关于功能安全编程的软件实现方法

Author: Zhanzr21 @ 21ic BBS

功能安全与信息安全其实是两个概念,两者都很重要但相互独立.在汽车电子设计中,两个安全都很重要,但是功能安全往往涉及到很严重的事故,所以显得更为重要.

功能安全-一般使用Safety这个词

信息安全-使用这个词:Security

但是到了中文两者都是安全,本文只涉及到功能安全,也就是前者Safety.

关于功能安全,业界一直在积极研究与推行相关标准.比如IEC,VDE.应该说功能安全的研究与发展永远不会停止,因为没有任何一种设计能够达到百分百的绝对安全标准.

功能安全与EMC测试联系较为紧密,因为电磁辐射会影响其他部件或者使用者的健康,而如何防范EMC噪音也是评价功能安全的一项指标.

# 兼容IEC,VDE的标准

IEC(International Electrotechnical Commission)是一个非营利,非政府的标准制定组织.IEC制定的标准主要关注安全,性能,环保,电气能效与再生能力.IEC与ISO和ITU有着紧密联系.这些制定的标准不仅包括对硬件的规定也有软件方面的.另外这些标准一般会根据应用场景细化为若干子标准.

除了老资格,国际化的IEC之外,这个领域中比较知名的,认可度较高还有德国的VDE,英国的IET,美国的IEEE.其中VDE还包括一个测试与认证机构专注于软件功能安全方面的前沿性研究.该机构属于德国的国家注册的认证机构.其主要目的在于给各家电子制品厂商提供标准符合与质量检验服务.

IEC的标准中最为人熟知的是IEC 60335-1.这个标准主要覆盖家用或类似场合的电子制品的功能安全与信息安全规范.其原则:被测品应该在各种元器件失效的情况下保持安全.从此标准的角度观察,微控制器(MCU)也属于众多器件之一.如果电子器件影响到最终产品的安全性,那么在连续的两次失效后该制品依然能够保持安全.这意味着该制品必须在微控制器不工作(正在复位或者运行异常)且硬件发生失效的情况下依然能保持安全.

如果安全取决于软件,那么软件被当做第二次失效来考虑.该标准规定了三种软件的安全类型:

Class A: 安全根本不取决于软件

Class B: 软件能够防范不安全的操作

Class C: 软件主动防范特殊危险

一般而言,软件设计中谈到的功能安全都是指的Class B,对于Class C,则需要额外的措施,比如双控制器冗余设计,本文暂不涉及.

是否符合Class B的认定与硬件软件都相关.与微控制器相关的因素被分为两组来考虑:微控制器相关与应用相关.

应用相关的部分依赖于用户的应用结构,必须由用户负责(通信,IO控制,中断,模拟输入与输出),微控制器相关的部分则纯粹与微控制器的结构相关,能够以一般的方法来进行(内核自诊断,挥发性与非挥发性存储器完整性检查,时钟系统测试等等).

对于英飞凌的微控制器产品来讲,微控制器部分的测试有着强大的硬件功能支持.比如Lock-Step内核的设计能够防范极为难以追钟的主控制器失效,Aurix系列的处理器都有丰富的存储器ECC校验功能,还有看门狗,SMU等等.

需要注意的是,除了IEC之外,还有数家其他机构在此领域的标准也是业内较为认可的:比如刚刚提到的德国VDE,英国IET,美国IEEE.本文为了篇幅原因,主要描述IEC的标准.其他组织的相关标准的方法与原理与之类似,但是细节可能有所差异.英飞凌的软硬件产品设计过程与此几家机构都有紧密的合作与联系.

与此同时,一些国家自己制定的标准也向国际标准借鉴.比如UL 60335-1, CSA 60335-1与EN-60335-1都是基于IEC 60335-1制定的.

[为甚么一些国家不直接使用国际标准而进行自定标准? 一般而言是为了设定贸易壁垒,保护本国的厂家.这些标准总体来讲跟国际标准非常类似,差别可以说几乎没有.但是认证方法,途径等等有一些本国化的特点.如果要进行本地标准的认证,最好找一些当地的专门从事该业务的机构进行合作.如果产品设计是根据国际标准来的,通过本地化的标准基本是没有问题的,只是要走个流程而已]

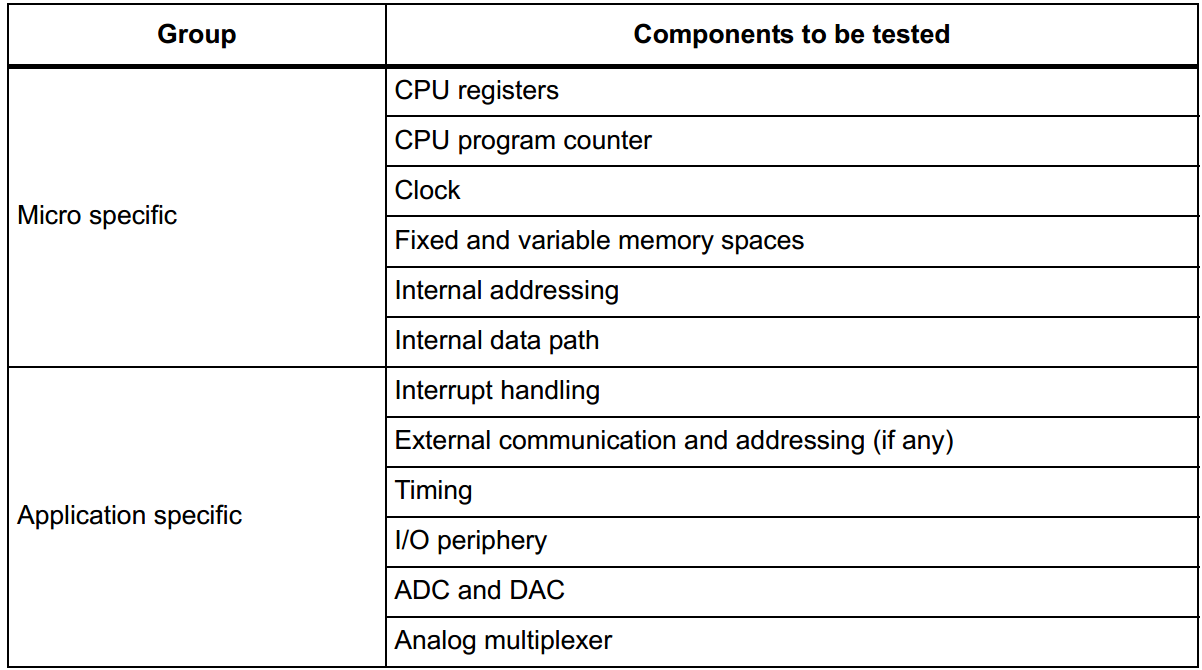


图 MCU中被Class B规范考虑的因素

为符合Class B的规范,软件上一般要考虑如下因素:

* CPU寄存器测试
* 时钟监视
* RAM功能检测
* Flash校验和完整性检测
* 看门狗自测
* 栈溢出监视

其中最后两条并非标准明文规定的,但是拥有这两条功能会提高软件的整体健壮性.

这些测试的一般工程实践:

|  |  |
| --- | --- |
| 测试项目 | 工程实践 |
| CPU寄存器测试 | 启动时检测core寄存器的读写,定期检测栈指针,一旦发生错误立即跳转至Fail Safe处理函数. |
| 程序计数器(PC) | 主要由看门狗来防止程序跑飞 |
| 寻址与数据路径 | 此项由RAM功能,Flash完整性测试与栈检查组成,栈底的最末端会写入一种特定的内容用于检测栈溢出,栈顶也会写入另一种特定内容用于检测栈underflow |
| 时钟 | 两组独立的时钟源相互检验 |
| NV存储 | 程序启动时即将整个NV空间进行CRC校验,如果有NV空间用作参数保持会在运行时被修改则也要考虑此因素,另外部分内容定期被检查.程序在烧写的时候一般会将计算好的CRC烧写在特定位置以作校验之用. |
| RAM存储 | 启动时即要进行遍历检查,另外关键变量使用双反向冗余方法进行保护(位相反的值被存储在非相邻的区域用作校验). |

为了提升用户通过Class B认证的速度,英飞凌为其微控制器提供了已经通过认证的Class B软件库供用户集成.(Aurix/Tricore与特定型号的相关库要通过厂商认证才能获取).当然需要提醒的一点是,用户即使使用已经认证的厂家软件库,整体应用是否符合认证要求依旧需要通过相应的机构进行认证.

至于以下内容属于应用相关的因素:

* 模拟:ADC/DAC
* GPIO
* 中断与外部通信
* 定时器
* 外部存储器寻址

模拟器件(ADC/DAC)与用户应用场景,特定的型号的外设能力相关度较大.一般而言应该定期检查相应的管脚.一些多余的模拟引脚可用于检测额外设定的检测点.如果使用了内部参考源,则该参考源也应当定期检测.

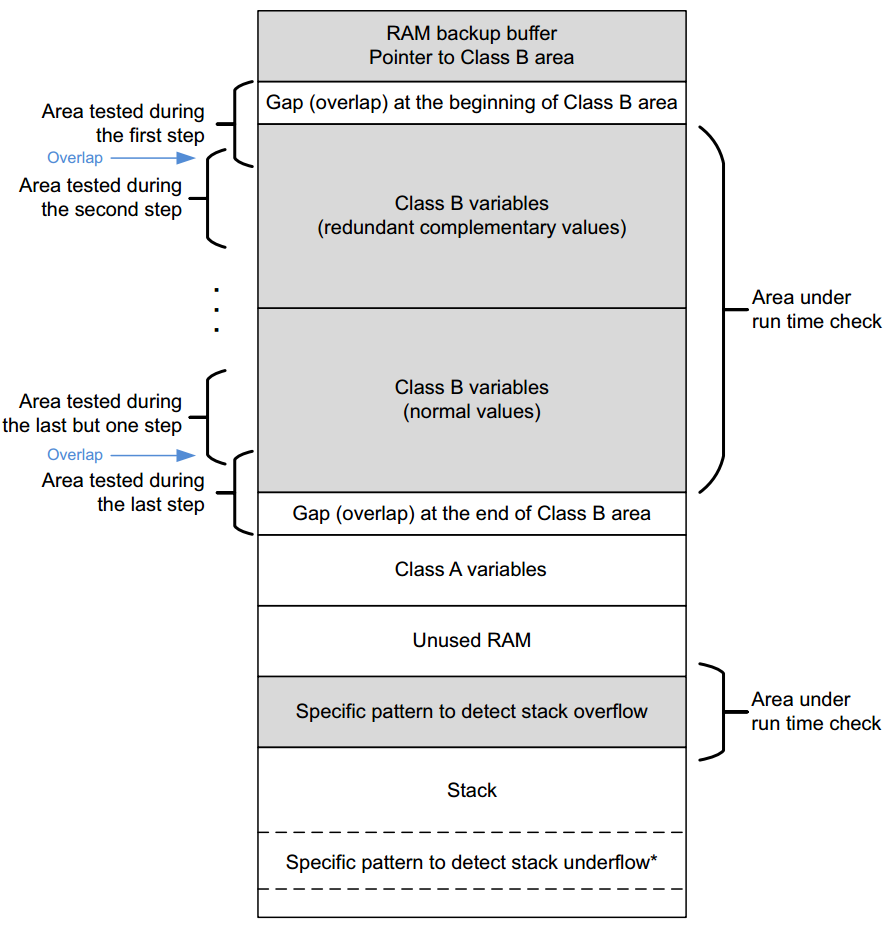
对于GPIO,Class B规定必须侦测到任何数据IO的失效.这一点可以配合其他应用部分进行检测,比如关掉一个制冷/加热开关,用模拟采样值来回来验证该开关是否失效.如果数字部分与模拟部分是相互独立供电,则此点需要额外小心两者电压上的差异.

对于中断与外部通信接口,可也通过设定一个软件计数器来检验中断与通信发生的次数,在通过经验检验过的时钟源来通过该计数器的结果来判定被测试的中断与通信接口是否正常工作.

对于定时器,可以通过某函数在指定的时间范围内运行的次数来验证.也可以通过不同的时钟源来交叉验证定时器.定时器与应用的耦合度属于较高的部分.

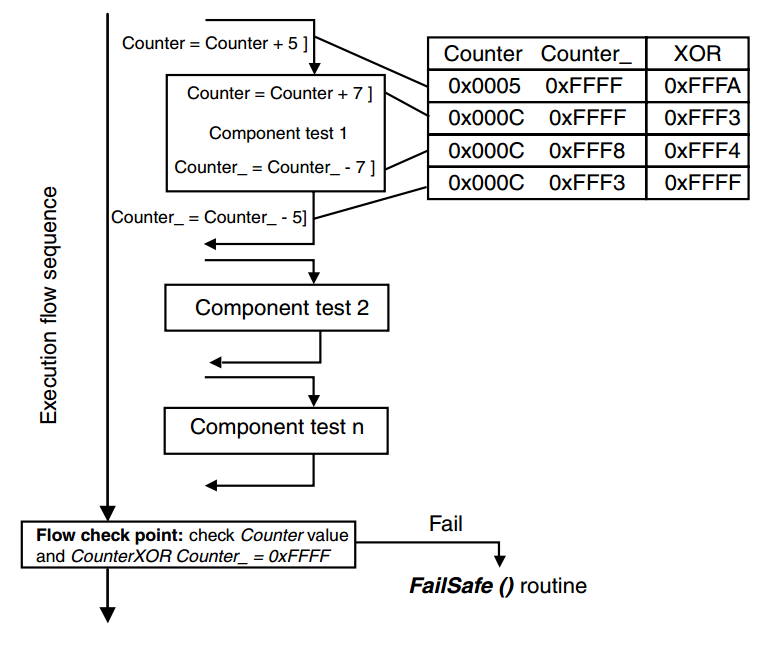
对于外部存储器寻址,大多数英飞凌的微控器型号不涉及到这点.涉及到的型号请参考RAM/Flash检测的方法.

以下是一种内存分配方法举例,关键变量以互补的形式在不同的RAM区间被保留了两份以互相校验.另外堆栈也做了特殊处理已检测Stack Overflow与under Flow的情况.



# 一种检测程序执行流程的方法

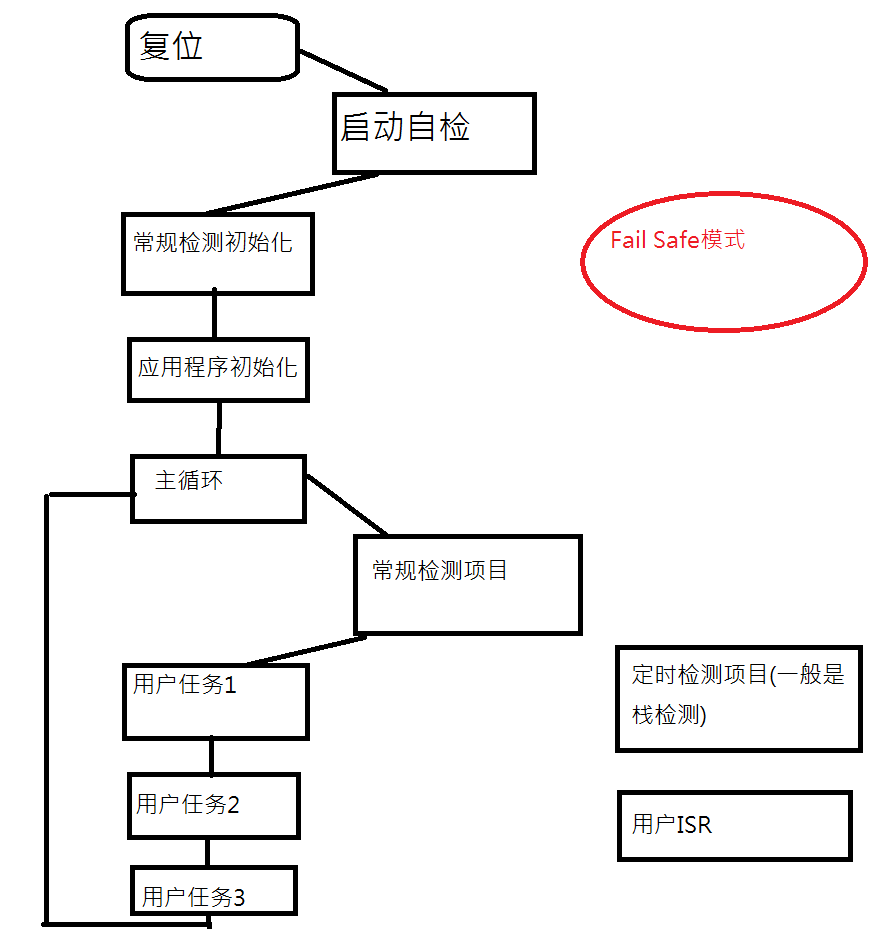
对于程序执行的各个阶段都分配一个不同的数字进行编码.这些变量以双互补形式进行冗余存储.当一个程序的阶段被执行,进行一种四步对称检测.前两步检测该程序阶段被正确调用(在调用该程序段与返回的地方执行).后两步检测该程序是否被正确执行(该程序段的入口与出口).



这种检测方法对CPU负载影响不大,因为每次检测只需操作互补变量对的其中一个.因为调用/返回与入口/出口总是配对出现的,故此互补变量对的互补关系总是被保持.之后安排数个检测点对非预料性结果进行检测,一旦发生非预料结果及跳入FailSafe模式.

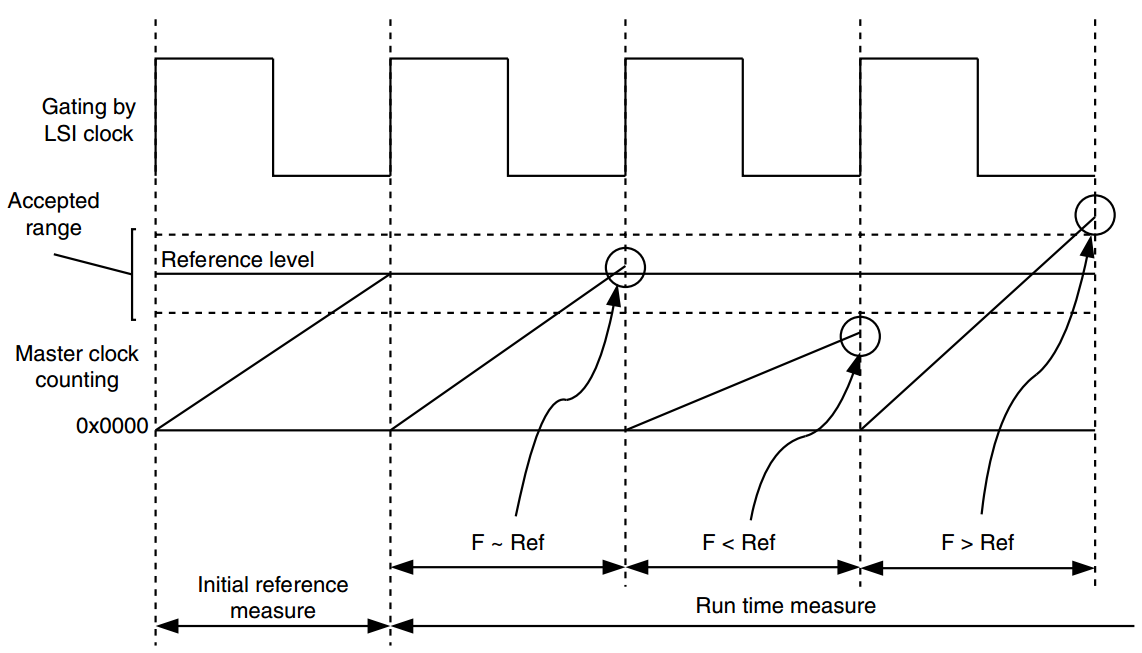
此例子中该程序阶段的关键数字是5,而该被执行的程序段为7.

集成之后的程序流程示例:



其中开机自检一般安排在c\_startup之前,因为c startup就假定所有的部件运行正常.

两个独立时钟互相校验:



一般是用高低速时钟分别做定时中断,高速时钟中断增加计数器,低速时钟ISR中校验高速时钟计数器是否为预料误差之中.

提高EMC性能的软件技巧

要提高EMC性能,软件硬化是很关键的一个步骤.

在软件设计阶段就考虑防范EMC干扰是非常重要的,也是成本最低的做法.一般而言要考虑的EMC干扰源:

• 微控制器不响应输入  
• 程序跑飞

• 执行未预料的指令  
• 野指针  
• 子程序执行错误  
• 寄生复位/中断  
• 单元配置错乱  
• I/O状态被错误配置

可能的后果:

• 产品产生非预料的响应  
• 丢失上下文状态  
• 分支错误  
• 中断丢失  
• 数据完整性丢失  
• 读入错误的输入状态

本文介绍两种措施:

1.预防性措施,可以在现有设计上添加这些措施以增强软件的健壮性

2.自动恢复措施,当检测到错误状态,进行记录(如有需要可以进行错误提示),Fail-Safe操作,之后进行尝试自动恢复到已知状态,理想状况是最终的用户感觉不到这个过程

# 预防性措施:使用看门狗或者类似的定时方法

看门狗的原理很简单,一旦开启了,那么防止其复位微控制器的惟一方法就是在其复位操作之前进行喂狗操作.

但是开启看门狗与喂狗操作的位置必须正确设计,否则看门狗起不到预防错误的目的.

下图是两种常见的错误看门狗使用方法:

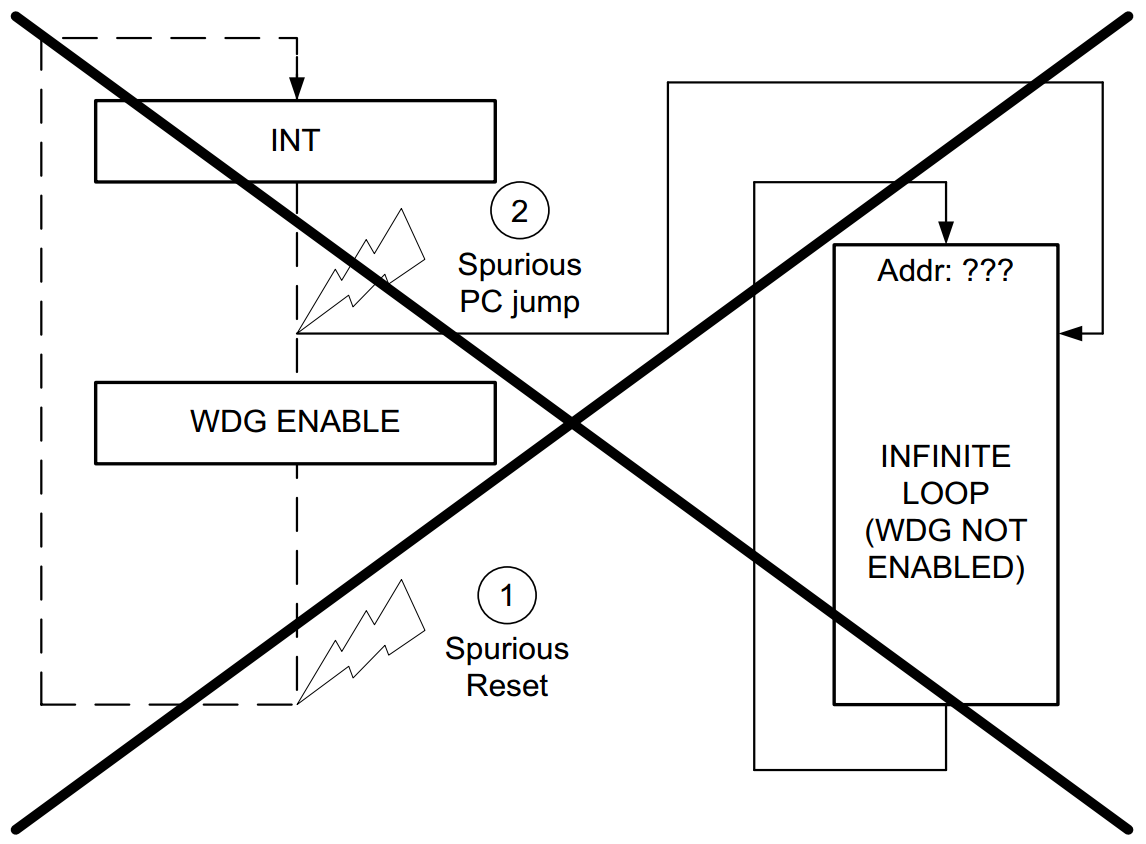


图 看门狗使能过晚,导致在初始化阶段程序就跑飞了

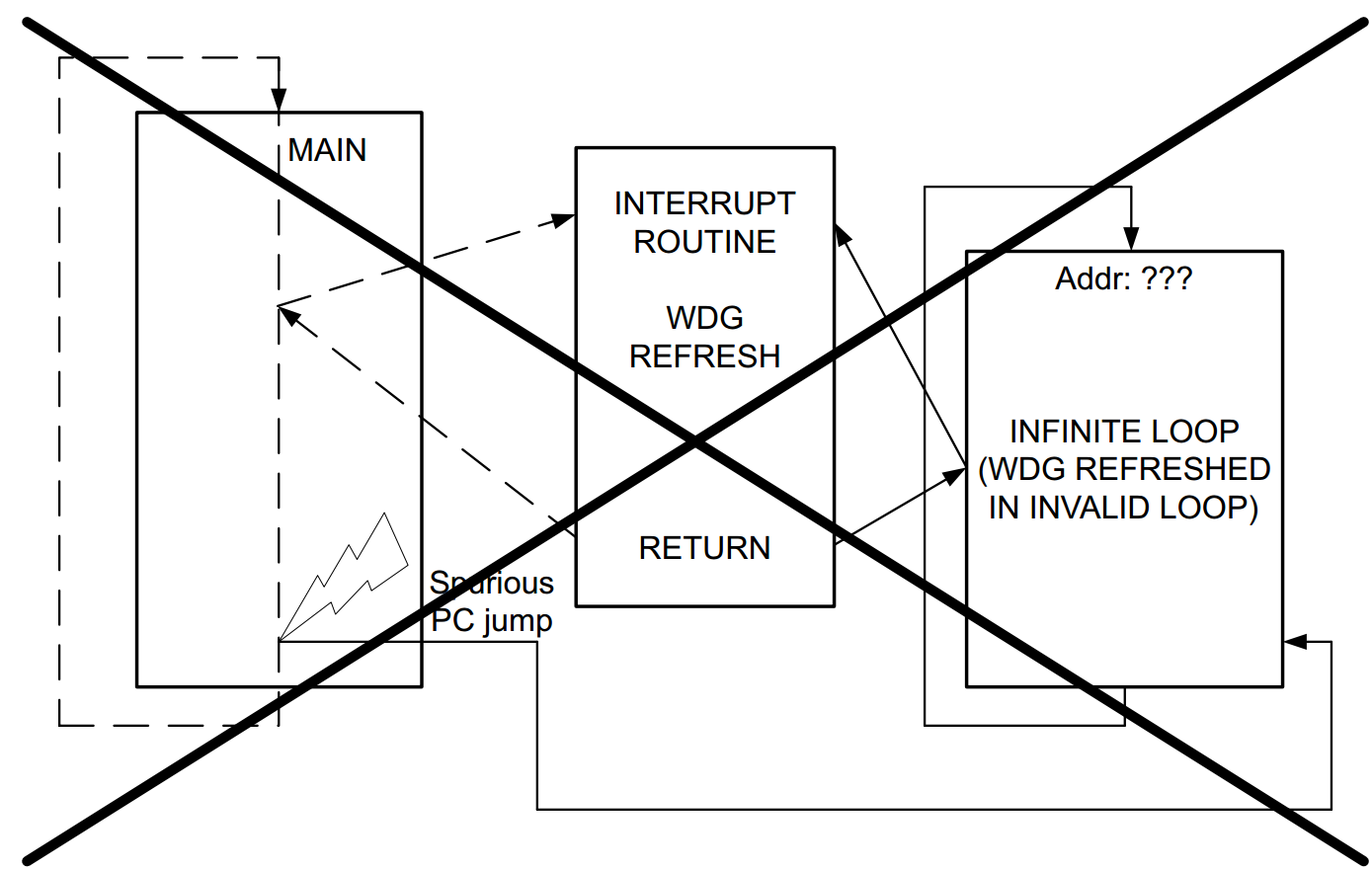


图 程序跑飞了,但是看门狗在中断中喂,结果是看门狗不能正确复位

针对以上两种经典错误,以下两点需注意:

1.看门狗越早使能越好,某些处理器如Aurix系列在复位之后看门狗就是自动开启的为理想状态

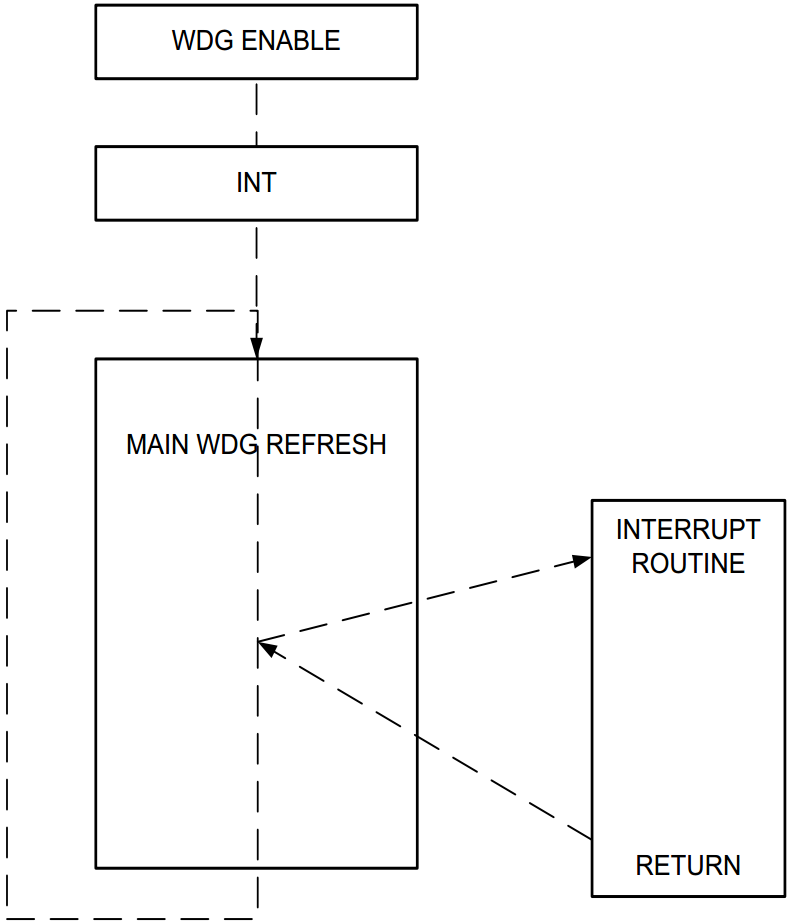
2.不要在中断或者未被超时机制保护的程序点进行喂狗操作

两次喂狗之间的间隔需要精心计算,中断也需要被考虑进去.

看门狗能起到的最小的作用就是能复位整个程序,这也意味着程序执行的上下文与数据完整性发生丢失.

所以在程序启动的时候,需要借助一些状态位来判断复位源.

正确的看门狗使用示意:



# 预防性措施: 对未用的程序空间进行处理

基本上所有的应用中,微控制器的代码空间是没有使用完全的.不管它的话,那么它的内容为杂乱的数据(新MCU第一次烧写程序的话,这些未使用的代码空间为0xff,但MCU产品大多会进行出厂测试,所以0xff也是不能保证的).保险的做法就是将这些未使用的空间填充为跳转至某已知地址的指令,这样保证程序即使跑飞了也能跳回到已知的位置.

对于Aurix处理器,可以填充为未知指令触发trap,或者直接填充非法指令以引发trap,还有MPU也可以应用起来触发trap.

# 预防性措施:冗余输入

读取状态的时候,多读取几次,尤其是外部信号,多读取几次以确认不是电磁干扰引起的错误触发.这个就是最便宜的输入滤波器的效果!

# 预防性措施:处理未使用的中断向量

对于tricore内核的处理器,中断向量表是由软件实现的.那么尽量不要使用多余的向量表项目.但是对于已经使用的向量与最大可能的向量之间也可以做类似处理.一般的做法就是设定一个默认的ISR.比如:

Default\_Handler PROC

EXPORT WWDG\_IRQHandler [WEAK]

EXPORT PVD\_VDDIO2\_IRQHandler [WEAK]

EXPORT RTC\_IRQHandler [WEAK]

EXPORT FLASH\_IRQHandler [WEAK]

EXPORT RCC\_CRS\_IRQHandler [WEAK]

EXPORT EXTI0\_1\_IRQHandler [WEAK]

EXPORT EXTI2\_3\_IRQHandler [WEAK]

EXPORT EXTI4\_15\_IRQHandler [WEAK]

EXPORT TSC\_IRQHandler [WEAK]

EXPORT DMA1\_Channel1\_IRQHandler [WEAK]

EXPORT DMA1\_Channel2\_3\_IRQHandler [WEAK]

EXPORT DMA1\_Channel4\_5\_6\_7\_IRQHandler [WEAK]

EXPORT ADC1\_COMP\_IRQHandler [WEAK]

EXPORT TIM1\_BRK\_UP\_TRG\_COM\_IRQHandler [WEAK]

EXPORT TIM1\_CC\_IRQHandler [WEAK]

EXPORT TIM2\_IRQHandler [WEAK]

EXPORT TIM3\_IRQHandler [WEAK]

EXPORT TIM6\_DAC\_IRQHandler [WEAK]

EXPORT TIM7\_IRQHandler [WEAK]

EXPORT TIM14\_IRQHandler [WEAK]

EXPORT TIM15\_IRQHandler [WEAK]

EXPORT TIM16\_IRQHandler [WEAK]

EXPORT TIM17\_IRQHandler [WEAK]

EXPORT I2C1\_IRQHandler [WEAK]

EXPORT I2C2\_IRQHandler [WEAK]

EXPORT SPI1\_IRQHandler [WEAK]

EXPORT SPI2\_IRQHandler [WEAK]

EXPORT USART1\_IRQHandler [WEAK]

EXPORT USART2\_IRQHandler [WEAK]

EXPORT USART3\_4\_IRQHandler [WEAK]

EXPORT CEC\_CAN\_IRQHandler [WEAK]

EXPORT USB\_IRQHandler [WEAK]

WWDG\_IRQHandler

PVD\_VDDIO2\_IRQHandler

RTC\_IRQHandler

FLASH\_IRQHandler

RCC\_CRS\_IRQHandler

EXTI0\_1\_IRQHandler

EXTI2\_3\_IRQHandler

EXTI4\_15\_IRQHandler

TSC\_IRQHandler

DMA1\_Channel1\_IRQHandler

DMA1\_Channel2\_3\_IRQHandler

DMA1\_Channel4\_5\_6\_7\_IRQHandler

ADC1\_COMP\_IRQHandler

TIM1\_BRK\_UP\_TRG\_COM\_IRQHandler

TIM1\_CC\_IRQHandler

TIM2\_IRQHandler

TIM3\_IRQHandler

TIM6\_DAC\_IRQHandler

TIM7\_IRQHandler

TIM14\_IRQHandler

TIM15\_IRQHandler

TIM16\_IRQHandler

TIM17\_IRQHandler

I2C1\_IRQHandler

I2C2\_IRQHandler

SPI1\_IRQHandler

SPI2\_IRQHandler

USART1\_IRQHandler

USART2\_IRQHandler

USART3\_4\_IRQHandler

CEC\_CAN\_IRQHandler

USB\_IRQHandler

B .

ENDP

# 预防性措施:移除关键的指令与非法的指令

## 关键指令

关键指令就是状态切换,比如正常模式进入低功耗模式的代码.可以通过观察.list文件来找到其位置.一般而言程序跑飞了之后,很有可能会读取到这段指令,可以在其前后加一些NOP指令让这个位置发生位移以降低读取到这段指令的概率.

## 非法指令

非法指令就是内核不认识的指令,对于Tricore或者大多数其他内核来讲执行这样的指令会发生Trap.处理方式如上.

# 预防性措施:对AD采样值进行处理

一般而言就是多采样几次取平均,也可以参照电视节目海选评分的做法去除最高分去除最低分再平均以防止读入受到干扰的值.

# 预防性措施:配置定期检查与重新配置

一般而言发生配置丢失,错乱的事故多发地在于时钟与IO的配置,因为这些地方的配置离外部电路的物理位置最近,最易受干扰.对这些配置采取定期检查并且重新配置可以有效提高EMC性能.

|  |  |  |  |
| --- | --- | --- | --- |
| 预防性技巧 | 优点 | 缺点 | 实施要点 |
| 看门狗 | 与CPU相互独立,较少受CPU影响 | 如果有低功耗模式,则看门狗要格外留意 | 实施简单但要精心放置看门狗初始化与喂狗操作 |
| 未使用代码空间插入复位 | 快速从跑飞状态跳出 | 丢失了执行的上下文 | 注意读取复位源 |
| 未使用代码空间插入跳转或Trap | 快速从跑飞状态跳出 | 无 | 注意精心设计相应的Trap以使恢复更加彻底 |
| AD采样平均 | 提升了采样的精度与抗干扰性 | 要多采样几次且需要多余的运算 | 注意选取有效值的技巧 |
| 移除关键与非法指令 | 有效防范误执行相应的指令 | 对真正要使用此指令的情况有干扰 | 观察.list文件 |
| 冗余读取输入 | 提升了读取的稳定性 | 要多读取几次还要判断哪种是错误的需要软件决策时间 | 注意选择决策的判据 |
| 处理未使用的中断源 | 防止向量错误执行 | 无 | 看上文的参考代码 |
| 定期检测与重新恢复重要配置 | 防止重要配置丢失 | 需要软件工夫 | 只需保护易于丢失的时钟与IO配置,其他配置可以无需此种考虑 |

表 技巧优缺点一览表

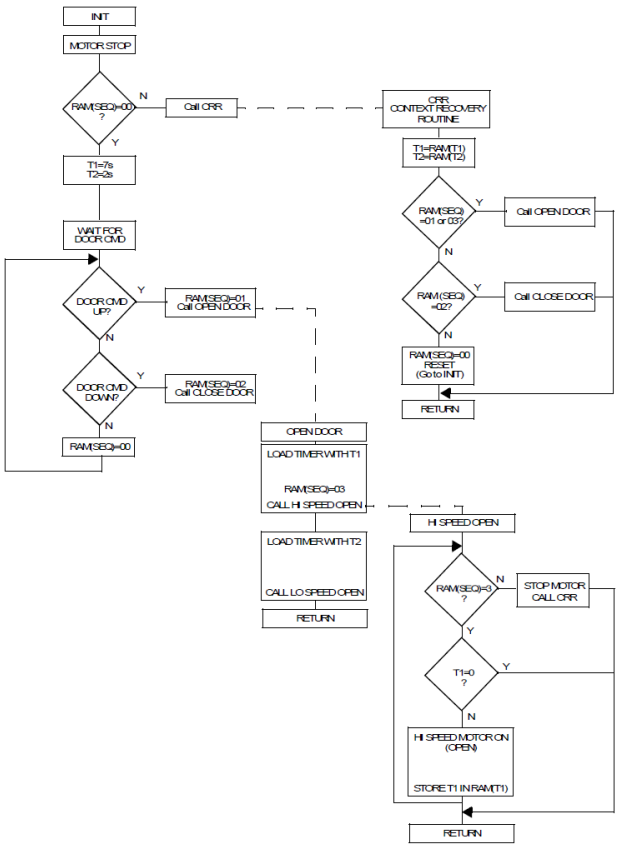
# 预防性措施:冗余数据存储与交换

只要是存储在片内的数据,那么一定会受到EMC的影响.常见的保护方法包括双冗余互补存储,ECC校验.值的一题的是某些产品硬件上就支持某些冗余与校验,比如Aurix产品的存储器都有ECC校验的硬件特性,可以利用起来.

以上是预防性措施,下面介绍自恢复技巧.在工程实践中,非预料性复位,PC跑飞与寄生中断是三种最为常见的EMC造成的结果.以下的技巧也针对此几种结果而试图自恢复.值的注意的是,这几种情况中,RAM内容,Flash/EEPROM的内容一般而言都是完整的(断电了就另当别论),所以自恢复要将这特点给利用起来.

# 自恢复技巧: RAM中保持程序运行的上下文状态

下图是一个在RAM中保持当前执行状态的例子,要点是每次输出状态改了就在RAM中保持其状态,如果有PWM输出之类的配置,也加以保持,这样在发生看门狗或者其他的复位之后,程序可以读取复位源+当前的执行状态自动恢复到跑飞之前的状态.

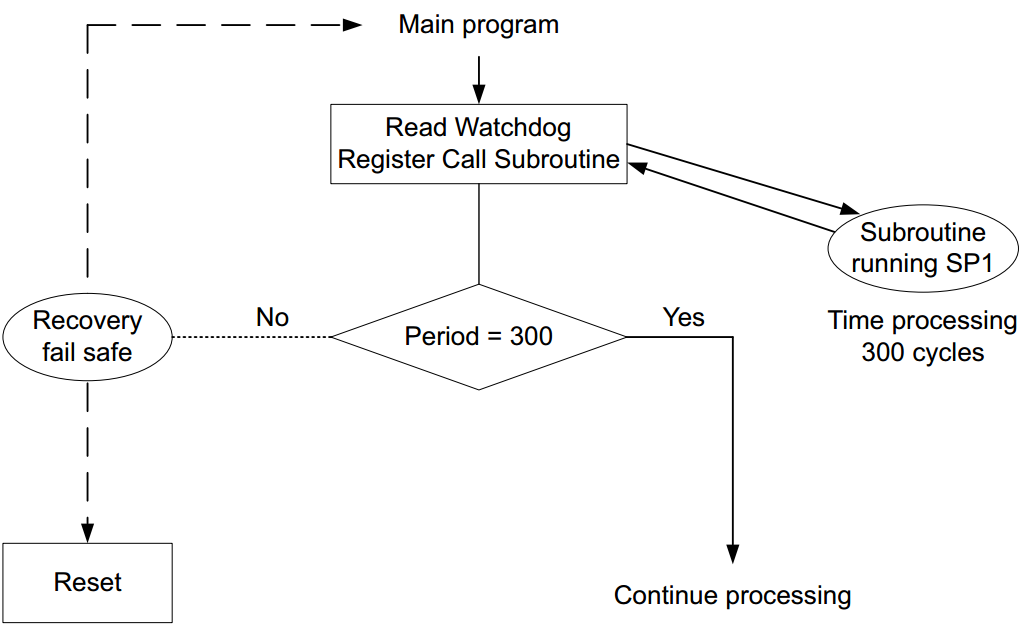


# 自恢复技巧:把看门狗用精确一点

大多数程序员使用看门狗的习惯就是:能喂狗的时候,喂到最饱,在喂狗点将看门狗的值恢复到最大可允许的数字以最大限度减少遗漏喂狗的风险.

这样做显然不是正确的喂狗方法.

正确的做法是精确掐好看门狗的计数器的值,某个函数运行时间过长的话,果断采取相应措施以自恢复.



# 自恢复技巧:使用复位标志来确定复位源

复位有很多种,上电复位,看门狗复位,电压监测复位,复位引脚复位等等.精确判断是哪一种复位,之后采取正确的相应措施能够恢复某些EMC引起的干扰.

# 自恢复技巧:将程序执行的上下文保存到NV存储器

这是一步更绝的技巧,因为编程NV存储器的时间比RAM写入要花的长得长的时间.以Aurix产品为例,时间大概在数十ms左右.而在这期间很可能又发生了EMC干扰导致复位.所以要使用此技巧需要考虑多种因素,包括在多个存储区间进行冗余保持,添加特殊的标记/CRC等等.而且这种技巧只适合于保持变化并非很频繁的上下文.频繁发生变化的上下文还是RAM保持为妙.另外现在新工艺日新月异发展,有很多新的工艺的存储器,比如铁电存储器有NV存储器读写都较快,还有自带电池的RAM都是设计者值得考虑的选项.

# 后记

英飞凌的半导体产品从硬件上来讲,经过了严格的验证与测试,且通过了多项业界认可的标准,比如AEC-Q100等等.但是这并非意味着软件预防与恢复技巧是杞人忧天,庸人自扰.任何汽车电子的设计工程师应该很严肃地对待产品的安全性.预防性措施可以提高软件的健壮性,自恢复技巧能够在受到严重干扰的情况下进行自救.考虑了这些情况的产品将与未考虑这些情况的产品拥有非常显著的差别.