

COMENTARIO TÉCNICO

Buceando en los MCUs Freescale.....



Por Ing. Daniel Di Lella
Dedicated Distributor Field Application Engineer
For Freescale Semiconductor Products
Dto. Ingeniería de Electrocomponentes S.A.
fae@electrocomponentes.com
dilella@arnet.com.ar

“Serie Flexis”.... “Como migrar de 8 a 32 Bits sin traumas”

Tercera Entrega.



¡Hola Amigos! En el artículo anterior, habíamos empezado con un ejemplo de migración de la familia S08 a la familia V1 ColdFire. Allí quedó planteado que al compilar un proyecto hecho para S08 y querer migrarlo a V1 Coldfire se generan múltiples errores.

Veremos ahora como solucionarlos uno a uno.

Tip de Migración 1.

Código Assembler dentro del código del programa.

Cuando se migra a diferentes arquitecturas el uso de instrucciones en assembler es ineficiente. Por ejemplo, los “core” S08 de 8 bits y V1 ColdFire de 32 bits poseen diferentes sets de instrucciones debido a sus muy diferentes arquitecturas. Si el código del programa posee algunas líneas en assembler, esto llevará al compilador a emitir mensajes de error por operaciones no válidas.

Para evitar ello, es necesario reemplazar dichas líneas de código en assembler por su equivalente en código “C” de fácil conversión por el compilador.

```
void main(void) {  
    //asm BSET 0,PTEDD  
    PTEDD_PTEDD0 = 1;  
    PTEDD_PTEDD1 = 1;  
    RTC_Init();  
    KBI_Init();  
    EnableInterrupts; /* enable interrupts */  
    /* include your code here */  
  
    for(;;) {  
        RESET_WATCHDOG(); /* feeds the dog */  
        //asm stop;  
        _Stop;  
    } /* loop forever */  
    /* please make sure that you never leave main */  
}
```

Notece que en el ejemplo de código assembler para configurar el bit 0 del registro de datos de direcciones del Port E ha sido reemplazado por su equivalente de instrucciones en código C.

Se han reemplazado las instrucciones “Stop” y “Wait” utilizadas para colocar al MCU en bajo consumo, por las definiciones “_Stop” y “_Wait”. Estas definiciones estarán contenidas en la definición del derivativo (por ejemplo, MCF51QE128.h) y deben actualizarse cuando se migra de la familia S08 a V1 ColdFire. Por medio del uso de estas definiciones para las instrucciones Stop y Wait se asegura que el código trabaje en ambos “cores”.

Si no hay código en assembler, el proyecto compilará, sin embargo, ello no es garantía de que el código realmente trabaje para ambas familias, ya que hay más cosas para chequear.

Tip de Migración 2.

Asignación de Vectores de Interrupción usando Declaraciones de Interrupción en los “Header Files” del CodeWarrior.

Las tablas de Vectores de interrupción entre la familia S08 de 8 Bits y la V1 ColdFire de 32 Bits no son idénticas y residen en diferentes lugares de memoria, por lo que la asignación de vectores no coincidirá en el espacio de memoria. La asignación de los vectores de Interrupción mantiene un número de vector relativo entre S08 y V1 ColdFire.

La tabla 3 nos muestra un ejemplo de las diferencias en el vector de interrupción para el RTC (Vrtc).

| | Address | Vector |
|-------------|---------------|--------|
| MC9S08QE128 | 0xFFCE | 24 |
| MCF51QE128 | 0x(00)00_0158 | 86 |

Tabla 3 – Dirección Vs Vector.

Error Común:

1. Resolver el problema con la ecuación:

`Número de Vector del V1 ColdFire = Número de Vector del S08 + 62.`

2. Asignación inapropiada de un vector de interrupción que solo funciona para el S08:

`Interrupt 24 RTC_ISR {} // Solamente trabaja para el S08.`

La solución de migración es usar la declaración de número de interrupción “**VectorNumber_Vname**” definida en los archivos “header” del entorno CodeWarrior, según se puede ver en la **Figura 13**.

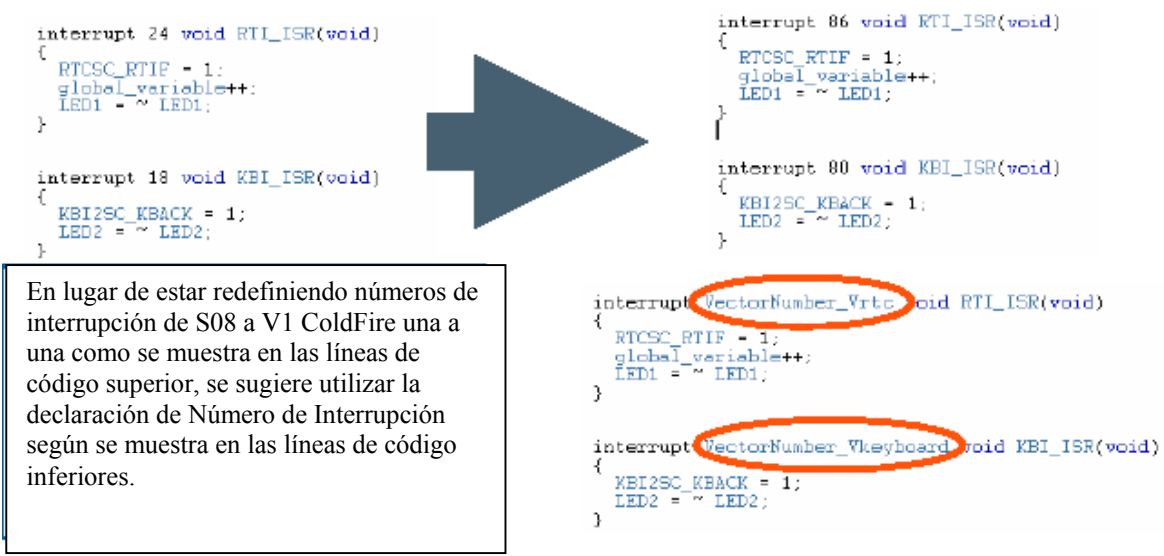


Figura 13.- Modo correcto de Asignación de Vectores.

Tip de Migración 3.

Referencia de memoria usando las declaraciones de periféricos “Register_Bitname” en los “Header Files” del entorno CodeWarrior.

Los mapas de memoria entre la familia S08 y la V1 ColdFire son diferentes, de esta forma, declaraciones absolutas de memoria no coincidirán con el espacio de memoria. La Figura 14 muestra las diferencias entre los mapas de memorias de ambas familias.

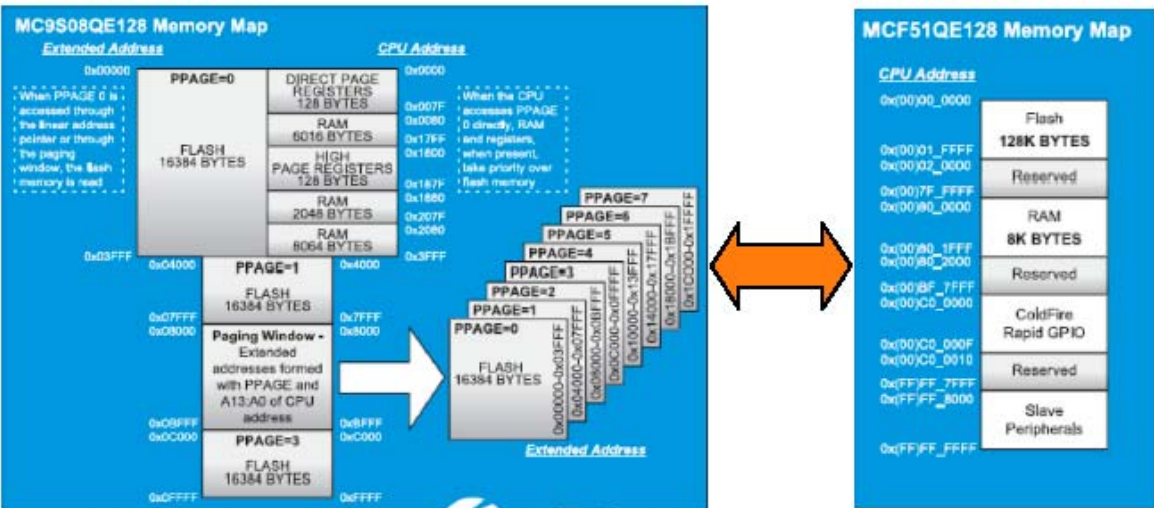


Figura 14.- Diferencias entre los mapas de memoria del S08 y V1 ColdFire.

La asignación de la memoria RAM y FLASH queda determinada por los respectivos archivos del linker. Los mapas de los registros de periféricos mantienen el direccionamiento relativo entre S08 y V1 ColdFire. Por ejemplo, en el MC9S08QE128 el registro SCI1C1 está localizado en la dirección 0x0022 y el mismo registro en el MCF51QE128 está localizado en la dirección $0x(\text{FF})\text{FF_}8000 + 0x0022 = 0x(\text{FF})\text{FF_}8022$.

Errores Comunes:

1. Un ejemplo de las declaraciones absolutas de memoria hechas en forma impróprias es:

```
int var @ 0x400 = 1;
```

2. Y en una variable global (global_variable):

```
unsigned int near global_variable @ 0x80 = 0;
```

La solución de migración en la referencia de memoria es utilizar las declaraciones “Register_Bitname” en los archivos “Header” del CodeWarrior y permitirle al linker que pueda colocar variables en espacios de memoria disponibles. Remover el direccionamiento absoluto y la declaración “near”, de la siguiente manera:

```
Unsigned int global_variable = 0;
```

A continuación se puede efectuar un “Make” (compilar) y un Debug del código. El proyecto debería ahora funcionar sin problemas en el V1 ColdFire.

Algo importante de hacer notar es que los errores de migración dados en el Tip 2 y Tip 3 no son marcados como errores por el CodeWarrior.

Bueno amigos, en el próximo artículo, veremos algunos Tips más a tener en cuenta y otras cositas útiles que nos harán más fácil la migración de S08 a V1 ColdFire.....

Hasta la próxima ;!!