Cika Electrónica

Nota de Aplicación: CAN-068
Título: **Logging en FRAM y con Ramtron VRS51L3074**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	14/02/07	

En esta nota, analizamos cuestiones de performance y durabilidad en algunas implementaciones de sistemas de logging, para luego desarrollar un ejemplo de uso de la FRAM interna del VRS51L3074.

Frecuencia y velocidad de escritura, durabilidad

Supongamos por ejemplo tener un totalizador de caudal. El mismo toma mediciones y totaliza cada un determinado tiempo, digamos un segundo. Dicho totalizador se alimenta del lazo de corriente 4-20mA que genera el transmisor de caudal, por lo que está sujeto a ser pateado y desconectado por operadores y otras personas que no deberían pero transitan por esa zona. Si bien podemos darnos el lujo de perder alguna que otra medición, no podemos perder el total y empezar de nuevo, por lo que este sistema deberá utilizar memoria no-volátil. Veamos cómo desarrollamos esto.

Una opción es mantener una alimentación de backup para la RAM o para el micro. Un inconveniente que tiene esto es que tenemos consumo de energía para mantener la información, y un mantenimiento obligado cuando se agota la batería, que si necesitamos que sea preventivo necesitará de un sistema de detección o visitas periódicas, más un obligado tiempo de fuera de servicio necesario para reemplazar la batería; además del riesgo de que se pierda el total por un error o mal funcionamiento.

Otra alternativa es implementar el totalizado sobre una EEPROM. Dado que las mediciones son, dijimos, cada un segundo, no nos preocupa el excesivo tiempo de borrado de la EEPROM, pero si escribimos cada un segundo en la misma ubicación, fatigaremos la memoria al cabo de aproximadamente 10^5 segundos (algo más de un día), por lo que deberemos implementar algún sistema que nos permita ir rotando la posición de escritura y encontrar después el dato verdadero. Vemos que por cada año de duración y byte de resolución, deberemos emplear algo menos de 365 bytes, es decir que para lograr 10 años con un número de 32-bits necesitaríamos una memoria de unos 16KB. Una solución posible es detectar la pérdida de alimentación y grabar solamente en ese instante. Esto disminuye aparentemente la cantidad de escrituras necesarias pero no tiene en cuenta la posible complejidad adicional del sistema para detectar la pérdida de alimentación y obrar inmediatamente y asume que es posible incorporar capacitores lo suficientemente grandes como para mantener la alimentación por el tiempo que dure la escritura. En nuestro caso calculamos que debería entregar entre 1 y 10 mA (memoria, micro) durante unos 50ms, lo cual requiere un capacitor de entre 100 y 1000uF, aproximadamente[1], sobre la alimentación de memoria y micro, o de una batería. Esto no es posible en sistemas de seguridad intrínseca. Si pensamos en resolver el problema del tiempo de escritura utilizando una flash, descubriremos que se compensa con que su durabilidad es 10 veces menor.

FRAM al rescate

En este caso, una FRAM puede ser accedida más de 10^{13} veces como mínimo, por lo que cumple con este requisito sin inconvenientes ni preocupaciones, no es necesario detección temprana, ni capacitores, ni complicar el circuito.

Corolario

Si tenemos una aplicación que necesita escribir mucha información y/o hacerlo muy rápido en un tiempo muy corto, con posibilidad de pérdida de alimentación; digamos, por ejemplo, una sonda de exploración, una caja negra, un registro de choque. Del cálculo anterior, podemos deducir que para más de 100 bytes por segundo, o más de un byte en 10ms, no será posible de realizar con una EEPROM, por lo que deberemos recurrir a un

1 Asumimos una caída de tensión de 0,5V

sistema que escriba en RAM y luego copie, o batería de respaldo, etc. Una vez más, una FRAM I^2C, por ejemplo, puede realizar más de 5000 escrituras por segundo (una cada 200us), en un período de 40 años, y la FRAM interna del VRS51L3074 puede ser accedida en el orden del microsegundo.

Buffers circulares

Supongamos que tenemos un sistema que debe almacenar registros que indican un determinado suceso; por ejemplo alarmas de un sistema de control industrial, ingresos en un sistema de control de acceso, posiciones y eventos en un AVL, etc. El sistema almacenará una determinada cantidad de estos registros, y una vez lleno el buffer, por lo general, dado que la información nueva suele ser más importante que la vieja en muchas aplicaciones, sobreescribirá a la información anterior, por lo que podemos implementar esto como un buffer circular. En muchos de estos sistemas, la información almacenada debe sobrevivir a pérdidas casuales y/o intencionales de la alimentación, por lo que se lo debe implementar en algún tipo de memoria no volátil.

Supongamos que pensamos en resolver el problema con una EEPROM. En un buffer circular, el puntero se actualiza a cada acceso, por lo que no podemos almacenarlo en una posición fija sino que deberemos implementar un esquema para evitar la fatiga de la memoria, de igual modo que analizáramos para el caso del totalizador. Sin duda, esto nos limita el espacio disponible para almacenar registros, por lo que suele descartarse en favor de un 'tag' que indica el primer registro libre en el buffer. Este sistema requiere que al inicializar, se deba hacer una búsqueda por todo el buffer hasta encontrar el *tag*, lo cual requiere de un cierto tiempo de inicialización que no todos los sistemas se pueden dar el lujo de derrochar. Por ejemplo, en un buffer de 16KB con memorias I^2C, necesitaremos de unos 164ms[2] en promedio para encontrar el *tag*. Siempre asumiendo que es posible poner un *tag*, que no hacemos otra cosa mientras lo buscamos, y que el bus puede funcionar a esa velocidad. Este sistema escribe en el mismo byte a cada vuelta del buffer, por lo que dependiendo de la frecuencia de escritura de los registros, y de la longitud de los mismos, determinaremos el tiempo de mantenimiento (reemplazo de la EEPROM). Por ejemplo, para registros de 16bytes a 1 por segundo, la fatiga de la memoria ocurrirá aproximadamente al cabo de 3 años[3].

FRAM al rescate

Una vez más, con FRAM no necesitamos complicar el trabajo, dado que podemos guardar y recuperar el puntero del buffer en FRAM, varias veces por segundo, con una durabilidad mayor a los 100 años, dado que disponemos de 10^{13} accesos como límite, en el peor de los casos. Si bien debemos considerar además la lectura, seguimos cumpliendo con creces una durabilidad excepcional.

En el caso particular del VRS51L3074, dado que la FRAM está en el mapa de memoria del micro (lo cual también se puede hacer con otros micros que tengan posibilidad de conexión al bus y FRAM de acceso paralelo), podemos implementar este buffer circular de la misma forma que si estuviera en RAM, operando directamente sobre el espacio de memoria del micro, y desentendiéndonos del tema de la no-volatilidad.

Ejemplo

A continuación veremos un ejemplo de manejo de buffers circulares sobre la FRAM de un VRS51L3074. Para simplificar la operatoria, haremos que el buffer contenga un número entero de registros. Esto es posible si todos los registros tienen la misma longitud.

Definición del buffer y los punteros en FRAM

Tendremos, como vimos, el buffer y los punteros en FRAM. A fin de minimizar la operatoria sobre la FRAM, no sólo por la durabilidad sino por la velocidad de ejecución, es conveniente mantener una copia en RAM de los punteros, e ir actualizando conforme se los modifica, como veremos más adelante.

```
#define NUMRECORDS      12
#define RECORD_SZ       16

typedef struct {
  unsigned char dat[RECORD_SZ];
} logrecord;

struct ptrstruct {
```

2 16KB/2 * 8 * 1/400KHz
3 16384/16 * 10^5 segundos

```
        unsigned int index;
        unsigned int count;
        unsigned int rdindex;
        unsigned int rdcount;
};

__xdata at 0x8000 logrecord record_buffer[NUMRECORDS];
__xdata at 0x9F00 struct ptrstruct bufpos;

logrecord rcrdbfr;
unsigned int rcrdidx,rcrdcnt;
unsigned int rdrcrdidx,rdrcrdcnt;
```

Ingreso de un registro

Copiamos el registro en FRAM y modificamos el puntero y cuenta de registros; si el buffer se hallaba lleno y sobreescribimos un registro viejo, deberemos actualizar también puntero y cuenta de registros extraídos.

```
void putrecord(void)
{
        memcpy((void *)&record_buffer[rcrdidx],(void *) &rcrdbfr,sizeof(logrecord));
                                        // escribe registro
        rcrdcnt++;                      // incrementa cuenta
        if(++rcrdidx>=NUMRECORDS)       // mantiene circularidad
                rcrdidx=0;
        if(rcrdcnt >= NUMRECORDS){      // buffer lleno: sobreescribe registros viejos
                rcrdcnt=NUMRECORDS;
                if(rdrcrdcnt)           // actualiza puntero de extracción
                  rdrcrdcnt--;
                if(!rdrcrdcnt)
                  rdrcrdidx=rcrdidx;
        }
        // update index and record count in FRAM
        bufpos.index=rcrdidx;
        bufpos.count=rcrdcnt;
        bufpos.rdindex=rdrcrdidx;
        bufpos.rdcount=rdrcrdcnt;
}
```

Lectura de un registro

Simplemente buscamos la posición en FRAM donde comienza el registro deseado y lo copiamos al buffer

```
void readrecord(unsigned int idx)
{
register int j;

        j=rcrdidx-rcrdcnt+idx;
        if(j<0)
               j+=NUMRECORDS;
        else if(j>=NUMRECORDS)
               j-=NUMRECORDS;

        memcpy((void *)&rcrdbfr,(void *)&record_buffer[j],sizeof(logrecord));
}
```

Extracción de un registro

Copiamos el registro al buffer y corregimos puntero de lectura y cuenta de registros extraídos

```
int getrecord(void)
{
        if((rcrdcnt-rdrcrdcnt)<=0)
          return(-1);
        memcpy((void *)&rcrdbfr,(void *)&record_buffer[rdrcrdidx],sizeof(logrecord));
        rdrcrdcnt++;                    // incrementa cuenta de leídos
        if(rdrcrdcnt >= rcrdcnt) {      // si no hay más, resetea índice
                rdrcrdcnt=rcrdcnt;
                rdrcrdidx=rcrdidx;
        }
        else if(++rdrcrdidx>=NUMRECORDS)     // caso contrario mantiene circularidad
                rdrcrdidx=0;
        // actualiza copia en FRAM
        bufpos.rdindex=rdrcrdidx;
        bufpos.rdcount=rdrcrdcnt;
```

```
        return(rcrdcnt-rdrcrdcnt);
}
```

Inicialización

Al inicializar el sistema, deberemos activar la FRAM y recuperar los punteros y cuenta de registros de la misma

```
        DEVMEMCFG  |= 0xC0;                    // Activa FRAM
        rcrdidx=bufpos.index;                  // Recupera punteros y cuenta de FRAM
        rcrdcnt=bufpos.count;
        rdrcrdidx=bufpos.rdindex;
        rdrcrdcnt=bufpos.rdcount;
```

Ejemplo de uso

Para escribir, debemos llamar a *putrecord()* con el registro armado en el buffer de RAM. Para otras opciones, se requerirá de leves modificaciones a la función.

```
        for(i=0;i<NUMRECORDS;i++){
          sprintf((void *)rcrdbfr,"Record #%d",i+1);
          putrecord();
        }
```

Para leer, simplemente llamamos a *readrecord()* con el número de registro, entre *0* y *rcrdcnt-1*, que queremos leer

```
        for(i=0;i<rcrdcnt;i++)
          readrecord(i);
```

Para extraer registros de la cola, llamamos a *getrecord()* hasta que devuelva el valor *-1*

```
        i=0;
        while(getrecord()>0)
          i++;
```

Tarea para el hogar

El sistema descripto tiene algunas falencias, pero cumple con los requisitos didácticos de una nota de aplicación, siempre y cuando los índices y cuentas hayan sido puestos a cero alguna vez.

Puede ocurrir, que una pérdida de alimentación ocurra en un instante tal que el micro interrumpa su ejecución justo en el momento que actualiza los punteros o cuenta de registros, pudiendo dejar todo el buffer inservible. Esto es así debido a que dichas variables son multi-byte, y para colmo de males todas estas variables mantienen una coherencia entre sí, por lo que no debe interrumpirse la escritura desde al primero hasta el último byte. Para evitar esta catástrofe, tenemos algunas alternativas:

- Antes de iniciar la escritura, hacer una detección del estado de la alimentación principal y estimar el tiempo restante
- Implementar algún sistema de recuperación, mediante el cual mantendremos, por ejemplo, una copia de seguridad de estas variables, la cual se actualizará antes de proceder a la modificación de las "oficiales". Ante una pérdida de alimentación, el sistema determina al inicio si debe utilizar la copia principal o la de respaldo.

También puede ocurrir que hagamos una extracción del buffer pero nunca lleguemos a poder utilizar realmente ese registro, sea por no poder reportarlo (pérdida de conexión), o utilizarlo internamente (pérdida de alimentación). En ese caso, deberemos partir la función de extracción de registros en una que hace la lectura propiamente dicha, y otra que actualiza puntero y cuenta de registros, una vez operado sobre éste. En el caso que ingrese otro registro o registros mientras estamos intentando reportar o utilizar el primero, a buffer lleno, el sistema sobreescribirá los registros más antiguos y actualizará los punteros, por lo que esto no será problema.

En caso de no ser posible mantener registros de igual longitud, ya no podemos manejarnos con una estructura de registros, nuestro buffer será simplemente un array de bytes. Además, deberemos modificar el programa para soportar la copia de parte del registro en la parte 'superior' del buffer y el resto en la parte 'inferior' del buffer, tanto al momento de escribir como de leer. A fin de determinar la longitud del registro antes de leerlo, podemos utilizar, por comodidad, el primer byte del mismo.