Principios de mecatrónica
Clase 12

SDI-11561-004, SDI-11561-002

Interrupciones

Una interrupción es una situación especial que suspende la ejecución de un programa de modo que el sistema pueda realizar una acción para tratarla. Tal situación se da, por ejemplo, cuando un periférico requiere la atención del procesador para realizar una operación de E/S.

Las interrupciones constituyen quizá el mecanismo más importante para la conexión del microcontrolador con el mundo exterior, sincronizando la ejecución de programas con acontecimientos externos.

De esta forma un **interrupción** se pueden definir como: eventos que hacen que el microcontrolador deje de ejecutar la tarea que esta realizando para atender dicho acontecimiento y luego regresarse a continuar la tarea que estaba realizando antes de la interrupción. Siendo un mecanismo que permite ejecutar un bloque de instrucciones deteniendo la ejecución del programa y luego restablecer la ejecución del mismo sin afectarlo directamente.

En los microcontraldores AVR el programa asociado con la interrupción es llamada Rutina de servicio de interrupción (ISR).

Rutina de servicio de interrupción (ISR)

Cuando una interrpción es invocada, el microcontrolador ejecuta el servicio de rutina de interrupción. Generalmente, en muchos microcontroladores, para cada interrupción hay una localidad fija en memoria que guarda la dirección de su ISR. El grupo de ubicaciones de memoria reservado en AVR32 contiendo las direcciones de las ISR's denominado vector de interrupciones, se muestra en la siguiente tabla.

Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface (I2C)	0026
Store Program Memory Ready	0028

Pasos en la ejecuación de una interrupción

Tras la activación de una interrupción, el microcontrolador realiza los siguientes pasos:

i) Finaliza la instrucción que está ejecutando actualmente y guarda la dirección de la siguiente instrucción (contador de programa) en una stack.

II) Brinca a una localidad fija en la memoria llamada *interrupted vector table*. La tabla de vectores de interrupción dirige el microcontrolador a la dirección del ISR.

III) El microcontrolador empieza a ejecutar la "subrutina de servicio de interrupción" hasta que alcanza la última instrucción de la subrutina, la cual es RETI (volver de la interrupción).

IV) Al ejecutar la instrucción RETI, el microcontrolador regresa al lugar donde se interrumpió. Primero, obtiene la dirección del contador del programa(PC) y entonces empieza a ejecutarse desde esa dirección.

Fuente de interrupciones en el AVR

Hay muchas fuentes de interrupciones en el AVR, dependiendo de cual periférico es incorporado en el chip. Las siguientes son algunas de las fuentes más utilizadas de interrupción en el AVR.

- Hay al menos dos interrupciones reservadas para cada uno de los timers, una para overflow (desbordamiento) y otra para comparación.
- Se reservan tres interrupciones para interrupciones externas. Los pines PD2 (PORTD.2), PD3 (PROTD.3) y PB2 (PORTDB.2) son para la interrupción de hardware externo INT0, INT1 e INT2, respecrivamente.
- Comunicaciones seriales USART tiene 3 interrupciones, una para recibir y dos interrupciones para transmitir.
- Convertidor analógico digital.

De la tabla anterior podemos notar que un número limitado para de bytes están reservados para interrupciones.

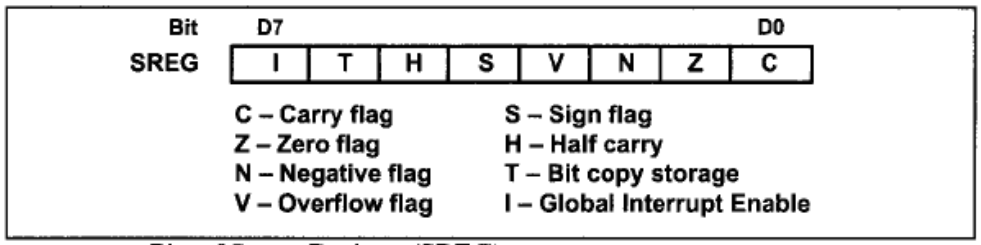
Normalmente, la rutina de servicio para una interrupción es demasiado larga para encajar en el espacio de memoria asignado. Por este motivo,

una instrucción JMP se coloca en el punto de la tabla de vectores a la dirección del ISR.

Por este motivo, se coloca el JMP como primera instrucción y redirigimos el procesador fuera de la tabla de vectores de interrupción.

Habilitar y deshabilitar una interrupción

Al reiniciar, todas las interrupciones se desactivan (enmascaran), lo que significa que el microcontrolador no responderá a ninguna de ellas si están activadas. Las interrupciones deben estar habilitadas (desenmascaradas) por software para que el microcontrolador responda a ellas. El bit D7 del SREG (Status register) es responsable de habilitar y deshabitar las interrupciones globalmente. De esta forma, el bit I hace el trabajo de deshabitar y habilitar todas las interrupciones fácilmente. Con una sola instrucción "CLI" (Clear Interrupt), nosotros podemos hacer I=0 durante la operación de una tarea critica.



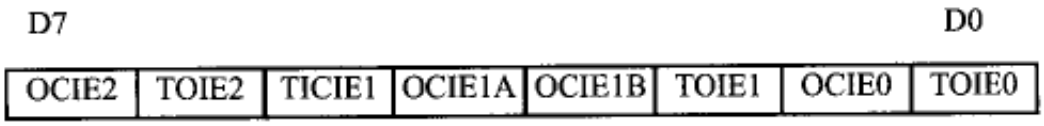
Bits of Status Register (SREG)

Pasos para habilitar una interrupción

Para habilitar cualquiera de las interrupciones, nosotros realizamos los siguientes pasos:

1) Bit D7 (I) del registro SREG debe configurarse en ALTO para permitir que ocurran las interrupciones. Esto se realiza con la instrucción "SEI'' (Set Interrupt).

II) Si $I = 1$, cada interrupción se habilita configurando en ALTO el bit de indicador de habilitación de interrupción (IE) para esa interrupción. Hay algunos registros de I/O que contienen los bits de habilitación de interrupción. La siguiente figura muestra que el registro TIMSK tiene bits de habilitación de interrupción para Timer0, Timer1 y Timer2. Examinaremos el registro que contiene los bits de habilitación de algunas interrupciones. Se debe tener en cuenta que si I = 0, no se responderá a ninguna interrupción, incluso si los bits de habilitación de interrupción correspondientes son altos.



TOIE0 Timer0 overflow interrupt enable
= 0 Disables Timer0 overflow interrupt
= 1 Enables Timer0 overflow interrupt

OCIE0 Timer0 output compare match interrupt enable
= 0 Disables Timer0 compare match interrupt
= 1 Enables Timer0 compare match interrupt

TOIE1 Timer1 overflow interrupt enable
= 0 Disables Timer1 overflow interrupt
= 1 Enables Timer1 overflow interrupt

OCIE1B Timer1 output compare B match interrupt enable
= 0 Disables Timer1 compare B match interrupt
= 1 Enables Timer1 compare B match interrupt

OCIE1A Timer1 output compare A match interrupt enable
= 0 Disables Timer1 compare A match interrupt
= 1 Enables Timer1 compare A match interrupt

TICIE1 Timer1 input capture interrupt enable
= 0 Disables Timer1 input capture interrupt
= 1 Enables Timer1 input capture interrupt

TOIE2 Timer2 overflow interrupt enable
= 0 Disables Timer2 overflow interrupt
= 1 Enables Timer2 overflow interrupt

OCIE2 Timer2 output compare match interrupt enable
= 0 Disables Timer2 compare match interrupt
= 1 Enables Timer2 compare match interrupt

Figura 1: Registro TIMSK (Mascara de interrupciones de los timers)

Los bits de la tabla, junto con el bit I, deben ser altos para que se pueda responder a una interrupción. Tras la activación de la interrupción, el propio AVR borra el bit I para asegurarse de que otra interrupción no pueda interrumpir el microcontrolador mientras da servicio al actual. Al final de ISR, la instrucción RETI hará que I = 1 permita que entre otra interrupción.

Ejemplo: Mostrar las instrucciones para

a) Habilitar (unmask) la interrupción del overflow del Timer0 e interrupción de compare match del Timer2.

b) Inhabilitar (mask) la interrpción overflow del Timer0, entonces

c) mostrar como inhabilitar (mask) todas las interrupciones con una sola instrucción.

```
(a)   LDI R20,(1<<TOIE0)|(1<<OCIE2) ;TOIE0 = 1, OCIE2 = 1
      OUT TIMSK,R20  ;enable Timer0 overflow and Timer2 compare match
      SEI                          ;allow interrupts to come in

(b)   IN    R20,TIMSK              ;R20 = TIMSK
      ANDI  R20,0xFF^(1<<TOIE0)    ;TOIE0 = 0
      OUT   TIMSK,R20              ;mask (disable) Timer0 interrupt
```

Además, podemos realizar la acción (b) con las siguientes instrucciones

```
      IN    R20,TIMSK          ;R20 = TIMSK
      CBR   R20,1<<TOIE0       ;TOIE0 = 0
      OUT   TIMSK,R20          ;mask (disable) Timer0 interrupt
                               .
(c)   CLI                      ;mask all interrupts globally
```

Es importante notar que en a) se puede usar "$LDI,\ 0x81$" en lugar de LDI R20, "$(1 << TOIE0)|(1 << OCIE2)$" .

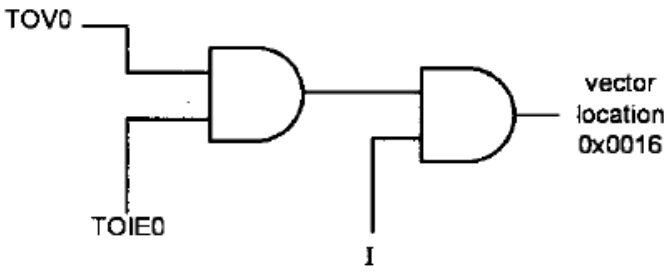
En secciones pasadas, aprendimos que la bandera de desbordamiento del timer se activaba cuando el timer se desbordaba. También mostramos cómo monitorear la bandera del timer con la instrucción SBRS R20,

TOV0. En el sondeo de TOV0, tenemos que esperar hasta que se levante TOV0.

El uso de interrupciones evita atar al controlador. Si la interrupción del timer en el registro de interrupción está habilitada, TOV0 se eleva cada vez que el timer se desborda y el microcontrolador salta a la tabla de vectores de interrupción para dar servicio al ISR. De esta manera, el microcontrolador puede hacer otras cosas hasta que se le notifique que el temporizador se ha desbordado. Para usar una interrupción, primero debemos habilitar la interrupción porque todas las interrupciones se enmascaran al reiniciar. El bit TOIEx habilita la interrupción para un timer dado. Los bits TOIEx son retenidos por el registro TIMSK como se muestra en la tabla.

Timer Interrupt Flag Bits and Associated Registers

Interrupt	Overflow Flag Bit	Register	Enable Bit	Register
Timer0	TOV0	TIFR	TOIE0	TIMSK
Timer1	TOV1	TIFR	TOIE1	TIMSK
Timer2	TOV2	TIFR	TOIE2	TIMSK



Consideraciones

• Debemos evitar utilizar el espacio de memoria asignado a la tabla de vectores de interrupción. Por lo tanto, colocamos todos los códigos de inicialización en la memoria comenzando en una dirección como $100. De esta forma se debe usar inicialmente la instrucción JMP en la dirección 0000 para redirigir al controlador lejos de la tabla de vectores de interrupción.

Ej.- Realizar un programa donde se asuma que PORTC esta conectado a 8 switches y PORT D a 8 leds. De esta forma el programa usará Timer0 para generar una onda cuadrada sobre el pin PORTB.5, mientras al mismo tiempo los datos son transferidos del PORTC al PORTD.

```
.INCLUDE "M32DEF.INC"
.ORG  0x0          ;location for reset
      JMP   MAIN
.ORG  0x16         ;location for Timer0 overflow (see Table 10.1)
      JMP   T0_OV_ISR          ;jump to ISR for Timer0
;-main program for initialization and keeping CPU busy
.ORG  0x100
MAIN: SBI   DDRB,5             ;PB5 as an output
      LDI   R20,(1<<TOIE0)
      OUT   TIMSK,R20    ;enable Timer0 overflow interrupt
      SEI                ;set I (enable interrupts globally)
      LDI   R20,-32      ;timer value for 4 µs
      OUT   TCNT0,R20    ;load Timer0 with -32
      LDI   R20,0x01
      OUT   TCCR0,R20    ;Normal, internal clock, no prescaler
      LDI   R20,0x00
      OUT   DDRC,R20     ;make PORTC input
      LDI   R20,0xFF
      OUT   DDRD,R20     ;make PORTD output
;--------------- Infinite loop
HERE: IN    R20,PINC     ;read from PORTC
      OUT   PORTD,R20    ;give it to PORTD
      JMP   HERE         ;keeping CPU busy waiting for interrupt

;--------------ISR for Timer0 (it is executed every 4 µs)
.ORG  0x200
T0_OV_ISR:
      IN    R16,PORTB    ;read PORTB
      LDI   R17,0x20     ;00100000 for toggling PB5
      EOR   R16,R17
      OUT   PORTB,R16    ;toggle PB5
      LDI   R16,-32      ;timer value for 4 µs
      OUT   TCNT0,R16    ;load Timer0 with -32 (for next round)
      RETI               ;return from interrupt
```