Antes de abordar lo que es una interrupción como tal, expondremos la motivación que llevo a implementarlas dentro de los microporcesadores, para poder utilizarlos para afrontar principalmente problemas que requieren de la interacción con el mundo real.

Imagine que hoy llegará a su casa un envío importante, sin embargo el timbre de la puerta está averiado y usted no puede avisarle al repartidor sobre éste inconveniente, la primera opción que puede considerar consiste en esperar cerca de la puerta hasta que el envío llegue, pero ésta no es una muy buena idea pues quizá tenga muchas cosas que hacer, o simplemente no le parezca que estar sentado esperando sea una actividad muy enriquecedora, por lo cual usted opta por continuar con sus actividades regulares pero ir a revisar la puerta constantemente para ver si el repartir ha llegado con su envío; no obstante, ésta idea tampoco resulta ser muy prometedora puesto que el repartidor no puede quedarse esperando en la puerta hasta que usted aparezca y la probabilidad de que coincidan es ciertamente baja, además, a la larga perdería el tiempo que le tomaría ir a revisar la puerta en las veces que no encontró al repartidor.

Este problema, que dentro del ámbito de los microcontroladores consiste básicamente en ser capaz de detectar y responder a una señal que puede ocurrir en cualquier momento, y que no da tiempo a ser aplazada, se presenta en múltiples escenarios, ya sea dentro de un microcontrolador para un sistema embebido o incluso dentro de las actividades que se realizan comúnmente en un PC, por lo cual es un problema que resultar ser recurrente y de importante solución.

Dada la necesidad de poder atender estas señales en cualquier instante, una alternativa, justo como en la analogía del principio, pudo haber sido destinar al microcontrolador a estar al pendiente exclusivamente de susodicha señal, lo cual evidentemente es un desperdicio total de la capacidad de procesamiento de este, que además imposibilita el reconocimiento de varias señales de este tipo; una segunda aproximación podría ser programar al microcontrolador para que revisara periódicamente todos los periféricos verificando si alguno de ellos estaba emitido alguna señal, este método es conocido como *polling*, el cual fue el primero en ser implementado en respuesta a este problema, sin embargo, no era una solución muy acertada pues consumía recursos y tiempo al microcontrolador revisar todos los periféricos, que a la larga lo hacían un proceso muy ineficiente, además de que señales que sólo se activasen durante periodos de tiempo inferiores al de revisión podrían llegar a ser omitidas, lo cual es incluso peor que la primera alternativa, por lo tanto se continuo con la búsqueda de una solución alterna, la cual concluyó con lo que se conoce dentro del ámbito de los microcontroladores como una interrupción.

Una interrupción no es más que un mecanismo cuyo propósito es poder hacer que el microcontrolador detenga lo que está procesando, pase a ejecutar unas instrucciones previamente establecidas, las cuales se conocen técnicamente como una ISR (*Interrupt Service Routines*), en el momento en que una señal prioritaria es recibida desde el exterior denominada IRQ (*Interrupt Request*), como un periférico (el teclado o el mouse por ejemplo), o desde el interior, como en el caso del desbordamiento de una variable, una división por 0 o una violación de segmento [Manual], con el fin de atender esta señal y luego poder retomar el proceso anterior en las mismas condiciones en que las que fue interrumpido.

Para poder controlar las interrupciones, los microcontroladores poseen una unidad de interrupciones que se encargar del flujo de este proceso. Como se mencionó antes, las fuentes de interrupción pueden ser externas o internas, pero todas estas son manejadas por lo que se conoce como la tabla del vector de interrupciones, cuya ubicación en la memoria depende del modelo del microcontrolador, pero generalmente se encuentra en la posición 0x0000 [UNO], aunque otros autores afirman que en términos prácticos, la tabla en realidad comienza en 0x004h, pues no consideran la primera de las interrupciones de la tabla que es la de RESET, ya que esta siempre está habilitada y predefinida en todos los microcontroladores [Fonseca].

Esta tabla está ordenada según la prioridad de las interrupciones, es decir, cuál interrupción debe ser atendida primero en caso de que varias de estas se presenten simultáneamente; en el microcontrolador Atmega328P, la primera interrupción es la de RESET, la cual es seguida por INT0, luego por INT1, etc. (para más información sobre el ordenamiento de la tabla de interrupciones del Atmega328P consultar [UNO]), las interrupciones INT0 e INT1 son utilizada para censar señales del exterior, ya que están asociadas a los pines 2 y 3, respectivamente, del microcontrolador, permitiendo conectarlos con otros circuitos externos para censar señales provenientes del mundo exterior.

En cada entrada de la tabla se encuentra la dirección de memoria donde están almacenadas las instrucciones de la ISR de la interrupción correspondiente, por ejemplo, al ejecutar la instrucción: **attachInterrupt**(0, lectura, CHANGE) (sobre la forma en que se implementan las interrupciones hablaremos en detalle más adelante), le estamos indicando al microcontrolador que cuando se detecte un cambio (por eso la palabra clave *CHANGE*) en el pin 2 que está asociado a la interrupción INT0 que corresponde al 0 del primer argumento de la instrucción, se salte a la región de la memoria donde está almacenada la función **lectura**(), y la ejecute en respuesta a la interrupción; en este caso la interrupción es la INT0, la señal fue el cambio en el pin 2 provocado desde el mundo exterior, y la ISR fue la función **lectura**().

Para controlar cuáles interrupciones son atendidas por el microcontrolador, se utiliza un sistema para poder habilitarlas; éste sistema cuenta con un bit denominado GIE (*Global Interrupt Enable*), el cual, si está desactivado (en un valor de 0), deshabilitará todas las interrupciones a excepción de la de RESET; si el GIE está activado, igualmente cada interrupción posee su propio bit de permiso que debe estar activado para que dicha interrupción pueda ser atendida, de esta forma el programador puede seleccionar cuáles interrupciones desea habilitar y cuáles no [Ilustraciones].

Con todo esto en mente, expondremos ahora como es el flujo del proceso para atender una interrupción externa:

1. En el mundo exterior se genera el evento de nuestro interés que, por medio de nuestro circuito exterior, se ve reflejado en una señal (como un flanco de subida, un flanco de bajada, un cambio en algún voltaje, etc.) la cual es detectada por la unidad de interrupciones del microcontrolador, que activa una bandera para indicar que una interrupción ha sido detectada; gracias a esto, si la señal del exterior deja de ser detectada antes de que el microcontrolador atienda la interrupción, como la bandera se había dejado activada, el microcontrolador la tenderá de todas formas, evitando así la omisión de esta [Fonseca].
2. Al final de cada instrucción, el mircocontrolador revisa si hay alguna solicitud de interrupción pendiente (IRQ) [Intel] y, en caso afirmativo, en lugar de ejecutar la instrucción siguiente del proceso que está llevando a cabo, el microcontrolador guarda el estado de registros y banderas actual, además de la dirección de memoria del contador del programa (abreviado PC, *Program* *Counter* o también abreviado como IP, *Instruction* *Pointer*) para saltar a la tabla del vector de interrupciones y revisar cuál fue la fuente de la interrupción. Algo que es importante resaltar es que para que la interrupción sea detectada el bit GIE debe estar activado y la interrupción en cuestión debe estar habilitada.
3. Al encontrar la fuente de la interrupción dentro de la tabla del vector de interrupciones, se desactiva el bit GIE para no recibir alguna otra interrupción, y se salta a la dirección de memoria asignada para la ISR de la interrupción recibida.
4. Se ejecutan las instrucciones de la ISR y posteriormente a la instrucción de retorno, se activa de nuevo el bit GIE, además de restaura los registros, banderas y el contador del programa guardados, para continuar con la instrucción siguiente a la que termino de ejecutar cuando detectó la interrupción [UNO].

Notemos que en el ítem 3 se menciona que justo antes de comenzar a ejecutar la ISR se desactiva el bit GIE para no recibir otra interrupción durante el proceso, en caso de que se requiera procesar otra posible interrupción dentro de la ISR, es necesario habilitar explícitamente el bit GIE por medio de las instrucciones que esta contiene [Intel].

Como se mencionó antes, cuando se hablaba sobre la tabla del vector de interrupciones, en caso de que dos o más interrupciones estén activadas cuando el microcontrolador las está revisando, se atenderá la interrupción que primero aparezca en la tabla, de donde cobra sentido lo que se mencionó antes de que las interrupciones de la tabla están organizadas por su prioridad.

Otro detalle relevante es que hay modelos de microcontroladores que poseen prioridades respecto a las interrupciones clasificándolas en una prioridad alta o baja, la ejecución de una ISR correspondiente a una interrupción de prioridad baja, puede ser detenida para atenderá una interrupción de alta prioridad, pero no para atender una de baja; no obstante, la ejecución de una ISR de una interrupción de prioridad alta, puede ser interrumpida por la interrupción RESET, pero sólo por esta interrupción en particular. Si una interrupción no es atendida dentro de otra, simplemente lo será justo después de que se finalice la que está en curso, postergando la continuidad del proceso previo a las interrupciones para el momento en que todas han sido atendidas.

Finalmente, expondremos la forma de implementar una interrupción para censar una variable externa dentro de la plataforma Arduino, más específicamente en un Arduino UNO, debido a que en esta plataforma es que se implementará el ejemplo práctico. La gran mayoría de la siguiente información fue tomada de [UNO].

Las ISR son funciones cuya implementación debe permitir que se ejecuten lo más rápido posible, pues recordemos que estas interrumpen el procesamiento del microcontrolador, generalmente son funciones que no utilizan parámetros de entrada o salida (salvo en casos excepcionales), además de que son orientadas a la escritura o lectura de datos y cálculos numéricos.

Para definir la ISR de una de las interrupciones INT0 o INT1, que en la plataforma Arduino UNO son las únicas que permiten una interrupción externa, se utiliza la sentencia:

**attachInterrupt**(num\_interrupt, ISR, modo)

**NOTA:** Es mejor utilizar digitalPinToInterrupt() para poder usar el mismo código en otro modelo de Arduino.

Donde *num\_interrupt* corresponde a cuál interrupción de la tabla del vector de interrupciones activará la ISR, en la plataforma Arduino UNO únicamente los pines 2 y 3 pueden censar este tipo de interrupciones (para más información sobre otros modelos de Arduino consultar [LINK DE ARDUINO]), su correspondencia en la tabla del vector de interrupciones es 0 y 1 respectivamente, por lo cual para censar una señal en el pin 2 se coloca 0 y para el pin 3 se colca 1; *ISR* es el identificador de la función que implementaremos para que se ejecute en respuesta a la IRQ, es decir, se colocará en la posición *num\_interrupt* de la tabla la dirección de memoria de la función *ISR*, y finalmente *modo* corresponde a la forma en que la interrupción será disparada, la cual puede ser cualquiera de los siguientes:

**LOW**: Se activa la interrupción cuando se encuentra un valor de 0 V en el pin designado.

**CHANGE**: Se activa la interrupción cuando se detecta un cambio en el voltaje o valor lógico del pin.

**RISING**: Cuando hay un flanco de subida, es decir, se pasa de un estado de LOW a HIGH.

**FALLING**: Cuando hay un flanco de bajada, es decir, se pasa de un estado de HIGH a LOW.

Adicionalmente, si se desea desasociar de la interrupción para censar una variable exterior su ISR, se puede utilizar la sentencia:

**detachInterrupt**(num\_interrupt)

Donde *num\_interrupt* corresponde al índice de la interrupción deseada dentro de la tabla del vector de interrupciones.

Para deshabilitar todas las interrupciones a excepción de la de RESET, se puede utilizar la sentencia:

**noInterrupts**()

Para activarlas se puede utilizar:

**interrupts**()

Cuando se está implementando una ISR hay que tener en cuenta varias consideraciones:

1. Si la ISR fue adjuntada por medio de **attachInterrupt**(), la función **delay**() no funciona.
2. No utilizar la función **Serial.print**(), recordemos que la ISR debe ejecutarse lo más rápido posible.
3. Las variables externas a la ISR deben ser declaradas utilizando el modificador **volatile**, esto le indicará la compilador que estas variables pueden modificar su valor en cualquier momento de la ejecución

Ahora veamos un ejemplo demostrativo de una interrupción para… en la plataforma de Arduino UNO.

**OJO:** Cuando se entra en una ISR no se aumente el tiempo registrado por millis() ni micros(), delay() no funciona porque se basa en millis()