

**Diego Armando Becerra Iñiguez**

**Interrupciones**

**Tarea 2**

**Timers e interrupciones**

**Un procesador embebido normalmente tiene que:**

* Medir tiempos
* Generar eventos basados en tiempo
* Responder en tiempo real a eventos que ocurren en tiempos impredecibles

Un sistema embebido tiene, generalmente, que atender varias “tareas”, algunas de ellas periódicas, otras disparadas por eventos.

Las interrupciones y los timers son herramientas clave para lograr un manejo efectivo del tiempo y los eventos en un sistema embebido.

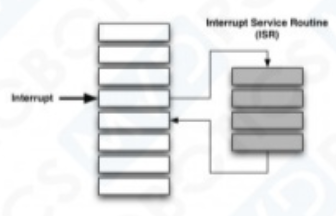
En la programación estándar, en una PC, las interrupciones son un recurso del hardware que se usa a nivel del S.O., el kernel, los drivers, lo que llamamos “bajo nivel”.

En un µC la programación es siempre de “bajo nivel”. Programamos directamente sobre el CPU, y no sobre la “máquina virtual” del S.O.

 Asimismo, en la programación normal de la PC, bajo el S. O., existen los threads. El equivalente en el uC son las interrupciones. Aparecen algunos problemas parecidos, de sincronización, y se resuelven de manera similar.

Las interrupciones son un recurso esencial de los sistemas embebidos. Básicamente, la interrupción es un mecanismo mediante el cual el CPU puede, ante cierto evento, suspender lo que está haciendo en ese momento y pasar a atender una rutina de alta prioridad. Una vez finalizada ésta, el CPU vuelve a su actividad anterior.

* La interrupción es disparada por un evento externo al CPU. Puede ser el cambio de estado de un pin (interrupción externa) o cierta señal de un dispositivo interno del uC, por ejemplo, el desborde de un timer, el estado del ADC, etc.
* En algunos procesadores también existen las interrupciones generadas por software (lo cual tiene sentido en el contexto de un 5.0.). En el AVR no existen, pero se pueden simular cambiando manualmente el estado de un pin



* El timbre de una casa, o el "ring" del teléfono, son analogías perfectas "SF" delo que es y cómo funciona una interrupción.
* Cuando el CPU recibe Ia señal de interrupción, abandona (interrumpe) inmediatamente lo que está haciendo y salta a una rutina especial. Dicha rutina suele llamarse interrupt Hand/ er o interrupt Service Routine (ISR). lnhrrupt
* Una vez que Ia ISR termina, el CPU retoma su actividad anterior desde el punto en que Ia dejó.
* Antes de atender la interrupción, el CPU guarda el estado de sus registros (incluyendo el Program Counter), para poder continuar luego con Ia ejecución de las instrucciones del programa principal. Ese” estado” del CPU se llama el contexto, y la operación de salvarlo y restituirlo se llama cambio de contexto.
* Las interrupciones son señales aleatorias, externas al CPU, que pueden ocurrir en cualquier momento, fuera del flujo predecible de un programa. Es un proceso asíncrono; funciona como si fuera otro hilo de ejecución (thread).
* El microcontrolador dispone de mecanismos para activar, desactivar, priorizar, inhibir, etc., distintas fuentes de interrupciones, y para asociar cada una de ellas con rutinas ISR programadas por el usuario.
* Las interrupciones son lo que permite evitar el polling.
* El polling consiste en consultar permanentemente la ocurrencia de cierto evento desde el programa.
* Al contrario que la interrupción, el polling es sincrónico y determinista.

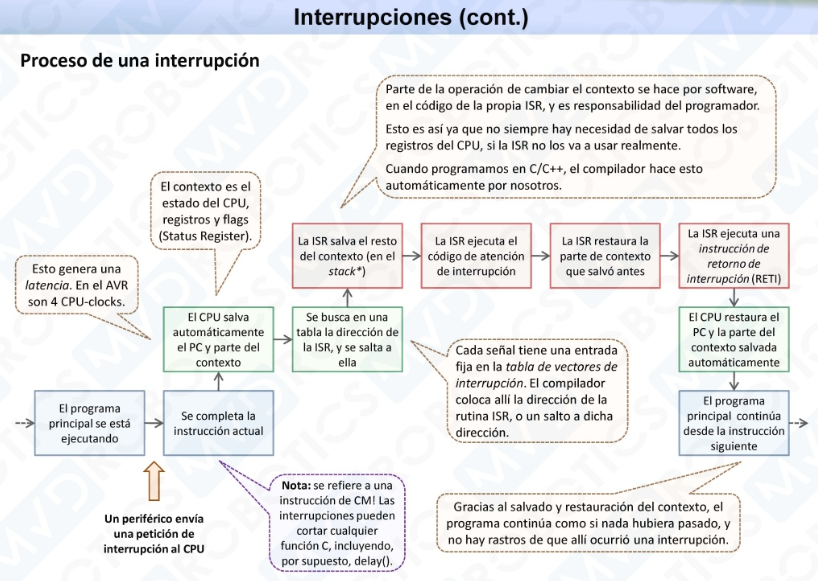
**Problemas del polling**

* Consume CPU.
* Se pueden perder eventos.
* Para no perder eventos hay que aumentar la frecuencia de consulta, y eso hace que el CPU no pueda realizar otras tareas entretanto.
* No tiene prioridad, todos los dispositivos son tratados igual.
* Es incompatible con las funciones de ahorro de ahorro de energía de uC.
* En una aplicación sencilla, con pocos inputs, esta bien, pero para manejar una gran cantidad de entradas es inviable.

**Ventajas de la interrupción**

* Ahorra CPU. Evita perder tiempo esperando que ocurran cosas.
* Permite al procesador hacer más cosas “simultáneamente”.
* No se pierden eventos.
* Permite tratar algunos eventos con más prioridad que otros.
* Permite responder a los eventos más rápido, si latencia.
* Permite poner el uC en sleep y aun asi responder a ciertos eventos.
* Simplifica el diseño de la aplicación, aunque hay que cambiar totalmente el modo de pensar.

**Proceso de una interrupción**



**Tópicos y terminología**

**Prioridad**

 Las interrupciones pueden tener prioridad, es decir, algunos eventos pueden deﬁnirse como “más importantes” que otros. Cuando ocurren los 2 eventos a la vez, se atiende primero el que tiene prioridad más alta.

**Enmascarar**

Enmascarar una señal (una ﬁiente de interrupción) consiste en bloquear de forma individualizada su recepción por parte del CPU. De esta manera podemos inhabilitar las interrupciones que no queremos que se disparen, ya sea permanentemente, o en forma temporal durante una “sección crítica” del programa.

**Anidamiento**

interrupciones anidadas es cuando ocurre una interrupción adentro de otra interrupción, es decir, una interrupción puede interrumpir a otra. Normalmente este modo de operación está inhabilitado, pero a Veces se puede activar explícitamente. No recomendado para principiantes.

**Latencia**

La latencia de interrupción es el tiempo que tarda el CPU en responder a una solicitud de interrupción. Este tiempo se debe en parte al proceso intrínseco de la interrupción, pero también puede aumentar debido a que el CPU está atendiendo otra interrupción, por ejemplo. Una interrupción que está esperando una respuesta del procesador, se dice que está pendiente.

**Vectorización**

Esto signiﬁca simplemente que cada fuente de interrupción posee una rutina de atención (ISR) especializada, propia, que se ubica en algún lugar de la memoria (de programa) que el programador puede asociar de alguna manera. Lo contrario a esto es que todas las interrupciones sean atendidas por la misma rutina, y que dentro de ella se haga polling para determinar el evento que la originó.

**Fuentes de interrupción**

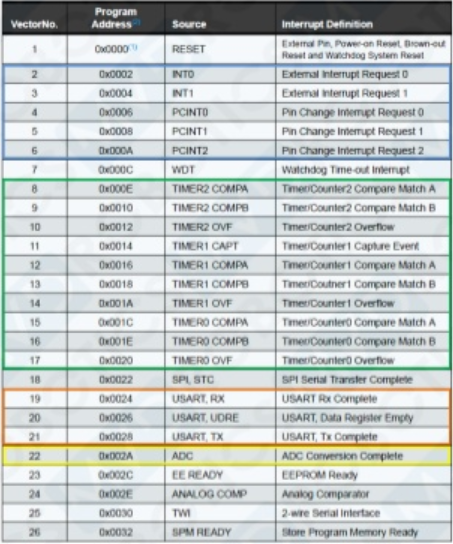
En el AVR, diversas condiciones pueden generar una interrupción, por ejemplo:

* El ADC, para indicar que esta pronta la conversión;
* Los timers, para indicar que se produjo un overflow u otra condición;
* El controlador UART, para indicar que llego información por el bus;
* El cambio de estado de un pin de E/S -interrupcion externo- para multiples usos;
* Poner en LOW el pin de reset,etc.

**Tabla de vectores de interrupción**

Cada evento tiene una dirección de memoria en donde se aloja el código especifico a ejecutar, en la llamada tabla de vectores de interrupción, que se ubica al comienzo de la memoria de programa.

En esa dirección, el compilador coloca un “jmp” a la dirección de la ISR, que puede estar en cualquier otro lado.

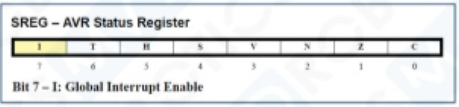


**Global Interrupt Enable bit (l-bit)**

Este bit se encuentra en el registro de estado (SREG), y se encarga de habilitar o deshabilitar globalmentetodas las interrupciones.

Al iniciar el sistema el bit se encuentra por defecto apagado.

Al entrar en una ISR, el I-bit se apaga automáticamente, y se vuelve a prender al salir con la instrucción RETI.



Dentro de la ISR el bit se puede prender manualmente, lo que tiene el efecto de habilitar las nested interrupts.

En assembler, existen dos instrucciones especiales para aprender y apagar este bit: SEI y CLI, respectivamente. En C/C++, hay dos macros.

**Individual interrupt enable bit (mask)**

Además del bit global, cada fuente de interrupción tiene su propio tiene su propio bit local para habilitarla o inhibirla. Este bit esta en uno de los registros de configuración del periférico correspondiente, y se suele llamar mask (mascara).

**Interrupt Flag**

Casi todas las fuentes de interrupción tienen, además del bit de mascara, un flag.

El flag es un bit que se prende cuando se produce el evento propiamente dicho, independientemente de que este activada o no la mascara o las interrupciones globales.

El flag se apaga cuando es llamada la ISR correspondiente, o también se puede apagar por software, escribiendo un 1.

Por ejemplo, aunque no esté activada la interrupción de overflow del timer1, el flag correspondiente se va a activar cuando se produzca un overflow del timer1, y permanecerá activado hasta que se habilite la interrupción, o hasta que se apague a mano.

En algunos casos el flag se apaga no al disparar la ISR, sino al leer cierto registro.

**¿Qué pasa si 2 eventos ocurren a la vez,o si uno o mas eventos ocurren durante una ISR?**

* Toda vez que están habilitadas las interrupciones globales, y esta habilitada la máscara correspondiente, si el flag de una interrupción se activa, la interrupción se procesa (en realidad no importa el orden en que se activen estas 3 cosas).
* Si hay más de un flag activado, las interrupciones pendientes se procesan en orden de prioridad.
* Cuando un evento ocurre durante un ISR: si están habilitadas las interrupciones globales, la interrupción se procesa en ese momento, es decir ocurre una interrupción adentro de otra de lo contrario -el caso más normal- la interrupción se procesa al salir de la ISR actual.
* Esto es valido incluso si el evento en cuestión es del mismo tipo que la ISR que se está ejecutando.
* Obsérvese que, no obstante, si no esta activado el nesting y ocurre mas de un evento del mismo tipo durante una ISR, a la salida de esta se procesa una única interrupción de ese evento, es decir se pueden perder eventos.
* Por el contrario, si esta activado el nesting y se producen varios eventos del mismo tipo, cada uno de ellos interrumpirá al anterior, y acabará produciéndose un desbordamiento de pila.

**Como definir una ISR**

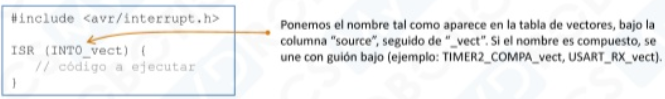
Para utilizar una interrupción necesitamos hacer 3 cosas:

1. Habilitar las interrupciones globales.
2. Habilitar la máscara de la interrupción especifica.
3. Escribir una función ISR y asociarla (“Linearla”) a esa interrupción.

Este ultimo paso, si trabajaremos en assembler seria engorroso, porque habría que ocuparse del contexto, de la instrucción RETI al final, y del linking. Pero en cambio en c/c++, con la librería avrlibc, es relativamente fácil.

Primero que nada, todos los nombres de los vectores de interrupción de todos los micros AVR están definidos en dicha librería, así como lo estaban los nombres de los registros.

Para definir una ISR, usamos estos nombres en combinación con el macro ISR(), de la siguiente manera:



Opcionalmente, podemos usar los parámetros ISR\_BLOCK, ISR\_NOBLOCK, ISR\_NAKED y ISR\_ALIASOF(vect).

