Consideraciones Especiales

Esta página cubre algunas consideraciones especiales a tener en cuenta cuando se trabaja con interrupciones en MCU AVR.

Compartir datos con el ISR

Las variables compartidas entre el ISR y el programa principal deben declararse como **volátiles** y tener un alcance **global .** Al compilar usando el optimizador, en un bucle como el siguiente:

```c
uint8_t flag;                ?
...
ISR(SOME_vect) {
    flag = 1;
}
...
while(flag == 0) {
...
}
```

el compilador generalmente accederá a "bandera" solo una vez y optimizará los accesos adicionales por completo, ya que su análisis de ruta de código muestra que nada dentro del ciclo podría cambiar el valor de "bandera" de todos modos. Para decirle al compilador que esta variable podría cambiarse fuera del alcance de su análisis de ruta de código (por ejemplo, dentro de una rutina de servicio de interrupción), la variable debe declararse así:

```c
volatile uint8_t flag;       ?
...
ISR(SOME_vect) {
    flag = 1;
}
...
while(flag == 0) {
...
}
```

Cuando la variable se declara **volátil** como se indicó anteriormente, el compilador se asegura de que dondequiera que se actualice o lea la variable, siempre escribirá los cambios en la memoria SRAM y leerá la variable desde SRAM.

Operaciones de datos atómicos

Para que una operación sea considerada **atómica** , debe garantizar el acceso **ininterrumpido** de una determinada variable. Muchos lenguajes ensambladores brindan esto en ciertos niveles, es decir, prueba y configuración de bits, sin embargo, no existe **ninguna disposición** para proporcionar automáticamente la atomicidad de todos los tipos de variables en el lenguaje ANSI C.

Las expresiones y declaraciones ANSI-C **no son atómicas** .

Este problema puede ser problemático (en ciertas situaciones) cuando **las variables de varios bytes se comparten** con un ISR. Si bien declarar una variable de este tipo como **volátil** garantiza que el compilador no optimizará los accesos a ella, no garantiza el acceso **atómico** a ella. Considere el siguiente ejemplo de código:

```c
#include <stdint.h>  ?
#include <avr/io.h>
#include <avr/interrupt

volatile uint16_t ctr;

ISR(TIMER1_OVF_vect)
{
  ctr--;
}
...
int
main(void)
{
    ...
    ctr = 0x0200;
    start_timer();
    while(ctr != 0)
      // wait
        ;
    ...
}
```

Existe la posibilidad de que el contexto principal salga de su ciclo `while()` cuando la variable **ctr** alcance el valor 0x00FF. Esto sucede porque el compilador no puede acceder de forma nativa a una variable de 16 bits de forma atómica en una CPU de 8

bits. Entonces, cuando **ctr** está, por ejemplo, en 0x0100, el compilador luego prueba el byte bajo para 0, lo que tiene éxito. Luego procede a probar el byte alto, pero en ese momento se activa el ISR y el contexto principal se interrumpe. El ISR disminuirá la variable de 0x0100 a 0x00FF, luego continúa el contexto principal. Ahora prueba el byte alto de la variable que (ahora) también es 0, por lo que concluye que la variable ha llegado a 0 y finaliza el ciclo.

Macros de acceso atómico

La biblioteca atómica (http://www.nongnu.org/avr-libc/user-manual/group__util__atomic.html) AVR-LIBC proporciona las macros ATOMIC_BLOCK que insertan la protección de interrupción adecuada cuando se desea acceso atómico. Estas macros operan a través de la manipulación automática del bit de **estado de interrupción global (I) del registro SREG** . Las rutas de salida de ambos tipos de bloques se gestionan automáticamente sin necesidad de consideraciones especiales, es decir, el estado de interrupción se restaurará al mismo valor que tenía al entrar en el bloque respectivo. Usando las macros de este archivo de encabezado, el código anterior se puede reescribir como:

```c
#include <stdint.h>?
#include <avr/io.h>
#include <avr/interru
#include <util/atomic

volatile uint16_t ctr

ISR(TIMER1_OVF_vect)
{
  ctr--;
}
...
int main(void)
{
   uint_16 ctr_copy;
dieciséis
   ...
   ctr = 0x0200;
   start_timer();
   do
   {
     ATOMIC_BLOCK(ATC
     {
       ctr_copy = ctr
     }
   } while(ctr != 0);
     // wait
       ;
   ...
}
```

La macro **ATOMIC_BLOCK** instalará la protección de interrupción adecuada antes de acceder a la variable **ctr** , por lo que se garantiza que se probará de manera consistente. En este caso, el parámetro **ATOMIC_RESTORESTATE** hace que ATOMIC_BLOCK restaure el estado anterior del registro SREG, guardado antes de que se deshabilitara el bit indicador de estado de interrupción global. El efecto neto de esto es hacer que el contenido de ATOMIC_BLOCK sea atómico garantizado, sin cambiar el estado del indicador de estado de interrupción global cuando se completa la ejecución del bloque.

Aprende más

Resumen de interrupciones de megaAVR®

Más información > (/8avr:interrupts-mega-overview)

Configuración de interrupciones megaAVR®

Más información > (/8avr:interrupts-mega-configuration)

Ejemplo de interrupción de megaAVR®

Más información > (/8avr:interrupts-mega-example)