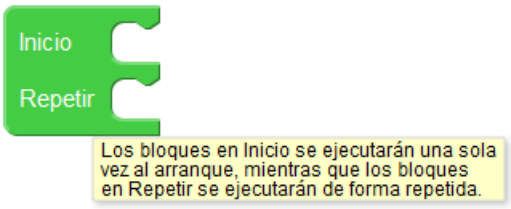
Visualino

Programación visual para Arduino



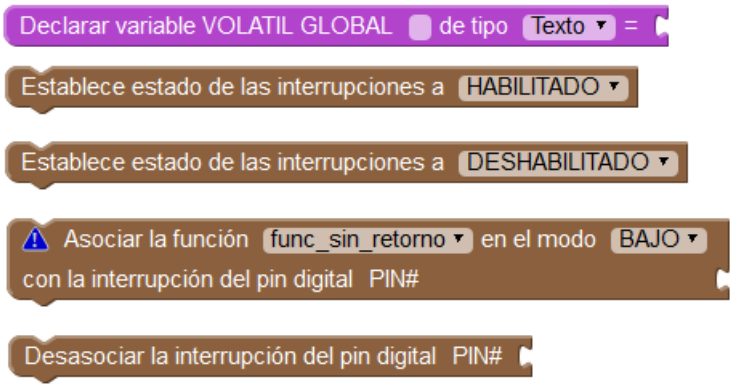
Uso de Interrupciones

Por Francisco José Molina Robles
fmrobles@latinmail.com

Prefacio

Como programador dedicado al mundo de la enseñanza, he de reconocer que entornos de programación como Visualino ofrecen a los docentes la posibilidad de enseñar programación algebraica de una forma mucho más sencilla, intuitiva y agradable para los alumnos. En mis años de experiencia en la docencia, he de reconocer que uno de los mayores desafíos a los que me he enfrentado es enseñar programación a mis estudiantes. Siempre he considerado la programación como una tarea que requiere, además de unos conocimientos algebraicos y matemáticos, una capacidad de abstracción y creatividad que para muchos resulta difícil de alcanzar. Hoy en día, gracias a la introducción del concepto de programación por bloques de Google Blockly, MIT AppInventor, Scratch y otros muchos, es posible enseñar las técnicas de programación algebraica a alumnos a edades tempranas.

Tengo que reconocer que Visualino es una herramienta que me ha resultado muy útil en mi trabajo diario en las aulas. Gracias a que se distribuye libremente y también a que se pueden realizar cambios sobre ella, me he tomado la libertad de incluir algunos bloques que considero son importantes para aumentar la potencia de este entorno de programación. Esta es mi pequeña contribución a Visualino, en agradecimiento a los desarrolladores de esta plataforma libre.



Almansa, diciembre de 2016

Introducción

Si alguna vez tiene ocasión de conversar con alguna mujer, es posible que en algún momento ella le cuente el viejo chiste que dice que los hombres son *monotarea*, es decir, que sólo son capaces de hacer una cosa cada vez. Y si esa mujer se dedica a programar sistemas informáticos o microcontroladores, también es posible que le cuente ese chiste haciendo la comparación entre Arduino y los hombres. Porque, en realidad, hablando ahora un poco más en serio, hay que decir que Arduino es un sistema muy sencillo si lo comparamos con los microprocesadores modernos, ya que solamente es capaz de realizar una sola tarea en un momento determinado. Esto supone una ventaja cuando comenzamos nuestro aprendizaje en esta plataforma, porque la programación es muy sencilla. Sin embargo, conforme avanzamos en el desarrollo de aplicaciones más complejas, esta característica puede llegar a convertirse en un inconveniente, sobre todo si queremos que nuestra placa de Arduino sea capaz de controlar varias cosas a la vez.

El sistema nervioso de la mayoría de los seres vivos, incluyendo a los humanos, es capaz de realizar muchas tareas al mismo tiempo. Sin embargo, casi todas estas tareas las realizamos de una forma inconsciente o automática, como los pulmones en la respiración o los latidos del corazón para bombear la sangre a través de nuestro cuerpo. Sin embargo, cuando debemos realizar varias tareas de forma consciente, no tenemos la capacidad de realizarlas todas a la vez, sino que debemos centrarnos en una sola y, sólo cuando la dejamos o cuando la terminamos, podemos pasar a la siguiente. Por ejemplo, si estamos en casa leyendo, debemos prestar toda nuestra atención a esa tarea y es posible que no escuchemos las conversaciones que se producen a nuestro alrededor. Si en ese momento suena nuestro teléfono, tendremos que interrumpir momentáneamente nuestra lectura para atender a nuestro interlocutor y, una vez terminada la conversación, podemos continuar con nuestra tarea por donde lo habíamos dejado.

Un microcontrolador como Arduino funciona de una forma parecida a cómo gestionamos nosotros nuestras tareas conscientes. Si queremos que sea capaz de realizar varias tareas al mismo tiempo, cosa que hablando estrictamente no puede hacer, podemos adoptar varias estrategias. Por un lado, podemos programarlo para que vaya realizando tareas conforme le van llegando o decidiendo cuáles son las más prioritarias. Y por otro, también podemos establecer mecanismos para que Arduino pueda realizar una parte de cada tarea cada vez. En este manual me he centrado en explicar el mecanismo de control de interrupciones en Arduino, por el que este microcontrolador es capaz de atender tareas más prioritarias cuando se presentan.

Qué son las interrupciones

Al igual que otros sistemas informáticos como los ordenadores, las placas de Arduino disponen de un sistema de gestión de **interrupciones**. Las interrupciones son útiles para hacer que determinadas acciones se realicen de forma automática en un programa y pueden ayudar a resolver problemas de sincronización. Algunas tareas bien gestionadas con interrupciones incluyen aquellas para las que no se sabe cuándo se van a producir. Los microcontroladores de Arduino ya incluyen programas con interrupciones cargados en el *firmware* que realizan algunas funciones, como las comunicaciones por los puertos serie, la carga de los programas o las funciones de control de tiempo. Por defecto, las interrupciones están activadas, aunque se pueden desactivar, pero sólo es recomendable cuando se quieren ejecutar determinadas secciones del código del programa que son *críticas*.

Las interrupciones son señales que recibe el microcontrolador que le advierten que se ha producido un suceso que es necesario tratar con una cierta urgencia. Por ejemplo, en la introducción ya se ha hablado de la situación que se nos plantea a nosotros diariamente cuando estamos realizando cualquier tarea y de repente suena nuestro teléfono; en ese caso, es necesario atender esa tarea (a no ser que estemos realizando una tarea todavía más importante) o de lo contrario esa llamada se perderá. Otra característica que define a las interrupciones es que éstas se pueden producir en cualquier momento y no existe ningún patrón que defina con qué periodicidad se van a producir.

Si es necesario asegurar que un programa siempre recoja los pulsos de un codificador rotatorio, de forma que nunca se pierda ninguno, sería muy difícil escribir un programa que además hiciera cualquier

otra tarea, porque el programa necesitaría consultar constantemente la conexión con el codificador, para poder recoger los pulsos cuando suceden (que puede ser en cualquier momento). Otros sensores tienen una interfaz dinámica similar, sobre todo aquéllos en los que no se sabe de antemano cuándo se va a producir un determinado evento que se desea recoger (por ejemplo, capturar un sonido cuando se produce, detectar la caída de una moneda con un sensor de infrarrojos, etc.). En todas estas situaciones, el uso de las interrupciones puede liberar al microcontrolador de tener que leer constantemente un determinado pin para capturar un cambio de estado, de forma que éste puede realizar otras tareas y a la vez no hay riesgo de perder esos eventos.

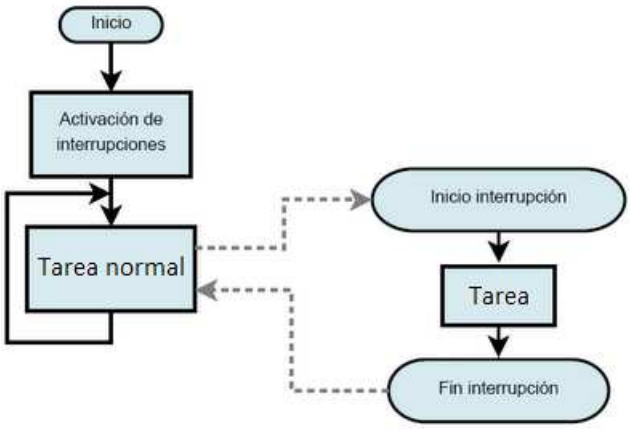
Cómo se gestionan las interrupciones en Arduino

El mecanismo de control de interrupciones en Arduino es bastante complejo de llevar a cabo, ya que es capaz de controlar diferentes niveles de prioridades y permite dejar a medio tratar algunas interrupciones cuando llegan otras de mayor prioridad.

Para simplificar el mecanismo de control de interrupciones en Arduino, vamos a trabajar a un nivel más alto, en el que damos por válidas algunas condiciones. La primera y más importante es que supondremos que Arduino sólo puede ejecutar una tarea de gestión de interrupción cada vez. Esto supone que:

- Si se producen varias interrupciones al mismo tiempo, sólo se ejecutará el código asociado a una de ellas y el resto se ignorarán.
- Si se produce una interrupción cuando se está tratando otra, también se ignorará.

A este nivel, podemos decir que la gestión de interrupciones en Arduino es bastante sencilla. Para ello, es necesario que el en código del programa se defina una función encargada de tratar la interrupción (también llamada **rutina de servicio de interrupción** o **ISR** por sus siglas en inglés). A continuación, es necesario asociar la señal de la interrupción deseada con la función (rutina) que se ha creado para manejarla. De esta forma, cuando el microcontrolador esté realizando su tarea normal, en el momento en el que se produzca la interrupción, dejará de ejecutar esas instrucciones y comenzará a ejecutar las instrucciones de la función asociada con la interrupción. Finalmente, cuando termine de ejecutar el código de esa función, el microcontrolador continuará con la ejecución de su tarea normal, justo por la instrucción donde la dejó. Gracias a este mecanismo, podemos aprovechar para que el microcontrolador realice otras tareas mientras no se produzcan interrupciones y, cuando éstas se produzcan, se pueden tratar inmediatamente sin que exista pérdida de información.



Qué interrupciones se pueden programar en Arduino

Las interrupciones que se pueden programar y gestionar a nivel básico en Arduino son aquellas que están relacionadas con un cambio de valor en un pin digital. Por ejemplo, podemos conectar un pulsador en un pin digital y asociarlo con una interrupción cuyo código se ejecutará cuando pulsemos ese botón. En la tabla siguiente se enumeran las interrupciones y pines asociados que se pueden utilizar en diferentes placas de Arduino.

Interrupciones disponibles y puertos asociados en diferentes placas de Arduino						
Placa	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	
Due	Todos los pines					

Limitaciones y restricciones

Hay que tener en cuenta que, cuando se ejecuta el código de una rutina de servicio de interrupción, algunas operaciones del microcontrolador no funcionan como se espera y existen algunas restricciones a la hora de escribir el código:

- La función (rutina) que maneja la interrupción no puede aceptar ni devolver ningún valor (debe definirse una función **sin retorno** y **sin parámetros**).
- Generalmente, la función (rutina) asociada debe ser lo más corta posible y no realizar tareas que no son necesarias para la gestión de la interrupción.
- El siguiente bloque no funciona mientras se está tratando una interrupción, ya que su funcionamiento interno se basa también en el uso de interrupciones. Esto significa que este bloque no se puede poner dentro de la función que trata una interrupción.

- Los datos recibidos por el puerto serie pueden perderse durante la ejecución de la función (rutina) asociada, ya que este puerto también hace uso de las interrupciones para enviar y recibir datos. Esto significa que no se deben usar los **bloques de comunicación** dentro de las funciones que tratan las interrupciones.

Manejo de interrupciones en Visualino

Otra cuestión importante a la hora de definir el código del programa que maneja una interrupción es que, si se declaran variables globales cuyos valores van a ser modificados por la función (rutina) que maneja la interrupción, éstas deben definirse como **volátiles** usando este bloque:

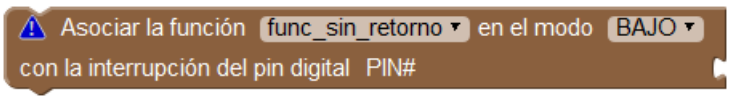
Los bloques utilizados para el manejo de interrupciones en Arduino con Visualino son los siguientes:

- 

desactiva todas las interrupciones, de forma que el microcontrolador no realizará ninguna tarea relacionada con ellas. Esto quiere decir que las operaciones que realiza el microcontrolador gracias a las interrupciones, no funcionarán (recibir datos por el puerto serie, comunicaciones, bloque Esperar, interrupciones definidas por el programador, etc.). Sólo se recomienda desactivar las interrupciones cuando queremos que el microcontrolador se dedique a ejecutar de forma exclusiva fragmentos *críticos* de código. En Arduino, las interrupciones están habilitadas por defecto, a no ser que se use este bloque.

- 

activa el uso de las interrupciones, que se supone que previamente han sido desactivadas con el bloque anterior.

- 

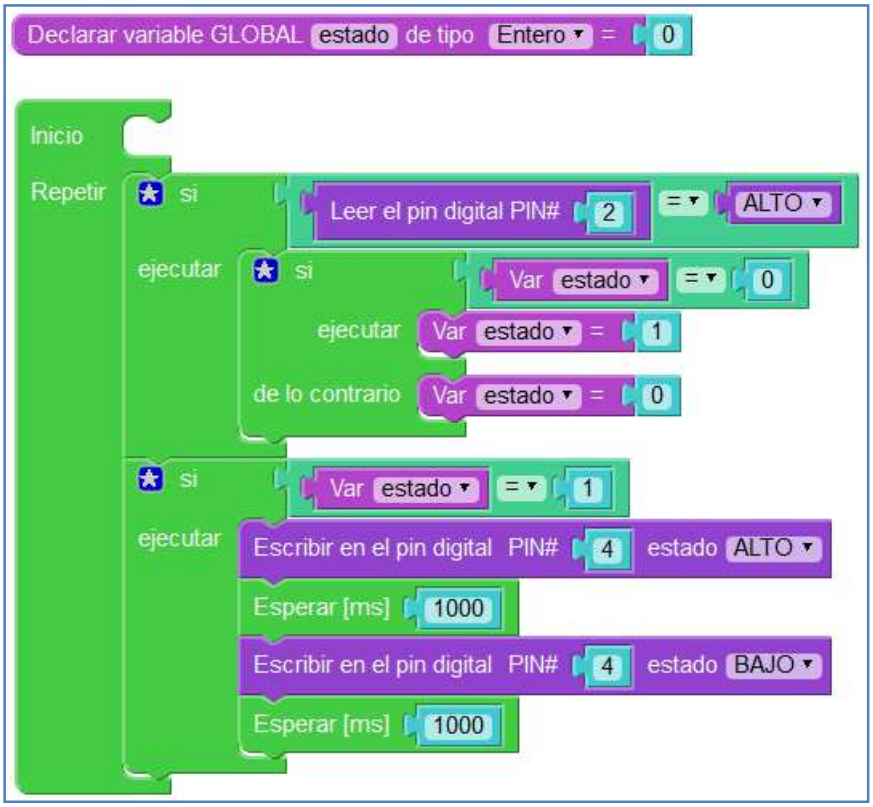
especifica el nombre de la función o rutina de servicio de interrupción que se debe ejecutar cuando se produce un cambio en un pin determinado. En la lista desplegable *modo* se indica cuándo se va a activar la interrupción (*BAJO* o *LOW* cuando el pin está a nivel bajo, *CAMBIO* o *CHANGE* cuando el pin cambia de valor, *SUBIENDO* o *RISING* cuando el pin pasa de nivel bajo a nivel alto y *BAJANDO* o *FALLING* cuando el pin pasa de nivel alto a bajo).

- 

desactiva la interrupción del pin relacionado y desasocia la función (rutina) de manejo.

Ejemplo de uso de interrupciones en Visualino

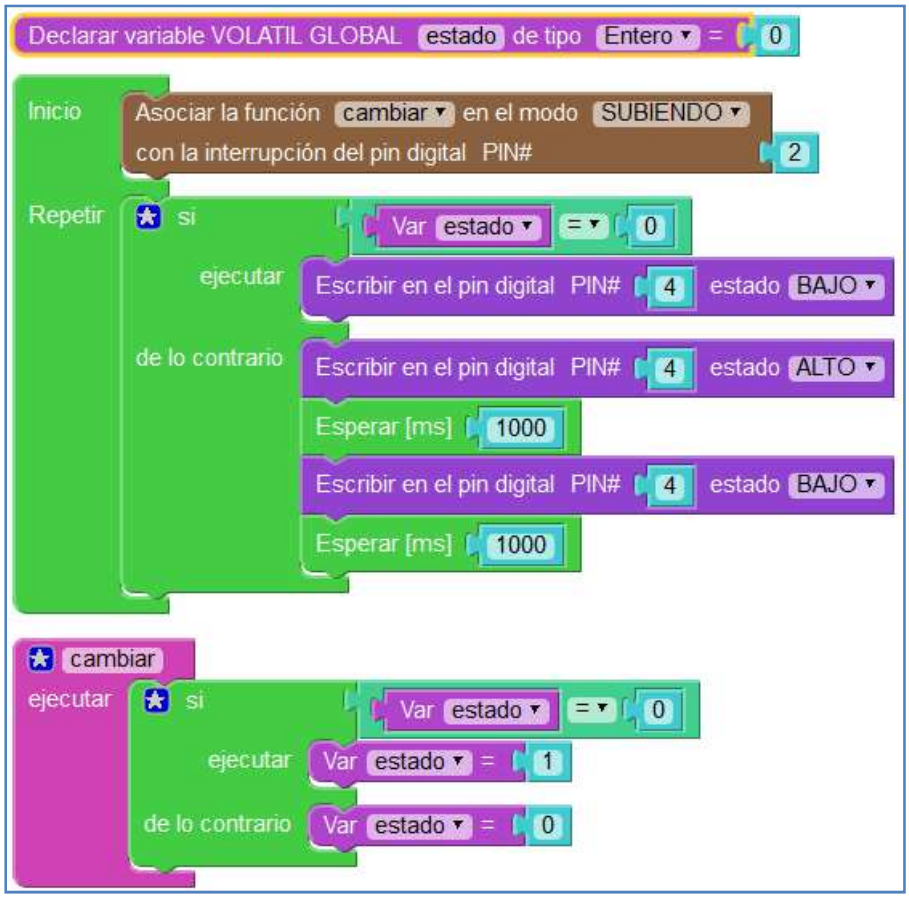
Suponemos que tenemos el siguiente programa de Arduino, en el que hay conectado un pulsador en el pin 2 y un led en el pin 4. Este programa enciende y apaga el led cuando se pulsa el interruptor del pin 4. Cuando el led se pone en marcha, se encenderá y apagará alternativamente cada segundo.



Si se prueba este programa, se comprobará que en muchas ocasiones, cuando se activa el pulsador, Arduino no detecta la pulsación y no se cambia el estado del led. Esto es debido a que es posible que se active el pulsador en el momento en el que Arduino está ejecutando uno de estos bloques:

En ese caso, Arduino está ocupado ejecutando el bloque de espera y por lo tanto no puede comprobar si el pin 2 está a nivel alto porque se ha activado el pulsador. Por lo tanto, la pulsación que nosotros realicemos se perderá y no tendrá ningún efecto sobre el programa.

Este problema que hemos encontrado en el programa lo podemos solucionar si reescribimos el código usando interrupciones. Como la pulsación se puede producir en cualquier momento, la interrupción hará que Arduino deje momentáneamente la tarea que está realizando (incluso si está esperando) para cambiar el valor de la variable estado. En este segundo programa, no se pierde ninguna pulsación.



Referencias

- https://www.arduino.cc/en/Reference/AttachInterrupt (Referencia oficial de Arduino sobre interrupciones)
- http://gammon.com.au/interrupts (Programación avanzada de interrupciones con Arduino, por Nick Gammon)
- http://www.visualino.net/index.es.html (Pagina oficial de Visualino)
- http://procomun.educalab.es/es/ode/view/1453974406581 (Apuntes y proyectos de Visualino, por Aurelio Gallardo Rodríguez)