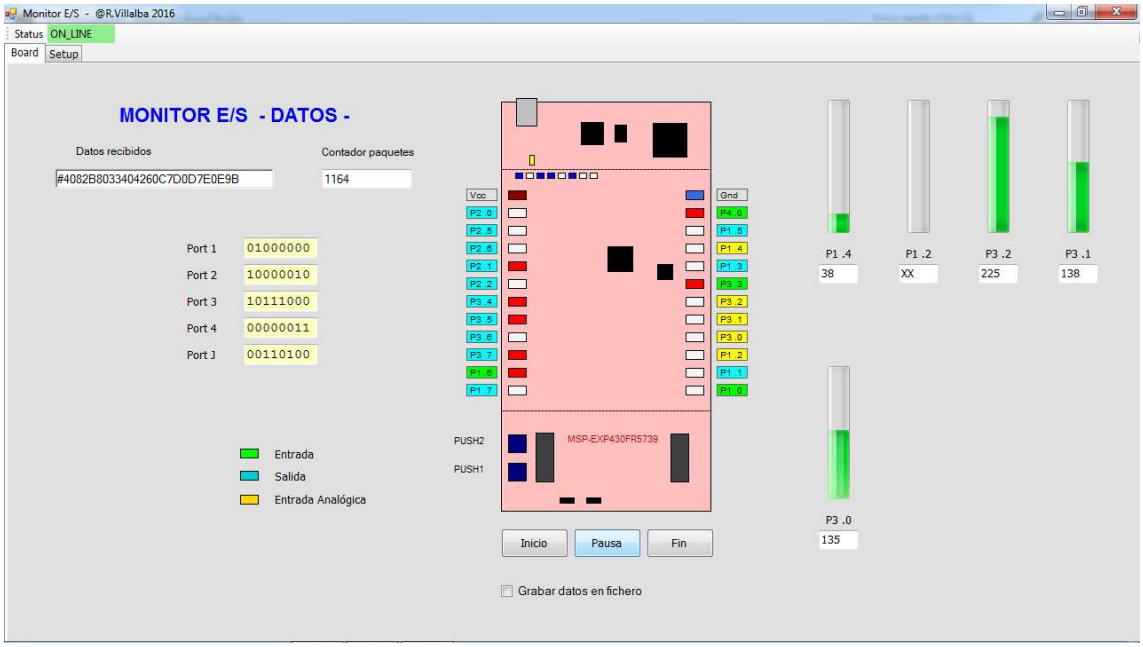


Figura 7-1. ***Aplicación gráfica en funcionamiento***

Se ha comprobado que el funcionamiento es correcto y fluido, tal y como se pensó en su diseño.

Quedan representadas todas las señales correspondientes a las entradas y salidas digitales, así como las entradas analógicas, y también los dos pulsadores incluidos en la tarjeta (en color azul). La presentación es bastante clara y sencilla.

8. Conclusiones

Es una satisfacción para el autor del trabajo haber llegado al final del proyecto alcanzando los objetivos planteados. También es muy gratificante la sensación de haber superado varios de los obstáculos que han aparecido en cuanto a programación se refiere, y que al final el sistema funcione como estaba previsto.

Es importante recalcar que además de una buena formación se necesita obtener la documentación adecuada en cada uno de los proyectos que se quieran emprender, pues de ello y de la experimentación va a depender el nivel de éxito en los resultados.

Se han estudiado en profundidad las características del microcontrolador para optimizar el funcionamiento e interferir mínimamente en el programa de usuario. En realidad, este no es un proyecto de uso general, pero se puede adaptar a distintos tipos de microcontroladores y a las distintas necesidades del usuario.

8.1 Ampliaciones y nuevas ideas

El proyecto ha servido para aprender y aumentar la experiencia en sistemas empotrados, e incluso para imaginar nuevas opciones y ampliaciones que pueden mejorar la funcionalidad y la utilidad del trabajo.

Se puede incluir, por ejemplo, una herramienta en la aplicación, que abra el fichero de datos grabado y lo represente en pantalla, tipo osciloscopio, para visualizar las señales registradas.

El microcontrolador también dispone de un reloj de tiempo real (*RTC*) que se puede utilizar para aumentar la capacidad del sistema, obtener registros, programar alarmas, etc.

El valor de la señal SMCLK que es con la que se han hecho los cálculos para la periodicidad de las interrupciones y el cálculo de velocidad de transmisión, se podría seleccionar desde la aplicación del PC.

Son muchas las opciones y posibilidades que ofrece la programación de estos sistemas. Con ideas, imaginación, y disfrutando del trabajo (como ha sido en esta ocasión) se pueden ofrecer soluciones a casi cualquier necesidad planteada.

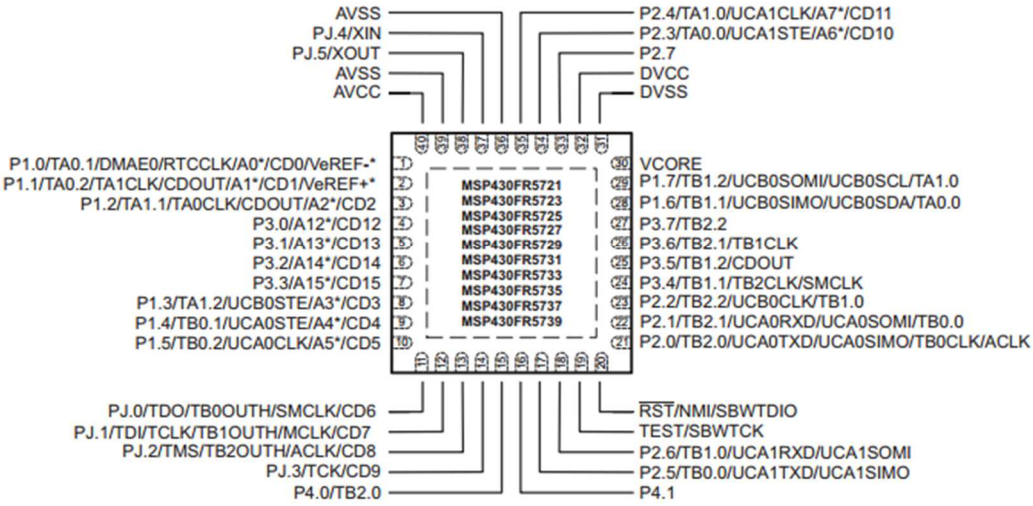
9. Anexo A

Características del microcontrolador **MSP430FR5739**

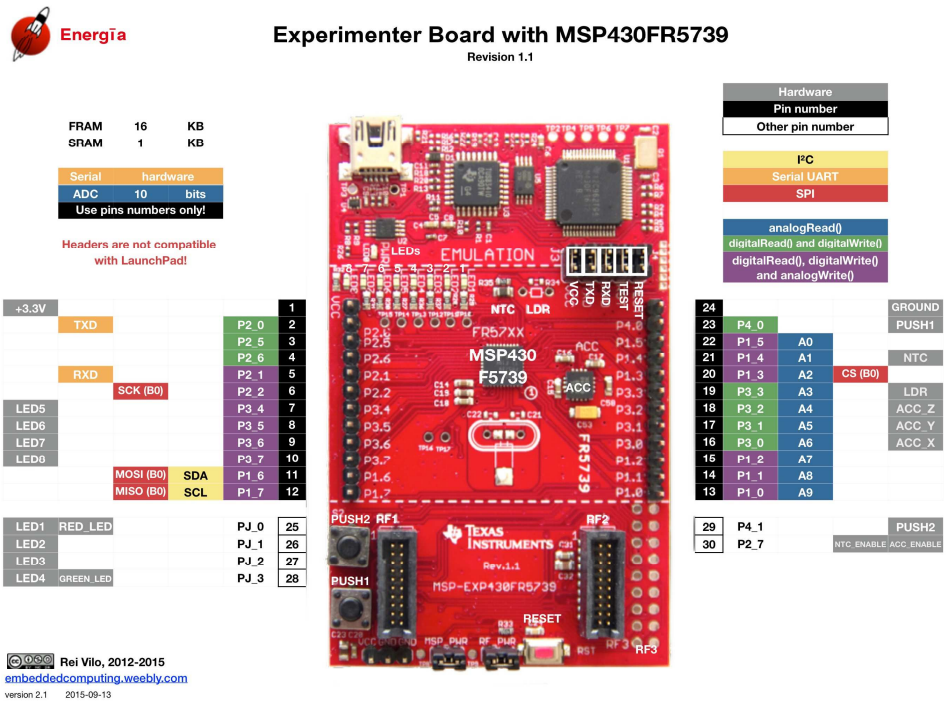
CPU	MSP430
Frequency (MHz)	24
Non-volatile Memory (KB)	16
RAM (KB)	17
GPIO	32
I2C	1
SPI	3
UART	2
DMA	3
ADC	ADC10 - 12ch
Comparators (Inputs)	16
Timers - 16-bit	5
Timers - 32-bit	0
Multiplier	32x32
AES	N/A
BSL	UART
Min VCC	2
Max VCC	3.6
Active Power (uA/MHz)	91.667
Standby Power (LPM3-uA)	6.4
Wakeup Time (us)	78
Additional Features	Real-Time Clock Watchdog Temp Sensor Brown Out Reset IrDA
Special I/O	N/A
Operating Temperature Range (C)	-40 to 85
Package Group	TSSOP VQFN
Estimated Package Size (WxL) (mm^2)	40VQFN: 6 x 6: 36 mm2 38TSSOP: 6.2 x 12.5: 103 mm2

10. Anexo B

Pinout del chip microcontrolador con encapsulado RHA



Pinout de la tarjeta *Experimenter's Board*



11. Bibliografía

Texas Instruments, «MSP430FR57xx Family User's Guide». 2013.

Allen Jones, Matthew Mc Donald, y Rakesh Rajan, *Visual C# 2010 Recipes*. United States: Apress, 2006.

Christian Nagel, Bill Evjen, Jay Glynn, Karli Watson, y Morgan Skinner, *Professional C# 2005 with .NET 3.0*. Indianapolis, Indiana: Wiley Publishing, Inc, 2004.

Herbert Schildt, *C# Manual de referencia*. McGraw-Hill Interamericana, 2003.

Joe Albahari y Ben Albahari, *C# 3.0 In a nutshell*. USA: O'Reilly Media, 2007.

Kirk Zurell, *C Programming for Embedded Systems*. Lawrence, Kansas: R&D Books, 2000.

Michael Barr y Anthony Massa, *Programming embedded systems with C and GNU development tools*. USA: O'Reilly Media, 2006.

Michael J. Pont, *Patterns for time-triggered embedded systems*. Great Britain: Addison-Wesley, 2001.

Michael J. Pont, *Embedded C*. Addison-Wesley, 2002.

Texas Instruments, «16-bit 32-bit MCU Low-power MCUs». [En línea]. Disponible en: http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/overview.page. [Accedido: 20-may-2016].

Texas Instruments, «MSP430FR5739 Tools & Software». [En línea]. Disponible en: http://www.ti.com/product/msp430fr5739/toolssoftware?keyMatch=msp430fr5739&tisearch=Search-EN-Everything. [Accedido: 07-may-2016].

Texas Instruments Incorporated, «MSP Debuggers». [En línea]. Disponible en: http://www.ti.com/lit/ug/slau647c/slau647c.pdf. [Accedido: 08-may-2016].

«Ventajas C#». [En línea]. Disponible en: http://urriellu.net/es/articles-software/csharp-advantages.html. [Accedido: 08-may-2016].