Mejorar la inmunidad electromagnética con un mejor firmware

Artículo cedido por Cemdal

CEMDAL

www.cemdal.com

Autor: Francesc Daura Luna, Ingeniero Industrial. Director de la Consultoría CEMDAL. Representante de CFC para España y Portugal.
www.cemdal.com
fdaura@cemdal.com
www.cfcele.com

Los ingenieros de diseño de software que trabajan con microcontroladores (μC) identifican como firmware o software empotrado a la programación muy cercana al hardware del μC, lejos de la programación de alto nivel como los sistemas operativos sobre los que se ejecutan las aplicaciones. Algunos conceptos explicados seguidamente también se pueden aplicar usando lenguajes de alto nivel como: C, C++, Perl, Java, Erlang, Python, Ruby, Haskell, Swift, etc. Aunque trabajan junto a los ingenieros de hardware, los ingenieros de software, a veces, no son totalmente conscientes de que con un buen diseño del firmware pueden ayudar a su colega de diseño de hardware en la solución de los problemas de inmunidad electromagnética (EM).

Una parte de las pruebas que deben realizarse en los sistemas electrónicos para ser conformes con la directiva europea de compatibilidad electromagnética (CEM) son pruebas de inmunidad EM (concepto opuesto a susceptibilidad EM), Las pruebas de inmunidad más usuales son: inmunidad radiada de 80 MHz a 1 GHz, inmunidad conducida de 150 kHz a 80 MHz, ráfagas de transitorios rápidos (EFT), sobretensiones ("surges"), bajadas de tensión ("dips"), interrupciones de tensión ("drops") y descargas electrostáticas (ESD).

Una vez se tiene un buen hardware con un óptimo diseño de CEM, el diseñador del firmware puede ayudar a incrementar un mayor nivel de inmunidad EM. Debemos negar los mitos siguientes: el diseñador de software no se debe preocupar por la CEM y con el software se puede solucionar todo. Por el contrario, podemos afirmar:

- el software es intrínsecamente inmune
- Un buen diseño del firmware no es efectivo contra los problemas de emisiones conducidas o radiadas, aunque en algún caso puede detectarse alguna pequeña diferencia
- si surge algún problema de inmunidad, el firmware puede ayudar a solventarlo
- Un buen diseño del firmware es muy efectivo contra la susceptibilidad, sobre todo contra ESD y EFT
- un software bien diseñado puede disimular un mal diseño de hardware
- un software mal diseñado puede destrozar un buen diseño de hardware

Un contribuyente importante a la mejora de la inmunidad en los sistemas electrónicos basados en μC es el diseño de firmware reforzado con técnicas adecuadas. Los problemas inducidos por perturbaciones tipo ESD, EFT y campos radiados se deberían considerar tan pronto como sea posible en la fase de diseño, orientando el firmware a la mejora de la inmunidad EM para aumentar la seguridad y la fiabilidad. El firmware reforzado para la CEM es barato de aplicar, mejora el rendimiento en cuanto a la inmunidad y ahorra costes de hardware y de desarrollo. Los problemas inducidos por las perturbaciones más usuales son:

- El μC no responde
- Pérdida de las comunicaciones con dispositivos externos
- Datos incorrectos en el visualizador
- Bloqueo de la funcionalidad
- Pérdida de la seguridad funcional
- Respuestas inesperadas de la funcionalidad del producto
- Pérdidas de fiabilidad
- Ejecución inesperada de procesos incorrectos
- Pérdida de la integridad de los datos
- Cambios en el contador del programa
- Ejecución incorrecta de instrucciones inesperadas
- Punteros de memoria en direcciones incorrectas
- Mala ejecución de subrutinas
- Reinicios esporádicos y/o interrupciones aleatorias
- Corrupción en la configuración de μC
- Desprogramación y lecturas corruptas de valores de las entradas y las salidas

El objetivo de las mejoras en el firmware consiste en "cazar" cualquier error antes de que éste cause un mal funcionamiento del sistema. Después de "cazarlo" se pueden aplicar técnicas de recuperación. El firmware no puede prevenir la destrucción debida a las ESD, pero sí puede evitar errores no destructivos. El firmware se debe diseñar de forma que continuamente verifique que se detectan errores y que se toman las acciones de recuperación adecuadas. Se estima que un buen firmware puede reducir los errores y bloqueos en más de diez veces con respecto a un firmware que no tenga en cuenta estas técnicas. Lógicamente ello tiene un coste: el código es aproximadamente un 10% mayor y se ejecuta en un tiempo algo más largo, pero compensa con creces todo lo que evita.

Debe quedar claro para el diseñador del "firmware" que las medidas de corrección no deben enmascarar cualquier error interno de diseño. Así, el diseñador debe comprobar que su sistema funciona correctamente deshabilitando todas las medidas de corrección. Estas medidas de corrección deben sólo actuar ante una perturbación externa, nunca ante reiterados errores internos de ejecución debidos a un mal diseño del firmware o del hardware.

Errores en el flujo de programa

La detección de errores en el flujo de programa consiste en la comprobación continua del programa preguntándose por dos condiciones : ¿el programa está necesitando demasiado tiempo?, ¿está dentro de un rango de memoria válido? Esta comprobación sólo necesita unas instrucciones de más y algo más de tiempo. También debe haber rutinas de recuperación contra los bloqueos o errores transitorios en el seguimiento del flujo de programa.

Otra técnica consiste en añadir «testigos» ("tokens") a las rutinas. Se salva un testigo cuando se entra en una rutina y se comprueba. Si no es el mismo, se ha detectado un error, dado que se ha entrado en la rutina incorrectamente. También se puede usar el concepto de «firma», dejando en algunos registros unos datos posteriormente comprobables para ver si han cambiado o no.

Si no se ha pasado por estos puntos de comprobación, o si no se ha reiniciado el temporizador "Watch-Dog Timer" (perro guardián) (WDT) o si se

ha perdido alguna rutina, es razonable asumir que ha ocurrido un error y que se deben tomar medidas correctoras. Además siempre se debe usar un WDT junto con rutinas que periódicamente comprueben si el flujo de programa es correcto.

Nunca se debe asumir que el estado de un puerto de E/S, un registro, una memoria, etc, no ha cambiado. Por el contrario se debe asumir lo contrario. Por ejemplo, si debe usarse un registro de índice, debemos preguntarnos que ocurriría si este dato fuera incorrecto. Si la consecuencia es que un LED no luce correctamente, no es importante, pero si es más seria, tal como el paro de un motor, deben tomarse medidas para evitarlo.

El flujo se puede verificar a intervalos, añadiendo algunos puntos de comprobación. Aquí, un programa independiente interrumpe periódicamente el programa principal, y comprueba si se ha pasado por algunos puntos de prueba preestablecidos. Si no es así, se pasa a ejecutar una rutina de gestión de errores.

Las interrupciones no usadas del μC son a veces fuente de problemas. Si un transitorio obliga a interrumpir el flujo normal del programa y a saltar a cualquier posición de éste, es difícil prever sus consecuencias. La solución es poner una instrucción de salto a una rutina de recuperación de errores en todas las posiciones de entrada de los vectores de interrupción no usados. La forma más sencilla es tener una rutina de reinicio ("reset"), aunque a veces es inaceptable y se debe ser más cuidadoso e intentar volver donde se estaba, causando el menor cambio posible.

Como resumen, para comprobar que el flujo de programa es correcto:

1. Comprobar periódicamente los punteros de la pila en el programa principal
2. Dentro de una rutina se deben comprobar los punteros de la pila ("stack") antes de ejecutar una instrucción de retorno. Ello asegura que las rutinas y subrutinas vuelven correctamente a donde han sido llamadas.
3. Además de o en lugar de comprobar los punteros de la pila, usar testigos ("tokens") para ayudar a detectar problemas en el flujo de programa. Cuando se entra en un rutina, se memoriza el valor testigo y al salir se comprueba.
4. Disponer códigos "trampa" en áreas prohibidas, tales como tablas de códigos o vectores de interrupción no usados. Si un programa intenta ejecutar estos códigos, caerá en la trampa. Un buen ejemplo es disponer instrucciones de retorno en localizaciones no utilizadas.
5. Debe usarse una rutina de interrupción de temporización que nunca se para o se deshabilita por el programa para verificar que el programa principal está operativo.
6. Se puede usar un temporizador interno o uno externo a modo de perro guardián (WDT). Si periódicamente no se habilita este circuito, provoca un reinicio del μC. El programa principal debe periódicamente comprobar el temporizador, para verificar que está funcionando correctamente

Detección de errores en las entradas/salidas

Los errores en las entradas se pueden controlar mediante un filtro por hardware y luego por firmware. Una técnica de firmware muy simple consiste en leer los datos en la entrada varias veces sucesivamente con un retardo de tres o más lecturas y hacer la media entre ellas, si son analógicas (a través del convertidor A/D), o dar como válido el valor digital coincidente con la mayoría de las lecturas realizadas en un intervalo de tiempo predeterminado (técnica de los votos). Para una buena protección contra las ESD, es suficiente aplicar un retardo de unas centenas de nanosegundos.

Esto es equivalente a un filtro pasobajos. Muchas veces es suficiente con no ejecutar ninguna acción si detectamos que los datos de entrada no son válidos.

Se pueden comprobar por diferentes vías los datos que se envían a las salidas, volviéndolas a leer a través de los propios puertos bidireccionales (si el μC lo permite) o usando circuitería adicional externa; por ejemplo, utilizando FIFOS con entrada en paralelo que recogen los datos de las salidas del μC y enviando en serie estos datos de nuevo a una entrada del μC para su comprobación.

Para comprobar que las entradas y las salidas son correctas podemos:

1. Comprobar la paridad, la estructura y/o el "checksum" y códigos de corrección de errores de las entradas
2. Verificar que las entradas tienen valores razonables. Algunos códigos recibidos son claramente erróneos dentro de una ventana de valores
3. Muestrear todas las entradas como mínimo dos o tres veces para poder filtrarlas mediante filtro vía firmware.
4. Las salidas se pueden comprobar haciendo que el receptor las devuelva los datos enviados para su comprobación.
5. Cualquier receptor debe enviar la correcta aceptación al emisor de todos los datos enviados. Si no se recibe esta aceptación, el emisor debe volver a transmitir repetidamente los datos.

Estas técnicas no pueden restaurar los datos en todas las circunstancias, pero si pueden evitar males mayores. Si cualquiera de las comprobaciones anteriores fallan, debe restaurarse el sistema. La restauración no debe ser equivalente a ejecutar las rutinas de inicialización general al poner en marcha el sistema o cuando se ejecuta un reinicio. Por ejemplo, la RAM no deberá borrarse. De hecho esto debe prevenirse cuando el contador de programa cambie debido a una ESD o a un EFT. Ello es posible comprobando un registro de "flag" antes de ejecutar las rutinas de inicialización general. El "flag" = 1 se dispone cuando se ejecuta el programa principal y normalmente solo de borra cuando se ejecuta un reinicio de hardware. Específicamente, la restauración usualmente debe hacer lo siguiente:

1. Reinicio del puntero de la pila.
2. Reinicio de los punteros FIFO.
3. Reinicio de los contadores.
4. Prevenir la transmisión de códigos sospechosos.
5. Deshabilitar las interrupciones hasta que la restauración sea completa. Entonces rehabilitarlas de nuevo y poner en marcha el temporizador WDT.
6. Reinicio de los "flags" de interrupción pendientes.
7. Refrescar las salidas.
8. Si el sistema principal lo acepta, es una buena idea enviarle un código, informándole que ha habido una restauración. Entonces, el sistema principal puede asegurarse de que todas las partes del sistema concuerdan.
9. Y, por último, que la rutina de restauración debe eliminar el problema específico que ha causado la restauración.

Detección de errores en la memoria

Para detectar errores en los datos en las memorias se pueden usar comprobaciones CRC ("Cyclical Redundancy Checks": Comprobación Redundante Cíclica), o códigos de corrección de errores como los códigos de Hamming, que pueden detectar y corregir errores. Por ejemplo, añadiendo 6 bits extra por cada palabra de 16 bits, se pueden detectar errores de 1 o 2 bits y se pueden corregir errores de 1 bit, pero no todo a la vez. El empleo de bits de paridad, "checksums" y códigos de corrección de errores, puede prevenir la memorización de datos erróneos debidos a ESD o EFT.

Estas técnicas no pueden restaurar los datos en todas las circunstancias pero, si se usan, se pueden evitar males mayores y tener que volver a transmitir. Las posiciones de memoria no utilizadas deben rellenarse con un conjunto de instrucciones NOP y de retorno RTN al comienzo de la rutina de inicialización, cada cierto número de NOP. Si el programa realiza un salto a alguna de estas posiciones, después de ejecutar algunas instrucciones NOP retornará a la rutina de inicialización.

Existen varias fuentes posibles para ordenar un reinicio interno: el LVD ("Low Voltage Device": Detector de baja tensión), o el reinicio del WDT, el POR ("Power On Reset": Reinicio al encender), el reinicio en caliente (reinicio externo después de un tener el pin de reinicio en estado bajo). La fuente de reinicio se marca en un "registro de reinicios" y esta información se mantiene mientras la fuente de alimentación del μC esté encendida.

En todas estas fuentes conviene programas correctamente sus parámetros para evitar problemas, en algunas especialmente durante el transitorio de puesta en marcha.

Como resumen, para comprobar que los valores memorizados y la información es correcta:

1. Periódicamente comprobar los valores guardados de forma redundante y si no coinciden, reiniciar el μC. Específicamente, los "flags" (banderas) de estado, los "flags" de habilitación y deshabilitación deben memorizarse redundantemente. Se pueden usar técnicas de corrección de errores en lugar de la redundancia, o ambas al mismo tiempo.
2. Debe comprobarse el valor o el rango de los registros de índices u otros registros importantes, antes de guardarse en RAM.
3. Si existe demasiada información crítica para utilizar la redundancia y si no es posible comprobar sus valores, entonces conviene recurrir al empleo de "checksums" o Comprobación Redundante Cíclica (CRC) para comprobar grandes bloques de datos.

Todos los datos almacenados en la memoria interna o externa pueden ser susceptibles a corromperse debido a perturbaciones CEM en condiciones extremas. La técnica preventiva de almacenar valores complementarios duplicados en áreas de memoria no adyacentes, almacenando y comprobando los bits de paridad o usando un ECC (código de corrección de errores) son todos métodos avanzados que ayudan a identificar y/o corregir la corrupción de los datos.

Técnicas de refresco

Cuando se refresca el estado de cualquier registro o memoria, el programador no tiene en cuenta la historia anterior al refresco. Sólo se asegura de que los datos son correctos para el siguiente paso. Por ejemplo, antes de leer un puerto en el μC se debe reprogramar como entrada, aunque el puerto se haya programado como tal en la inicialización general y en el principio del bucle principal del programa. No se debe asumir que el puerto está todavía programado como entrada.

Antes de asignar un valor a un puerto de salida, es conveniente reprogramarlo como salida, aunque el puerto haya sido programado como salida en el inicio. Debido a una ESD o un EFT podría desprogramarse y pasar a ser un puerto de entrada. Otras técnicas de refresco que se pueden considerar son:

1. Rehabilitar las interrupciones a intervalos regulares.
2. Refrescar los niveles de bits de stop cuando se usan los puertos como salidas de datos en serie.
3. Refrescar los estados de los puertos de salida regularmente.
4. Leer las entradas de control y de selección para asegurar que el sistema funciona en la forma adecuada.
5. En los μC con bancos de memoria es conveniente refrescar el registro de selección de banco en cada bucle de programa.

Cuando se refresca, también es importante tener en cuenta el orden en que se realiza el proceso de refresco. Algunas veces una acción de refresco debe realizarse antes que otra. Siempre se debe pensar en las consecuencias de un error debido a un cambio en el orden de las acciones de refresco. No todos los fallos en los μC son debidos a las perturbaciones en general. Otras fuentes de problemas son las conexiones intermitentes o errores en el diseño de la estructura del firmware.

La disminución de la inmunidad del hardware puede ser debida a un número insuficiente de capas en el circuito impreso, una mal asignación funcional de las capas del circuito, una mala estructura de masas y tierras, una mala estructura de desacoplo, etc. Pero estos puntos son a veces difíciles de mejorar solo a través de la mejora del firmware.

Técnicas comprobación y restauración

Algunas veces refrescar es insuficiente. En algunos casos el refresco puede enmascarar problemas muy serios. En estos casos, el registro, el puerto, los datos para enviar al visualizador, etc. deben comprobarse antes para determinar su estado. Si este estado es inadecuado, el programa debería intentar restaurarlo a su valor correcto. La restauración o la re-inicialización debe realizarse cuidadosamente. A pesar de que el estado del sistema pueda ser sospechoso, no es una buena idea borrar totalmente los datos y empezar de nuevo desde cero. Esto tendría como resultado la pérdida de toda la historia pasada de los datos. La guía a seguir sería volver a un estado anterior más estable del sistema. Cuando algún punto sea crítico, se puede memorizar redundantemente y usar la técnica de los votos. Esto es de tres memorizaciones, si dos son iguales y una es distinta, el valor bueno es el repetido dos veces. Si ninguno de los tres es igual, podemos suponer que los tres son incorrectos y dar como válido un valor de referencia por defecto. Si los tres son iguales, el valor es correcto.

Detección de errores y su corrección

En los equipos que incorporan algún tipo de visualizador de datos y errores, es importante analizar como se de-

Figura 1. Equipo controlado desde un ordenador PC vía una comunicación USB.

ben visualizar los errores de ejecución del programa. Se debe distinguir para quien se visualizan los errores: para el programador/desarrollador o para el usuario/cliente. En la visualización de errores de ejecución del programa, es conveniente establecer niveles de visualización. Podemos seguidamente clasificar tres niveles de visualización de errores:

Errores Nivel 1: útil sólo para el programador, para ayudarle a poner a punto el programa con un nivel de desarrollo suficientemente maduro, Nunca se deben realizar pruebas de inmunidad del equipo con este nivel. Podrían haber bloqueos del equipo por alguno de estos errores, si el programa no está totalmente terminado.

Errores Nivel 2: listado limitado de errores para ayudar al desarrollador del producto a poner a punto su conjunto de funciones. Se puede ir al laboratorio solo para realizar pruebas de pre-evaluación, nunca de certificación. En principio, cualquier error de este nivel no debería bloquear el producto durante la aplicación de las perturbaciones. Se debe asegurar la recuperación del funcionamiento correcto del equipo si desaparece la perturbación.

Errores Nivel 3: Listado muy limitado de errores enfocados para el usuario final. Este listado es un resumen de los errores más críticos desde el punto de vista funcional del equipo para ayudar al usuario/cliente en sus equivocaciones en el manejo del equipo. Ninguno de los errores de este nivel debe bloquear el equipo al aplicar las perturbaciones. Todos los errores de los niveles inferiores (1 y 2) se deben poder gestionar internamente sin visualizarlos, si su gestión se puede ejecutar en un tiempo muy corto, para que no aparezca de ningún modo en el sistema de visualización de datos del equipo. Veamos el ejemplo de la figura 1 en la que un equipo está controlado por un ordenador PC estándar. El ordenador no se debe someter a las pruebas de inmunidad y conviene que esté alejado del equipo bajo prueba. La comunicación en este ejemplo se establece vía USB entre el ordenado y el equipo.

El equipo (visualizador, bloqueos de movimientos habituales, apagado de pilotos luminosos, bloqueo de funciones diversas, etc. Este es el nivel adecuado para ir al laboratorio para pasar las pruebas de inmunidad. Para facilitar las pruebas de inmunidad EM, el equipo bajo prueba debería poder funcionar independientemente del ordenador, sin depender de la comunicación entre ellos.

Si no pueden ser independientes, se debe reforzar la comunicación entre ellos de forma que si se bloquea la comunicación durante la aplicación de la perturbación externa, el equipo debe poder seguir funcionando con un cierto nivel de independencia sin bloquearse, a la espera de recuperar la comunicación con el ordenador. Tanto el programa en el equipo como en el ordenador deben estar preparados para recuperar la comunicación entre ellos lo antes posible, una vez desaparezca la perturbación externa.

Se debe poder distinguir entre un fallo en el equipo bajo prueba, un fallo el ordenador o un fallo en las comunicaciones. Cualquier receptor debe enviar la correcta aceptación al emisor de todos los datos enviados. Si no se recibe esta aceptación, el emisor debe volver a transmitir repetidamente los datos. Durante las pruebas de inmunidad EM se pueden dar varios casos:

Caso 1: Si el fallo es solo del ordenador y se puede demostrar que el equipo sigue funcionado en cualquier circunstancia, el resultado de la prueba de inmunidad debe ser: PASA Nivel A porque no estamos probando el ordenador y el equipo sigue funcionando bien.

Caso 2: Si el fallo es de las comunicaciones o del equipo, pero se recuperan al desaparecer la perturbación externa, el resultado de la prueba debe ser: PASA Nivel B.

Caso 3: Si el fallo es de las comunicaciones o del equipo, pero no se recuperan al desaparecer la perturbación externa, el resultado de la prueba debe ser: NO PASA. Otra posibilidad sería adjudicar un PASA Nivel C si el sistema equipo-ordenador debe reiniciar manualmente. La decisión entre NO PASA y PASA Nivel C depende de las consecuencias en la imagen y reputación de "equipo fiable" que desee tener el fabricante del mismo.

Cuando las perturbaciones bloquean las comunicaciones o algún otro módulo periférico del programa principal en un ordenador que controla el equipo bajo prueba, una forma fácil de mejorar es repetir rutinas de re-inicialización dentro del bucle principal del programa general (figura 2). Si el tiempo de ejecución del bucle principal fuera excesivo, se puede repetir el bloque de rutinas de inicialización en una posición dentro del bucle equivalente al 50% de su tiempo de ejecución. Esta mejora reinicia continuamente todos los módulos periféricos como las comunicaciones, la visualización o los sensores, para evitar que lo deba hacer el usuario después de una perturbación. Se deben analizar las consecuencias en los tiempos de ejecución del programa.

El perro guardián (Watch-dog timer: WDT)

El resultado más grave de un transitorio es que el contador de programa o el registro de direcciones quede mo-

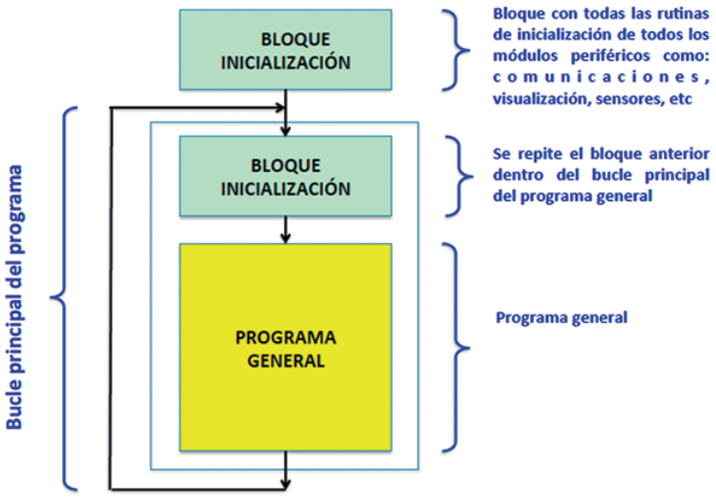

Figura 2. Repetición del bloque de inicialización dentro del bucle principal del programa.

dificado incorrectamente y el μC intente ejecutar códigos como instrucciones que realmente son datos o posiciones vacías de memoria. Esto pude causar que el μC entre en un bucle sin fin no haciendo nada o bien ejecutando instrucciones incorrectas que incluso pueden ser peligrosas para el equipo. El uso de un WDT es muy común y evita este problema. Como es sabido, consiste en un simple temporizador que si no es reiniciado por el μC cada cierto tiempo cuando el programa se está ejecutando correctamente, provoca un reinicio. El WDT puede ser interno al μC o externo.

Si el temporizador (interno o externo o los dos) no recibe un estímulo del μC antes del período fuera de tiempo, su salida fuerza un reinicio. El período fuera de tiempo debe ser suficientemente largo para que el μC no tenga que interrumpir rutinas con tiempos críticos para enviar los impulsos de reinicialización del WDT y que sea suficiente para que el μC pueda empezar la rutina de servicio del reinicio. De lo contrario continuamente actuaría el WDT y nunca terminaría el reinicio. Usualmente se trabaja con tiempos de entre 10 ms a unos 2 segundos.

Durante la ejecución de largas rutinas no conviene deshabilitar el WDT porque cualquier problema sucedido durante su ejecución imposibilitaría la recuperación. Es mejor insertar estímulos extra para el WDT en estas largas rutinas. No se debe actualizar el WDT durante una rutina de interrupción o dentro de cualquier bucle local sin protección por un límite de tiempo en el código.

Es esencial usar la entrada de "reset" del μC y no cualquier otra entrada tal como una interrupción incluso aunque no sea no-mascarable. El μC puede entrar en cualquier estado concebible cuando el WDT lo reinicia y debe retornarlo a un estado totalmente controlado.

El único estado que puede garantizar una apropiada inicialización es el RESET. A veces, es conveniente saber de donde viene la inicialización para ejecutarla de diferente forma en función del origen del reinicio. Hay μC que internamente ya facilitan esta información diferenciando el reinicio.

Cuando el μC entra en un estado catatónico debido a una perturbación externa no debe poder generar los estímulos para el WDT. El temporizador debe ser sensible sólo a los flancos y no al nivel, deberemos reiniciar regularmente todos los periféricos programables y al mismo tiempo el WDT. Así, si éste queda en un mal estado debido a una perturbación, aunque el μC funcione bien, provocará que el WDT reinicie y al mismo tiempo reprograme el puerto otra vez a su correcto estado.

Dos puntos críticos están en la inicialización y cuando se escribe en memorias EEPROM. Estos procesos conllevan varios milisegundos. Conviene calcular bien los tiempos de ejecución de todas las rutinas para saber los lugares óptimos.

El tiempo necesario para almacenar datos en la memoria no volátil (EEPROM de datos) es significativamente más largo que el necesario para almacenar datos en la memoria RAM. Durante este tiempo de programa, el sistema puede verse comprometido por perturbaciones externas, lo que puede dar lugar a un reinicio del sistema que termina el proceso de programa, resultando en la corrupción de datos. Para evitar esta situación, los datos deben ser almacenados en un contenedor redundante manteniendo su consistencia utilizando, entre otros, marcas de validación. La validez del contenido de este contenedor de datos debe comprobarse después de que cada sistema se inicie antes de utilizarlo.

La comprobación del correcto funcionamiento del WDT no es simple. No se pueden usar condiciones artificiales en el firmware porque no son representativas. Lo más adecuado es usar el firmware definitivo y someter al equipo a ESD repetidas con el suficiente nivel para provocar que el μC deje de funcionar bien, usando como prueba alguna modificación de hardware para que el μC sea más sensible (quitar filtros y alargar artificialmente algunas entradas con cables). Entonces debemos ver como el WDT responde adecuadamente. Un LED en el reinicio puede ser útil.

Otro tipo de circuitos WDT llamados "de ventana" protegen los μC contra los problemas de ejecutar el código demasiado rápido o demasiado lento. Los μC que ejecutan funciones críticas o relacionadas con la seguridad exigen un alto nivel de supervisión para garantizar que los fallos puedan detectarse y corregirse adecuadamente.

Los WDT de ventana ofrecen seguridad adicional advirtiendo al μC no sólo cuando cambia el WDT demasiado tarde, sino también cuando cambia demasiado pronto. Así, se deben especificar dos intervalos de tiempo, y su diferencia forma la ventana de tiempo de vigilancia (figura 3).

La seguridad funcional y la norma IEC 61508

La seguridad funcional de un equipo puede quedar afectada por los problemas de inmunidad EM. Una forma clara de asegurar la seguridad funcional es seguir las recomendaciones de la norma IEC 61508 para la "seguridad funcional" de los equipos electrónicos eléctricos, electrónicos y programables relacionados con la seguridad y está destinada ser una norma de seguridad funcional básica aplicable a todo tipo de industria. Incluye recomendaciones

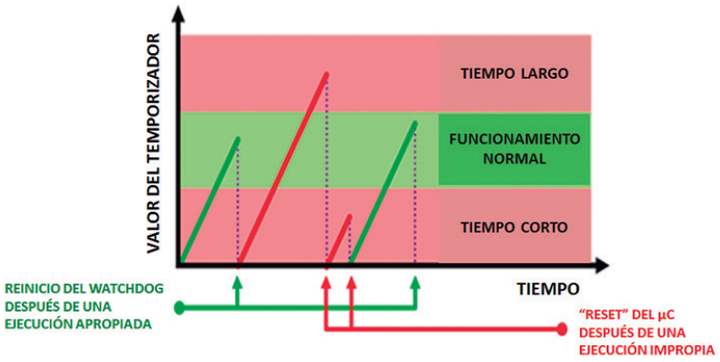

Figura 3. Watchdog de ventana

de mejora de la seguridad del software.

Un sistema electrónico funcionalmente seguro es aquel que responde como se espera para cada conjunto de entradas y se desarrollan y validan en función de criterios bien definidos de seguridad funcional. La norma inicial para la mayoría de las aplicaciones específicas de desarrollo de seguridad funcional en electrónica es la IEC 61508. Muchas nomas y pautas de seguridad específicas de varios mercado se han derivado de la IEC 61508: la ISO 26262 para vehículos de pasajeros (automoción), la EN50128 para el desarrollo de software de aplicaciones ferroviarias, la IEC61513 para centrales nucleares, la IEC61511 para la industria de procesos y su instrumentación asociada, la IEC62061 y la ISO 13849 para los sistemas de control eléctrico de maquinaria y la IEC62304 para equipos y sistemas médicos.

El hardware, el software y los sistemas pueden desarrollarse de acuerdo con la norma IEC 61508 con el objetivo de obtener un nivel SIL específico. Nivel SIL: "Safety Integrity Level". Los tres tipos de desarrollo deben cumplir con una serie de requisitos para gestionar el fallo sistemático. El hardware y los sistemas también se evalúan a través de métricas cuantitativas para determinar la probabilidad de fallo por hora y la fracción de fallo segura en la aplicación final. El software no tiene una tasa de fallos inherente y, como tal, sólo se evalúa de forma sistemática.

En la norma IEC 61508 destacan las partes 3 y 7 en lo que afecta al software. La parte 3 trata del software en el contexto del sistema y es uno de los aspectos obligatorios de la norma que debe ser abordado para que un sistema alcance una calificación SIL. A diferencia del hardware, el software no tiene desgaste ni ningún modo de fallo aleatorio. Se podría argumentar que un software perfectamente diseñado nunca falla. Y ahí radica el objetivo del nivel SIL para el software: se trata del nivel del proceso sistemático de prueba y desarrollo que determina el nivel SIL al que se puede evaluar el software. Cuanto más riguroso y sistemático sea el proceso de desarrollo, mayor es la calificación SIL que se puede lograr. El nivel SIL3 (probabilidad de fallo: >= 10-7 a < 10-8) tiende a ser el estándar más alto al que se desarrolla una amplia gama de sistemas. El nivel SIL4 (probabilidad de fallo: >= 10-8 a < 10-9) es un objetivo de seguridad ultra-alto, muy poco demandado por los proveedores de sistemas. Este nivel SIL4 es técnicamente imposible de lograr sin usar múltiples canales (como dos o más sistemas redundantes diferentes).

Conclusiones

Con las técnicas de buena programación preventivas y correctivas aplicadas al firmware aquí presentadas se puede ayudar eficazmente a superar las pruebas de inmunidad electromagnética. La tabla 1 aporta un resumen de técnicas preventivas y la tabla 2 aporta un resumen de técnicas de recuperación. Un buen método para obtener una alta fiabilidad y asegurar la seguridad funcional es seguir las recomendaciones de la norma IEC61508.

TÉCNICAS PREVENTIVAS	VENTAJA	DESVENTAJA	APLICACIÓN
watchdog (hardware o software)	El control es independiente de la CPU. Evita el bloqueo del µC	Se debe manejar con cuidado cuando se usa en modo de bajo consumo	Fácil pero las instrucciones de activación y refresco deben de insertarse cuidadosamente en el código para su máxima eficiencia
Forzar un reinicio del watchdog en el área de memoria de programa sin usar	Más directo y rápido que esperarell tiempo de espera del watchdog	Pérdida de contexto anterior	Borrar el bit de reinicio del watchdog (mirar especificaciones del dispositivo)
Rellenar la memoria de programa sin usar con instrucciones de interrupción de software	Más directo y rápido que esperar al tiempo de espera del watchdog	Ninguna	Rellenar las áreas sin usar con instrucciones de gestión de las interrupciones y gestionar los fallos en la rutina de la interrupción correspondiente
Rellenar la memoria de programa sin usar con instrucciones de retorno al inicio	Provoca un re-inicio del programa	Tiempo de re-inicio	Rellenar las áreas sin usar con instrucciones de re-inico intercalando también NOP's
Promediar de los valores del convertidor A/D	Asegura el correcto funcionamiento del convertidor A/D en entornos ruidosos	Mayor tiempo de procesado	Realizar un bucle iterativo para la adquisición y promediado del convertidor A/D
Eliminar códigos de operación ilegales o críticos	Evita bloqueos del µC debidos a lecturas inesperadas de códigos incorrectos	Ninguna, excepto la restricción en el uso de estos códigos de operación	Verificar el código de programa con las herramientas de desarrollo del software
Filtrado de la entrada de datos	Estabilidad en la adquisición de la información	Tiempo de procesado	Repetir las medidas muchas veces y realizar una elección estadística entre "1" ó "0"
Gestión de las interrupciones sin usar	Evita saltosincorrectos de código por interrupciones inesperadas	Ninguna	Muy fácil
Refresco de los registros críticos	Funcionamiento seguro	Utiliza recursos de la CPU	Refrescar los registros críticos en los bucles frecuentemente ejecutados

Tabla 1. Técnicas preventivas.

TÉCNICAS DE RECUPERACIÓN	VENTAJA	DESVENTAJA	APLICACIÓN
Control local del watchdog	Control del proceso de bloques secuenciales críticos	Se necesita un preciso calculo de una ventana de tiempo	Comprobar el tiempo de la secuencia de ejecución con los registros de tiempo del watchdog
Identificar fuentes de reinicio	Rápida recuperación de fallos de reinicio inesperados	Ninguna	Usar el registro de reinicio del µC o la memoria RAM para detectar las diferentes fuentes de reinicio
Guardar el contexto de la aplicación en memoria RAM, FLASH o EEPROM	Guarda los parámetros de la aplicación y asegura la reanudación de la ejecución de la tarea crítica en caso de fallo del µC	Usa recursos del µC	Guardar los parámetros y las fases críticas del software en la memoria RAM, FLASH o EEPROM. Usar la información de la RAM, FLASH o EEPROM para recuperar el último contexto antes del fallo

Tabla 2. Técnicas de recuperación.