TriCore™ AURIX™ 家族系列

32 位

CPU 和安全看门狗

AP32221

应用笔记

V1.0 2014-5

微控器

免责条款

为方便客户浏览，英飞凌以下所提供的将是有关英飞凌产品及服务资料的中文翻译版本。该中文翻译版本仅供参考，并不可作为任何论点之依据。虽然我们尽力提供与英文版本含义一样清楚的中文翻译版本，但因语言翻译和转换过程中的差异，可能存在不尽相同之处。因此，我们同时提供该中文翻译版本的英文版本供您阅读，请参见 www.infineon-ecosystem.org。并且，我们在此提醒客户，针对同样的英飞凌产品及服务，我们提供更加丰富和详细的英文资料可供客户参考使用。请详见 www.infineon.com

客户理解并且同意，英飞凌毋须为任何人士由于其在翻译原来的英文版本成为该等中文翻译版本的过程中可能存在的任何不完整或者不准确而产生的全部或者部分、任何直接或者间接损失或损害负责。英飞凌对于中文翻译版本之完整与正确性不担负任何责任。英文版本与中文翻译版本之间若有任何歧异，以英文版本为准，且仅认可英文版本为正式文件。

您如果使用以下提供的资料，则说明您同意并将遵循上述说明。如果您不同意上述说明，请不要使用本资料。

版本 2014/05

出版发行：
英飞凌科技公司
上海，中国

免责条款

本应用笔记中给出的信息仅作为实现英飞凌器件的建议，不得被视为英飞凌器件的任何特定功能、条件或质量作出的任何说明或保证。此应用笔记的接受者必须在实际应用中判定此种描述的任何功能。英飞凌科技在此否认承担此应用笔记中任何和所有信息相关的任何形式的保证和责任（包括但不限于不侵犯第三方知识产权）。

信息

有关技术、交货条款及条件和价格，请与您最近的 Infineon Technologies 办事处联系。

警告

由于技术要求，组件可能含有危险物质。如需相关型号的信息，请与您最近的 Infineon Technologies 办事处联系。如果可能合理地预期此类组件的故障会导致生命支持器件或系统发生故障或影响该器件或系统的安全性或有效性，则 Infineon Technologies 提供的组件仅可用于获得 Infineon Technologies 明确书面批准的生命支持器件或系统。生命支持器件或系统的目的是植入人体或支持和/或保持并维持和/或保护生命。如果出现故障，则可能危及使用者或他人的人身安全。

文献修订史			
日期	版次	修订人	修订内容
2012/10/15	V1.0	Tomislav Garaca	初版

请留下您的宝贵建议

您是否认为本文档中的任何信息存在错误，含糊不清或遗漏？您的宝贵意见和建议将帮助我们持续不断地改进文档质量。请将您的建议（请注明文档的索引号）发送电子邮件至：

ctdd@infineon.com

目录

1 介绍

该应用笔记描述了英飞凌 32 位微控器 AURIX™家族器件(尤其关注在 TC27x 器件)内置的 CPU 和功能安全看门狗(WDTs)。

1.1 对读者说

此文件为应用软件工程师撰写英飞凌微控器 AURIX™家族系列而编制。

1.2 范围和目的

该文件涵盖了实现和使用 WDT 看门狗功能所需的 CPU 和功能安全看门狗的硬件和软件架构，以便在运行 AURIX™家族微控器时，获取一种安全而又可靠的方法检测软件和硬件故障，并从这些故障中恢复。

1.3 缩略词

本文档使用了以下缩略词：

ACCEN	使能访问
ENDINIT	初始化结束
LFSR	线性反馈移位寄存器
NMI	非可屏蔽中断
SCU	系统控制单元
SMU	安全管理单元
SPB	系统外围总线
WDT	看门狗定时器
WDTx	看门狗定时器 x (x = S, CPU0, CPU1, CPU2)

1.4 参考文献

其它参考文档:

[1]AURIX™ TC27x User Manual V1.1 2012-07, Infineon Technologies (or later)

2 看门狗定时器(WDTs)

该章节描述了英飞凌 AURIX™ TC27x 看门狗定时器(WDTs)。所包含的主题有 WDT 功能概述，功能安全和 CPU 看门狗简介，主要 WDT 特性总述，以及 WDT 寄存器简要描述。

2.1 概况

有下列两种类型的 WDT 看门狗定时器:

•功能安全看门狗

-中心资源，监测与安全相关的功能

•CPU 看门狗

-位于每个 CPU 中，去监视软件的流程

TC27x 包含了下列看门狗定时器 WDT：

•功能安全看门狗

•CPU0 看门狗

•CPU1 看门狗

•CPU2 看门狗

WDT 提供了一种安全的方式来检测软件和硬件故障，并从这些故障中恢复过来。这些看门狗在用户定义的时间周期内帮助终止故障的 CPU 或基于 TC27x 的系统。

此外，每一个看门狗定时器都包含了一个初始化截止保护(ENDINIT) 特性，该特性保护关键的 TC27x 寄存器远离故障和未经授权的写访问。

为了确保 WDT 看门狗定时器正常工作,并正确地修改 ENDINIT 位，在此应用了一个访问错误检测机制以防止对 WDT 控制寄存器突发且不正确的写访问。以无效的密码（在首次访问期间）或者是保护位的不正确的值（在二次访问期间）对 WDT 控制寄存器进行的任何写访问会触发一个向安全管理单元（SMU)发送的看门狗报警请求。此外，在一个有效的写访问清除了 ENDINIT 位之后，看门狗为这个访问窗口强加了一个时间限制。如果在该时间限制内，ENDINIT 位未被再次置位，系统可能会出现故障，并且看门狗报警被报告给 SMU。

也可以获取配置选项使能看门狗服务，以检查代码执行序列并监控一个中级代码执行时间。如果使能了这些检查，那么任何不正确的执行序列或超出限制的执行时间将触发一个 SMU 报警请求。

任何 WDT 看门狗过期（定时器溢出）产生一个 SMU 报警请求。可以将 SMU 配置为生成一个中断或非可屏蔽中断（NMI)以便在采取进一步行动之前（如复位，或者是使 CPU 进入空闲状态），提供一些时间以便恢复或进行状态记录。

安全看门狗

安全看门狗保护关键系统寄存器远离意外及未经授权的写访问,并提供一个独立于任何 CPU 看门狗的总系统看门狗。当被使能时，如果在用户可编程的时间周期内未正确喂狗，安全看门狗可以触发一条 SMU 报警请求。因此定期喂狗是确保系统正确运行的保障。

典型情况下，配置 SCU 写保护（ACCEN)以便仅仅只有“安全”CPU 可以配置安全看门狗并喂狗。此外有关安全看门狗配置的使能和停用功能都需要一个安全的 ENDINIT 密码。

CPU 看门狗

单独的 CPU 看门狗可以在不需要软件协调公共看门狗的共享时，就能监控单独的 CPU 执行线程。每个 CPU WDT 都包含了一个 ENDINIT 特性，该特性保护还未被安全看门狗的 ENDINIT 保护的重要局部 CPU 寄存器和一些系统寄存器远离意外和未经授权的写访问。

当被使能时，如果在用户定义的时间周期内未喂狗， CPUx WDT 可以触发一条 SMU 报警请求。因此定期喂狗是确保正确执行 CPU 软件序列的保障。

复位之后，CPU0 处于 RUN 模式且 CPU0 WDT 自动启动。其它 CPU 起初在 HALT 状态，因此与它们相关的 WDT 看门狗定时器都被停用。CPU 看门狗仅可被与它相关的 CPU 配置，使能或停用。

2.2 看门狗定时器特性

每个看门狗都具备以下特性:

•16 位看门狗计数器

•可选择的输入时钟频率

- $f_{SPB}/64$
- $f_{SPB}/256$
- $f_{SPB}/16384$

•16 位可用户定义的重载值,用于正常看门狗操作，以及固定重载值(0xFFFC)，用于超时模式

•包含了相应的 ENDINIT 位和用以监控修改

•高级密码访问机制，拥有固定和用户可定义的密码字段

•访问错误检查

-无效密码(首次访问期间)或无效保护位(第二次访问期间)触发一条向 SMU（安全管理单元）发送的报警请求。

•可选的代码序列检查

-一个不正确的代码序列识别触发一条向 SMU 发送的报警请求

•可选的代码执行时间检查

-代码执行时间超限触发一条向 SMU 发送的报警请求

•溢出错误侦查

-WDT 计数器的溢出触发一条向 SMU 发送的报警请求

•看门狗功能可以被停用

-但访问保护和 ENDINIT 功能保持使能

•在接收到一条未喂狗安全警告报警之后，阻止看门狗重载的可配置机制，确保这些未服务的安全报警引起 SMU 硬件响应。

•外设“心跳”显示用于显示 WDT 在运行且 SMU 不处于报警模式

2.3 看门狗定时器寄存器

本章节简要地描述了 WDT 寄存器。

注释：有关 WDT 寄存器的详细描述，请参考 AURIX™ TC27x 用户手册[1].

WDT 控制寄存器 0：WDTxCON0

该寄存器包含：

•为相应的 WDT 保持当前密码和定时器重载值。

•控制访问 ENDNIT 保护寄存器的 ENDINIT 位。

•用于控制访问 WDTxCON0 寄存器自身的权限的锁定位（LCK)的当前值。LCK 值受硬件控制。

WDT 控制寄存器 1：WDTxCON1

WDTxCON1 寄存器管理每个 WDT 看门狗的运行，并提供下列控制和配置特性：

•输入频率请求控制位
  -决定看门狗定时器的当前输入频率
•停用请求控制位
  -使能或停用 WDT
•解锁限制请求控制位
  -在检测到一条未喂狗安全报警警报之后，提供一个机制阻止看门狗重载
•密码自动测序请求位
  -在每次对 WDTxCON0 寄存器修改访问或检查访问之后，使能或停用密码的自动更改
•配置位
  -在对 WDTxCON0.REL 字段检查访问或修改访问期间，用于检查时间戳计数。

注释：安全看门狗 WDTSCON1 寄存器有一个额外的位 CLRIRF，该位可被用于清除检测双 SMU（如 WDT）复位的内部复位状态。

每一个 WDTxCON1 寄存器都受到相应的 WDTxCON0.ENDINIT 位的保护。

WDT 状态寄存器:WDTxSR

WDTxSR 寄存器显示了每个 WDT 的当前状态。状态寄存器包含的位显示了：

•访问故障状态标志

•看门狗溢出故障状态标志

•看门狗输入时钟状态

•看门狗使能或停用状态标志

•看门狗超时模式标志

•SMU 解锁限制状态标志

•密码自动测序状态标志

•时间戳检查状态标志

•定时器值（WDT 当前内容）

3 ENDINIT 功能

3.1 概况

TC27x 使用一系列 ENDINIT 特性来保护系统关键寄存器远离意外和未经授权的写访问。

ENDINIT 位被并入到每个 WDT 控制寄存器。只有在下列情况时，才能写受 ENDINIT 特性保护的寄存器：

•在相应 WDTx 控制寄存器 0 (WDTxCON0) 里的 ENDINIT 位被清 0。

•超级管理员模式有效。

如果未满足这些条件，则忽略写寄存器的企图，且寄存器里的内容不会被修改。

为了建立最高级别的安全机制，对 ENDINIT 位进行修改受到 WDT 里执行的安全访问保护机制的保护。此外，一旦软件使能对系统关键寄存器的访问，每个 WDT 看门狗定时器通过启动超时序列监控 ENDINIT 位的修改，将 ENDINIT 位清 0。如果超时周期在相应的 ENDINIT 位再次置位之前过期，软件流可能会出现故障且报警请求被发送到 SMU。

注释： 需花费一定的时间清除 ENDINIT 位。为了确保已经真正地使能了对 ENDINIT 保护寄存器的访问，在 ENDINIT 位被清 0 后，首次访问 ENDINIT 保护寄存器首之前，应该先读取 ENDINIT 位。

3.2 对 WDTxCON0 寄存器的密码访问

必须将一个密码写入到 WDTxCON0 寄存器，以便解锁该寄存器而对其进行修改。软件必须事先知道正确的密码，或者是在运行期间将密码计算出来。

功能安全看门狗密码寄存器 WDTSCON0 受到通用 SCU 保护机制的保护，该保护机制仅赋予配置的主核写访问权限（请参考 AURIX™ TC27x 用户手册 [1]）SCU 访问限制寄存器里的 ACCEN0）。

CPU 特定的看门狗密码寄存器 WDTCPUyCON0 受到单独的保护，因此他们仅能被相应的 CPUy 访问。

为了确保 CPU 故障未被忽略，在 SMU 不运行时提供了一种选择阻止看门狗被解锁。该方案通过将位 WDTxCON1.UR 置 1 而使能。

如果密码有效，且 SMU 的状态符合 WDTxSR.US 位的要求，那么一旦密码访问完成，WDTxCON0 立即被解锁。解锁状态通过 WDTxCON0.LCK = 0 得以显示。为了确保正确的服务序列，只有当 WDTxCON0.LCK 位在访问前被置位，密码访问才得到许可。

当 WDTxCON0 被解锁，WDT 看门狗被自动切换到超时模式。只有 ENDINIT 位在对 WDTxCON0 有效修改访问里被置位之后，超时模式才被终止。如果超时周期在相应的 ENDINIT 位再次置位之前过期，软件流可能会出现故障且报警请求被发送到 SMU。

如果在密码访问期间，将一个无效密码值写入到 WDTxCON0，那么就触发了看门狗访问错误状态。此时，位 WDTxSR.AE 被置位，并向 SMU 发送一条报警请求。

表 1 总结了密码访问位模式的要求。表中有多种密码访问选项，并在以下子章节对这些密码访问进行了更加详细地描述。

注意：　从 WDTxCON0.PW 寄存器读取密码值，返回位字段 WDTxCON0.PW[7:2]的倒置值(翻转的)。这一点确保一个简单的读写并不足以喂狗或者是对 WDT 解锁以便修改。

表 1 密码访问位模式要求

位位置	要求值
[1:0]	Fixed; must be written to 01_B
[15:2]	If WDTxSR.PAS = 0: • WDTxCON0.PW[7:2] must be written with inverted current value read from WDTxCON0.PW[7:2] • WDTxCON0.PW[15:8] must be written with non-inverted current value read from WDTxCON0.PW[15:8] The default password after application reset is 00000000111100_B If WDTxSR.PAS = 1: • Must be written with Expected Next Sequence Password
[31:16]	If WDTxSR.TCS = 0: • Must be written with current value of user-definable reload value, WDTxCON0.REL The default value after application reset is $0xFFFC_H$ If WDTxSR.TCS = 1: • Must be written with inverted estimate of the WDT counter value, WDTxSR.TIM. This value must be within +/- WDTxSR.TCT of the actual value

3.2.1 静态密码

在静态密码模式下(WDTxSR.PAS = 0)，只有有效的修改访问才能更改密码。对密码访问的的设计使得不可能很简单地就读取并重写寄存器。从 WDTxCON0.PW[7:2] 所读取的密码位在被重写之前，就必须被取反（翻转）。这可以防止通过简单的读写序列就意外解锁 WDT 所造成的故障。

3.2.2 自动密码序列

如果使能自动密码序列(WDTxSR.PAS = 1)，那么在每次密码检查之后，（如密码访问或检查访问），密码就自动更新。预期的下一个密码遵循基于 14 位 Fibonacci LFSR（线性反馈移位寄存器）的伪随机序列，其特征多项式为 x14+x13+x12+x2+1。也可以通过 Modify Accesses 提供初始密码（或之后通过手动更新密码）。

图 1 显示了密码序列 LFSR，其中 PW[15:2]代表在 WDTxCON0.PW[15:2]里给定的当前密码位。

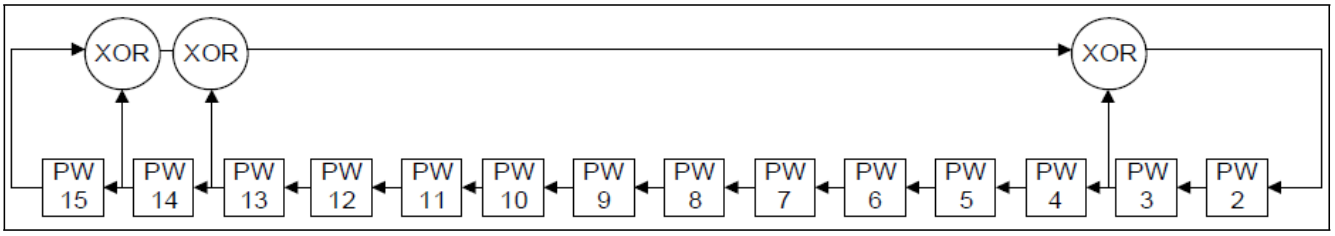

图 1　密码序列 LFSR

3.2.3 与时间无关的密码

如果不使能时间检查(WDTxSR.TCS = 0),那么在密码访问期间，必须使用已有的重载值重写 WDTxCON0 寄存器的 REL 字段（WDT 的重载值）。

3.2.4 时间检查的密码

如果使能时间检查 (WDTxSR.TCS = 1)，必须将当前 WDT 计数值 (WDTxSR.TIM)倒置（位翻转）写入 WDTxCON0 寄存器的 REL 字段。可接受的该估值(在 WDT 时钟周期内）误差大小由 WDTxSR.TCT 值（定时器检查公差）规定。如果写入的估值在 WDTxSR.TIM+/- WDTxSR.TCT 范围之外，则会指示 SMU 报警状况。该机制可以从上一次 WDT 启动之后，检查所消耗的程序执行时间。

注释： 当 WDT 在超时模式下运行时（如访问受 ENDINIT 保护的寄存器之后），时间检查比较仍然需要一个密码或检查访问。

3.3 对 WDTxCON0 寄存器的检查访问

检查访问用于 WDT 服务的中间点。可将该检查访问（如结合时间戳计数检查特性或序列密码）用于任务序列或执行时间监控。

除了锁定位 WDTxCON0.LCK 未被清 0，因而不允许之后的修改访问之外，其余检查访问等同于密码访问。如果在检查访问期间将一个不合理的密码值写入到 WDTxCON0，那么便发生看门狗访问错误状况。此时，位 WDTxSR.AE 被置位，并向 SMU 发送一条报警请求。只有当 WDTxCON0.LCK 位在访问前被置位，检查访问才得到许可。

表 2 总结了检查访问位模式要求。

表 2 检查访问位模式要求

位位置	要求值
[1:0]	Fixed; must be written to 11_B
[15:2]	If WDTxSR.PAS = 0: • WDTxCON0.PW[7:2] must be written with inverted current value read from WDTxCON0.PW[7:2] • WDTxCON0.PW[15:8] must be written with non-inverted current value read from WDTxCON0.PW[15:8] If WDTxSR.PAS = 1: • Must be written with Expected Next Sequence Password
[31:16]	If WDTxSR.TCS = 0: • Must be written with current value of user-definable reload value, WDTxCON0.REL If WDTxSR.TCS = 1: • Must be written with inverted estimate of the WDT counter value, WDTxSR.TIM. This value must be within +/- WDTxSR.TCT of the actual value

3.4 对 WDTxCON0 寄存器的修改访问

如果 WDTxCON0 寄存器成功地被密码访问解锁，那么它就可以被写访问修改了。然而，如果该访问要被接受，并被视为有效，它也必须达到一定的要求。

表 3 总结了修改访问位模式的要求。如果未达到规定的要求：

•检测到看门狗访问故障

•位 WDTxSR.AE 置位

•向 SMU 发送一条报警请求

表 3 修改访问位模式要求

位位置	要求值
0	User-definable; desired value for bit WDTxCON0.ENDINIT
1	Fixed; must be written to 1_B
[15:2]	User-definable; desired value of user-definable password field, WDTxCON0.PW The default password after application reset is 00000000111100_B
[31:16]	User-definable; desired value of user-definable WDT reload value, WDTxCON0.REL The default value after application reset is $0xFFFC_H$

在完成修改访问之后,锁定位(WDTxCON0.LCK) 再次置位,自动重新锁定 WDTxCON0。在寄存器被再次修改之前，必须再次执行一次有效密码访问。

外部 WDT 心跳指示

每个 WDTxCON0.LCK 位的当前状态也可被选择性地发送到外部引脚。因为只有在 WDT 服务期间（如在密码访问和修改访问之间），LCK 位低，因此就提供了一个周期性的“心跳”脉冲，该脉冲向外设监控器指示 WDT 仍然在运行。如果也使能了解锁限制位(WDTxCON1.UR = 1)，那么任何其它 SMU 安全报警状态都会阻止 WDT 运行并因此而停止心跳。

在 P20.9 上有 WDTSCON0.LCK 的状态显示 WDTSCLCK

在 P20.8 上有 WDTCPU0CON0.LCK 的状态显示 WDT0LCK

在 P20.8 上有 WDTCPU1CON0.LCK 的状态显示 WDT1LCK

在 P20.6 上有 WDTCPU2CON0.LCK 的状态显示 WDT2LCK

3.5 访问受 ENDINIT 保护的寄存器

只有在下列条件下，才能在运行时写受 ENDINIT 保护的寄存器：

•ENDINIT 位在相应的 WDTx 控制寄存器 0(WDTxCON0)里被清 0。

•超级管理员模式有效。

通过以下两步清除 ENDINIT 位：

•必须通过有效的密码访问解锁 WDTxCON0 寄存器以进行修改（见表 1）。

•必须通过有效的修改访问清除 ENDINIT 位 (WDTxCON0.ENDINIT = 0)。

-如表 3 的描述，必须满足修改访问位模式的要求。

现在在一个限定的时间周期内，开放了受 ENDINIT 保护的寄存器写访问，且 WDTxCON0 再次被锁定(WDTxCON0.LCK = 1)。

然而，当 WDTxCON0 被解锁时，WDT 看门狗被自动切换到超时模式。因此访问窗口是有时限的。超时模式只在 ENDINIT 位被再次置位之后才被终止，在 WDTxCON0 上需要另外一个密码和修改访问操作序列。如果超时时间在相应的 ENDINIT 位再次置位之前过期，软件可能会出现故障且报警请求被发送到 SMU。

通过以下两步设置 ENDINIT 位：

•必须通过有效的密码访问解锁 WDTxCON0 寄存器以进行修改（见表 1）。

•必须通过有效的修改访问设置 ENDINIT 位 (WDTxCON0.ENDINIT = 1)。

-如表 3 的描述，必须满足修改访问位模式的要求。

如果在应用中未使用 WDT 看门狗，而因此将其停用(WDTxSR.DS = 1)，ENDINIT 特性仍然有效。因此必须按照上述描述的步骤使能对受 ENDINIT 保护的寄存器写访问。

注释： 对于调试的支持，Cerberus 模块可以覆盖所有 WDT 的 ENDINIT 控制，以缓解调试流。如果位 CBS_OSTATE.ENIDIS 置位，无论当前 WDT 配置的状态如何，所有的 ENDINIT 保护都被停用。如果 CBS_OSTATE.ENIDIS 被清 0，那么整个控制都在 WDT 看门狗内。

4 定时器操作

4.1 概况

复位之后,用于功能安全和 CPU0 的看门狗定时器默认有效。其它 CPU 在最开始都处于 HALT 状态。因此他们相应的 WDT 都被停用。如果未在一个用户定义的溢出时间内（检测到定时器溢出）喂狗，那么该有效 WDT 看门狗定时器可以触发一条 SMU 报警请求。因此为了能够正确运行系统，需要定期喂狗（有效的 WDT）。

所有的 WDT 都使用 SPB 时钟 f_{SPB}。位于每个 WDT 之前的时钟分频器都提供 3 个计数器频率：

- $f_{SPB}/64$
- $f_{SPB}/256$
- $f_{SPB}/16384$

计算看门狗溢出时间的通用公式如下:

$$\text{period} = ((2^{16} - \text{start value}) * \text{divider}) / f_{SPB}$$

参数起始值可以是在 WDT 超时模式下计算溢出时间的固定值 $0xFFFC_H$,或者在 WDT 正常模式下计算溢出时间而在 WDTxCON0.REL 里给定的用户可编程 16 位重载值。

参数分频器代表在 WDTxCON1 寄存器里由输入频率请求控制位 IR0 和 IR1 所选择的用户可编程源时钟分频器。如表 4 所示，参数分频器可以有 64, 256 或 16384 几个分频值。

参数 f_{SPB} 代表 SPB 时钟频率。

注释: 如果 SPB 时钟频率 f_{SPB}是以 MHz 的单位给定，那么看门狗超时周期的计算则以 μs 为单位。

表 4 在 WDTxCON1 寄存器里的源时钟分频器

输入频率请求控制	IR1 [5]	IR0 [2]
Request to set input frequency to $f_{SPB}/16384$	0	0
Request to set input frequency to $f_{SPB}/256$	0	1
Request to set input frequency to $f_{SPB}/64$	1	0
Reserved, do not use.	1	1

4.2 定时器模式

每个 WDT 都可以在下列 3 种不同的模式下操作:

•超时模式

•正常模式

•停用模式

以下章节描述了这些模式,以及一个 WDT 如何从一个操作模式转换到另外一个操作模式。

4.2.1 超时模式

可在以下两种情况之后进入超时模式:

•在应用复位之后(除了 CPU1 和 CPU2 以外,这两个 CPU 最开始就处于 HALT 状态,因此与它们相关的 WDT 被停用。)

•在对 WDTxCON0 寄存器进行有效的密码访问之后

位 WDTxSR.TO = 1 时,显示进入超时模式,该位在进入超时模式时,由硬件自动置位。
定时器被设置为 $FFFC_H$,且开始向上计数。

只有使用正确的访问序列将 WDTxCON0.ENDINIT 位置 1,才可能正确地退出超时模式。如果执行了一个无效的 WDT 访问,或者是如果定时器在 ENDINIT 置位之前溢出,那么将向 SMU 发送一个警报请求。当退出超时模式时,TO 位自动清 0。

根据停用请求位 WDTxCON1.DR 的状态,从超时模式退出后,可能进入正常模式也可能进入停用模式。

4.2.2 正常模式

在正常模式下(WDTxCON1.DR = 0),WDT 以标准的看门狗方式运行。采用 WDTxCON0.REL 寄存器里用户可定义的重载值对定时器预载,且定时器开始向上计数。必须在计数器溢出之前喂狗。通过发送到控制寄存器 WDTxCON0 的有效访问序列进行喂狗。(见 4.3)在服务序列期间,WDT 转换到超时模式。

假设在定时器溢出之前仍未喂狗,那么可能发生了系统故障。之后正常模式被终止,且向 SMU 发送一条报警请求。

4.2.3 停用模式

***注意:** 不建议使用停用模式!该模式仅适用于那些保证不需要标准看门狗功能的应用。*

当停用请求位(WDTxCON1.DR)和 WDTxCON0.ENDINIT 位都被置位时,从超时模式进入停用模式。
定时器在该模式下停止。

注释: 停用 WDT 仅仅停用了它执行标准的看门狗功能,而不需定期喂狗。但是并未停用超时模式。如果在停用模式里,对寄存器 WDTxCON0 进行了访问,如果访问有效,则仍然进入超时模式,如果访问无效,则向 SMU 发送一条报警请求。ENDINIT 监控功能仍然有效。

4.3 喂狗

如果将 WDT 用于应用中，并因此将其使能(WDTxSR.DS = 0)，就必须在用户定义的超时周期内定期喂狗，以防止看门狗定时器溢出并触发一条 SMU 报警请求。如第 4.1 节所述，溢出时间由 WDTxCON0.REL 字段里给定的用户可编程 16 位重载值定义。因此为了能够正确运行系统，需要定期给有效的 WDT 服务（喂狗）。

通过以下两步喂狗：

•对 WDTxCON0 寄存器有效的密码访问（见表 1）

•对 WDTxCON0 寄存器有效的修改访问（表 3），其中 WDTxCON0.ENDINIT = 1

对 WDTxCON0 寄存器的有效密码访问自动将 WDT 切换到超时模式。定时器被设置为 $FFFC_H$，且计数开始向上计数。在溢出时间过期（定时器溢出）或者是将一条报警请求发送到 SMU 之前，必须执行有效修改访问。

在下一次修改访问期间，有一条严格的要求，那就是将 1 写入 WDTxCON0.ENDINIT。

注释：即使 ENDINIT 已被置 1，也必须将 1 写入 ENDINIT ，以便执行一条有效服务。

在喂狗时，不需要更改重载值 WDTxCON0.REL，或者是用户可定义的密码 WDTxCON0.PW。

当有效地执行了 WDT 服务时，超时模式终止，且 WDT 切换回先前的正常操作模式，显示 WDT 运行完成。

4.4 省电模式期间 WDT 看门狗的操作

如果一些 CPU 处于空闲模式，由于没有软件运行，因此无法向 WDT 发出指令。因此在 CPU 处于空闲模式时，就需要一种管理 WDT 的策略。其中有两种可选方案：

1. 在 CPU 进入空闲模式之前，停用看门狗。然而，这样做的缺点是在空闲模式期间，系统无法受到看门狗的监控。
2. 另外一个更好的解决方案则是依赖 WDT 的唤醒特性。每当 CPU 处于空闲或休眠模式，且 WDT 看门狗未被停用，则定期将 CPU 从节能模式唤醒。当 CPU 将它的计数值 (WDTxSR.TIM)从 $7FFF_H$ 更换到 8000_H 时，CPU 被唤醒，并遵循进入空闲或休眠模式之前的指令在下一个指令处继续执行代码。

注释： 在转换到非运行能量管理模式之前，软件应执行一个看门狗服务序列。在修改访问时，应重新编程看门狗重载值 WDTxCON0.REL，以便定期发生唤醒从而满足应用的最佳需求。两次 CPU 唤醒之间的最大周期是最大 WDT 周期的一半。

4.5 暂停模式支持

只要片上调试支持(OCDS)被使能,则默认 WDT 看门狗被停用。这样就可以避免意外报警和调试程序造成 CPU 中断而引起的 WDT 复位。

然而，在一些情况下，在 WDT 运行时，也要求对系统进行调试。因此，当采用 CBS_OSTATE.WDTSUS 使能 OCDS 时，可能会使能 WDT。在这种情况下，WDT 只有在 OTGS 暂停信号 (CBS_TSL.TL1)有效时，才被暂停。

表 5 总结了在调试期间，WDT 的状态。

注释：更多有关 OCDS 的信息，请参考 AURIX™ TC27x 用户手册[1]。

表 5 WDT OCDS 状态

STCON.STP	WDTxSR.DS	CBS_OSTATE.OEN	CBS_OSTATE.WDTSUS	CBS_TLS.TL1	WDT Action
0	X	X	X	X	Stopped
1	1	X	X	X	Stopped
1	0	0	X	X	Running
1	0	1	0	X	Stopped
1	0	1	1	0	Running
1	0	1	1	1	Stopped

表 5 使用了如下缩写词:

OCDS 片上调试支持

OSTATE OSCU 状态寄存器

OSCU OCDS 系统控制单元

OTGS OCDS 触发开关

STCON 启动配置寄存器

TLS 触发线状态和控制寄存器

5 执行和用途

本章节包含了一系列码组，阐述了 WDT 驱动程序的实现。

注释： 这些代码所显示的函数主要与 CPUx 看门狗相关。安全看门狗的等效函数在 WDT 驱动程序源文件中。

5.1 初始化看门狗

采用代码 2 中所示的 WDT 配置结构 IfxScuWdt_initConfig 所提供的用户定义配置参数，就可以使用代码 1 中给定的函数 IfxScuWdt_initCpuWatchdog 初始化 WDTx 控制寄存器 0 和 1(WDTxCON0 和 WDTxCON1) 。

代码 1 初始化 CPU 看门狗

```
void IfxScuWdt_initCpuWatchdog(Ifx_SCU_WDTCPU *wdt, IfxScuWdt_Config *p_cfg)
{
        Ifx_SCU_WDTCPU_CON0 wdt_con0;
        Ifx_SCU_WDTCPU_CON1 wdt_con1;

        /* Read WDT_CON0 register and clear wdt_con1 variable */
        wdt_con0.U = wdt->CON0.U;
        wdt_con1.U = 0;

        if (wdt_con0.B.LCK)
        {
                /* Password Access:Unlocking WDT_CON0 register */
                wdt_con0.B.ENDINIT = 1;
                 wdt_con0.B.LCK       = 0;
                wdt_con0.B.PW         ^= 0x003F;
                wdt->CON0.U           = wdt_con0.U;
        }

        /* Modify Access:Clear ENDINT and set LCK bit in WDT_CON0 register
         * Set user defined password and WDT reload value */
        wdt_con0.B.ENDINIT = 0;
        wdt_con0.B.LCK       = 1;
        wdt_con0.B.PW        = p_cfg->password;          //user defined password
        wdt_con0.B.REL       = p_cfg->reload;            //user defined reload value
        wdt->CON0.U          = wdt_con0.U;

        /* read back ENDINIT and wait until it has been cleared */
        while (wdt->CON0.B.ENDINIT == 1);

        /* Initialize WDT_CON1 register with user configuration */
        wdt_con1.U           |= p_cfg->inputFrequency;
        wdt_con1.B.DR        = p_cfg->dr;
        wdt_con1.B.UR        = p_cfg->ur;
        wdt_con1.B.PAR       = p_cfg->par;
        wdt_con1.B.TCR       = p_cfg->tcr;
        wdt_con1.B.TCTR      = p_cfg->tctr;
        wdt->CON1.U          = wdt_con1.U;

        /* Initialization finished - set CPU ENDINIT protection */
        IfxScuWdt_setCpuEndinit(p_cfg->password);
}
```

代码 2　WDT 配置结构

```
void IfxScuWdt_initConfig(IfxScuWdt_Config *p_cfg)
{
        p_cfg->password          = 0x003C;
        p_cfg->reload            = 0xFFFC;
        p_cfg->inputFrequency    = IFXSCU_WDT_FDIV_16384;
        p_cfg->dr                = 0;
        p_cfg->ur                = 0;
         p_cfg->par               = 0;
        p_cfg->tcr               = 0;
        p_cfg->tctr              = 0;
        p_cfg->clrirf            = 0;
}
```

5.2　清除 ENDINIT 保护

代码 3 中给出的运行函数 IfxScuWdt_clearCpuEndinit 被用于获取 CPU core ID 并产生一条内联函数以清除 ENDINIT 位，以参数形式传递 CPU 核 ID 和一个有效密码从而用于访问 WDTx 控制寄存器 0(WDTxCON0)。

代码 4 中的内联函数 IfxScuWdt_clearCpuEndinitInline 用于清除相应 WDTxCON0 寄存器里的 ENDINIT 位。

满足表 1 位模式要求的密码访问被用于解锁 WDTxCON0 寄存器，从而修改该寄存器。

满足表 3 位模式要求的修改访问与 WDTxCON0.ENDINIT = 0 一起用于清除 ENDINIT 位。

代码 3　清除 CPU ENDINIT 保护

```
void IfxScuWdt_clearCpuEndinit(uint16 password)
{
        uint8 localCoreId = IfxCpu_getCoreId();
        IfxScuWdt_clearCpuEndinitInline(localCoreId, password);
}
```

代码 4　清除 CPU ENDINIT 保护 - 内联函数

```
IFX_INLINE void IfxScuWdt_clearCpuEndinitInline(uint8 coreId, uint16 password)
{
        /* Select CPU Watchdog based on Core Id */
        Ifx_SCU_WDTCPU *wdt = (Ifx_SCU_WDTCPU *)(IFXSCU_CPUWDT_BASE + 0x0C * coreId);

        /* Read WDT_CON0 register */
         Ifx_SCU_WDTCPU_CON0 wdt_con0;
        wdt_con0.U = wdt->CON0.U;

        if (wdt_con0.B.LCK)
        {
                /* Password Access:Unlocking WDT_CON0 register */
                wdt_con0.B.ENDINIT = 1;
                wdt_con0.B.LCK     = 0;
                wdt_con0.B.PW      = password;
                wdt->CON0.U        = wdt_con0.U;
        }
```

```
        /* Modify Access:Clear ENDINT and set LCK bit in WDT_CON0 register */
        wdt_con0.B.ENDINIT = 0;
        wdt_con0.B.LCK     = 1;
        wdt->CON0.U        = wdt_con0.U;

        /* read back ENDINIT and wait until it has been cleared */
        while (wdt->CON0.B.ENDINIT == 1);
}
```

5.3 设置 ENDINIT 保护

代码 5 中给出的运行函数 IfxScuWdt_setCpuEndinit 被用于获取 CPU 核 ID 并产生一条内联函数以设置 ENDINIT 位，以参数形式传递 CPU 核 ID 和一个有效密码从而用于访问 WDTx 控制寄存器 0(WDTxCON0)。

代码 6 中的内联函数 IfxScuWdt_setCpuEndinitInline 用于设置相应 WDTxCON0 寄存器里的 ENDINIT 位。

满足表 1 位模式要求的密码访问被用于解锁 WDTxCON0 寄存器，从而修改该寄存器。

满足表 3 位模式要求的修改访问与 WDTxCON0.ENDINIT =1 一起被用于设置 ENDINIT 位。

代码 5 设置 CPU ENDINIT 保护

```
void IfxScuWdt_setCpuEndinit(uint16 password)
{
        uint8 localCoreId = IfxCpu_getCoreId();
        IfxScuWdt_setCpuEndinitInline(localCoreId, password);
}
```

代码 6 设置 CPU ENDINIT 保护-内联函数

```
IFX_INLINE void IfxScuWdt_setCpuEndinitInline(uint8 coreId, uint16 password)
{
        /* Select CPU Watchdog based on Core Id */
        Ifx_SCU_WDTCPU *wdt = (Ifx_SCU_WDTCPU *)(IFXSCU_CPUWDT_BASE + 0x0C * coreId);

        /* Read WDT_CON0 register */
        Ifx_SCU_WDTCPU_CON0 wdt_con0;
        wdt_con0.U = wdt->CON0.U;

        if (wdt_con0.B.LCK)
        {
                /* Password Access:Unlocking WDT_CON0 register */
                wdt_con0.B.ENDINIT = 1;
                wdt_con0.B.LCK     = 0;
                wdt_con0.B.PW      = password;
                wdt->CON0.U        = wdt_con0.U;
        }

        /* Modify Access:Set ENDINT and LCK bits in WDT_CON0 register */
        wdt_con0.B.ENDINIT = 1;
        wdt_con0.B.LCK     = 1;
```

```
        wdt->CON0.U          = wdt_con0.U;

        /* read back ENDINIT and wait until it has been set */
        while (wdt->CON0.B.ENDINIT == 0);
}
```

5.4 喂狗

喂狗的步骤与设置 ENDINIT 位的步骤一致。因此，代码 7 中用于喂狗的运行函数同样调用了设置 ENDINIT 保护位的运行函数。

代码 7 喂 CPU 看门狗

```
void IfxScuWdt_serviceCpuWatchdog(uint16 password)
{
        IfxScuWdt_setCpuEndinit(password);
}
```

5.5 更改 ENDINIT 密码

代码 8 中的函数 IfxScuWdt_changeCpuEndinitPassword 用于更改访问 WDTxCON0 寄存器的 ENDINIT 密码。该函数需要当前密码和一个新的密码作为参数.

代码 8 更改 CPU ENDINIT 密码

```
void IfxScuWdt_changeCpuEndinitPassword(uint16 password, uint16 newPassword)
{
        /* Select CPU Watchdog based on Core Id */
        uint8 coreId         = IfxCpu_getCoreId();
        Ifx_SCU_WDTCPU *wdt= (Ifx_SCU_WDTCPU *)(IFXSCU_CPUWDT_BASE + 0x0C * coreId);

        /* Read WDT_CON0 register */
        Ifx_SCU_WDTCPU_CON0 wdt_con0;
        wdt_con0.U = wdt->CON0.U;

        if (wdt_con0.B.LCK)
        {
                /* Password Access:Unlocking WDT_CON0 register */
                wdt_con0.B.ENDINIT = 1;
                wdt_con0.B.LCK       = 0;
                wdt_con0.B.PW        = password;
                wdt->CON0.U          = wdt_con0.U;
        }

        /* Modify Access:Set new Password, ENDINT and LCK bit in WDT_CON0 register */
        wdt_con0.B.ENDINIT = 1;
        wdt_con0.B.LCK       = 1;
        wdt_con0.B.PW        = newPassword;
        wdt->CON0.U          = wdt_con0.U;

        /* read back ENDINIT and wait until it has been set */
        while (wdt->CON0.B.ENDINIT == 0);
}
```

5.6 更改 WDT 重载值

代码 9 中的函数 IfxScuWdt_changeCpuReload 用于更改 WDT 重载值。
该函数需要当前密码和一个新的重载值作为参数.

代码 9 更改 CPU WDT 重载值

```
void IfxScuWdt_changeCpuReload(uint16 password, uint16 reload)
{
        /* Select CPU Watchdog based on Core Id */
        uint8 coreId          = IfxCpu_getCoreId();
        Ifx_SCU_WDTCPU *wdt= (Ifx_SCU_WDTCPU *)(IFXSCU_CPUWDT_BASE + 0x0C * coreId);

        /* Read WDT_CON0 register */
        Ifx_SCU_WDTCPU_CON0 wdt_con0;
        wdt_con0.U = wdt->CON0.U;

        if (wdt_con0.B.LCK)
        {
                /* Password Access:Unlocking WDT_CON0 register */
                wdt_con0.B.ENDINIT = 1;
                wdt_con0.B.LCK       = 0;
                wdt_con0.B.PW        = password;
                wdt->CON0.U          = wdt_con0.U;
        }

        /* Modify Access:Set new Reload value, ENDINT and LCK bit in WDT_CON0 register */
        wdt_con0.B.ENDINIT = 1;
        wdt_con0.B.LCK       = 1;
        wdt_con0.B.REL       = reload;
        wdt->CON0.U          = wdt_con0.U;

        /* read back ENDINIT and wait until it has been set */
        while (wdt->CON0.B.ENDINIT == 0);
}
```

5.7 使能看门狗

代码 10 中的函数 IfxScuWdt_enableCpuWatchdog 用于使能 CPUx 看门狗。这个函数需要当前的密码作为参数。该函数使用现有的运行函数清除并设置 ENDINIT 保护位。

代码 10 使能 CPU 看门狗

```
void IfxScuWdt_enableCpuWatchdog(uint16 password)
{
        /* Select CPU Watchdog based on Core Id */
        uint8 coreId          = IfxCpu_getCoreId();
        Ifx_SCU_WDTCPU *wdt= (Ifx_SCU_WDTCPU *)(IFXSCU_CPUWDT_BASE + 0x0C * coreId);

        IfxScuWdt_clearCpuEndinit(password);
        wdt->CON1.B.DR = 0;                        //Clear DR bit in WDT_CON1 register
        IfxScuWdt_setCpuEndinit(password);
}
```

5.8 停用看门狗

代码 11 中的函数 IfxScuWdt_disableCpuWatchdog 用于停用 CPU 看门狗。这个函数需要当前的密码作为参数。该函数使用现有的运行函数清除并设置 ENDINIT 保护位。

代码 11 停用 CPU 看门狗

```
void IfxScuWdt_disableCpuWatchdog(uint16 password)
{
        /* Select CPU Watchdog based on Core Id */
        uint8 coreId          = IfxCpu_getCoreId();
        Ifx_SCU_WDTCPU *wdt= (Ifx_SCU_WDTCPU *)(IFXSCU_CPUWDT_BASE + 0x0C * coreId);

        IfxScuWdt_clearCpuEndinit(password);
        wdt->CON1.B.DR = 1;                        //Set DR bit in WDT_CON1 register
        IfxScuWdt_setCpuEndinit(password);
}
```

5.9 获取 ENDINIT 密码

代码 12 中的函数 IfxScuWdt_changeCpuEndinitPassword 用于读取 ENDINIT 密码以访问 WDTxCON0 寄存器。该函数将当前密码返回至调用程序。

代码 12 获取 CPU ENDINIT 密码

```
uint16 IfxScuWdt_getCpuEndinitPassword(void)
{
        uint16 password;

        /* Select CPU Watchdog based on Core Id */
        uint8 coreId          = IfxCpu_getCoreId();
        Ifx_SCU_WDTCPU *wdt= (Ifx_SCU_WDTCPU *)(IFXSCU_CPUWDT_BASE + 0x0C * coreId);

        /* Read Password from WDT_CON0 register
         * !!!NOTE:!!! when read, bottom six bits of password are inverted so we have
         * to toggle them before returning the password */
        password    = wdt->CON0.B.PW;
        password ^= 0x003F;

        return (password);
}
```

6 附录

6.1 示例代码:WDT 在启动时初始化

代码 13 中的函数给出了 CPUx WDT 配置示例,以及采用代码 2 看门狗配置结构里提供的用户定义参数的安全看门狗 WDT 示例。该例程应该在系统启动时与其它程序的初始化一起执行。

代码 1 中的函数 IfxScuWdt_initCpuWatchdog 用于初始化 CPU 看门狗。WDT 驱动程序源文件中所提供的等效函数 IfxScuWdt_initSafetyWatchdog 被用于初始化安全看门狗。

注释: 请牢记在心仅仅只有“安全”CPU 可以配置安全看门狗并且喂狗。相同的看门狗配置结构被用于初始化 CPU 和安全看门狗定时器 WDT。用户必须将它们所需的默认值分配到所有的结构成员。

码表 13 WDT 在启动时初始化

```
void fooInit(void)
{
        IfxScuWdt_Config      wdgConfig;            // watchdog Configuration Structure
        Ifx_SCU_WDTCPU        *cpuWdt;              // pointer to CPU WDT object
        Ifx_SCU_WDTS          *sWdt;                // pointer to Safety WDT object

        /* Select CPU Watchdog based on Core Id */
        uint8 coreId = IfxCpu_getCoreId();
        cpuWdt        = (Ifx_SCU_WDTCPU *)(IFXSCU_CPUWDT_BASE + 0x0C * coreId);

        /* Initialize CPUx WDT configuration structure IfxScuWdt_Config with user
         * default parameters.This structure will be used to initialize Watchdog */
         IfxScuWdt_initConfig(&wdgConfig);

        /* !!!NOTE:!!!In case user wants to initialize some of WDT parameters with
         * values differentfrom default values given in IfxScuWdt_initConfig function
         * they should assign new values to particular parameter as follows */
        wdgConfig.password = 0x0AA1;
        wdgConfig.reload     = 0;

        /* Initialize CPU Watchdog with parameters from IfxScuWdt_Config structure */
        IfxScuWdt_initCpuWatchdog(cpuWdt, &wdgConfig);


        /* Only “Safety” CPU should initialize Safety Watchdog as well.
         * Initialize pointer to Safety Watchdog */
        sWdt = (Ifx_SCU_WDTS *)IFXSCU_SWDT_BASE;

        /* Initialize Safety WDT configuration structure IfxScuWdt_Config with user
         * default parameters.This structure will be used to initialize Watchdog */
         IfxScuWdt_initConfig(&wdgConfig);

        /* !!!NOTE:!!!In case user wants to initialize some of WDT parameters with
         * value different from default value given in IfxScuWdt_initConfig function
         * they should assign new value to particular parameter as follows */
        wdgConfig.password = 0x0BB2;
        wdgConfig.reload     = 0;
```

```
        /* Initialize Safety Watchdog with parameters from IfxScuWdt_Config structure */
        IfxScuWdt_initSafetyWatchdog(sWdt, &wdgConfig);
}
```

6.2 示例代码:运行时间 WDT 重新配置

代码 14 的函数是使用用户定义的函数重新配置 CPUx WDT 的示例。

代码 1 中的函数 IfxScuWdt_initCpuWatchdog 用于重新配置 CPUx 看门狗。WDT 驱动程序源文件中所提供的等效函数 IfxScuWdt_initSafetyWatchdog 被用于重新配置安全看门狗。

代码 14　运行时间 WDT 重新配置

```
void fooRunTimeTask(void)
{
        IfxScuWdt_Config  wdgConfig;        // watchdog Configuration Structure

        /* Select CPU Watchdog based on Core Id */
        uint8 coreId      = IfxCpu_getCoreId();
        Ifx_SCU_WDTCPU *wdt= (Ifx_SCU_WDTCPU *)(IFXSCU_CPUWDT_BASE + 0x0C * coreId);

        /* Initialize CPUx WDT configuration structure IfxScuWdt_Config with user
         * default parameters */
        IfxScuWdt_initConfig(&wdgConfig);

        /* Modify CPUx WDT parameters to application needs simply by overwriting values
         * in IfxScuWdt_Config structure and invoking IfxScuWdt_initCpuWatchdog function.
         * !!!NOTE:!!! all parameters including password can be modified here.
         * Parameters that are not modified will retain default values.*/
         wdgConfig.password         = 0x3BBB;                    // new password
         wdgConfig.reload           = 0x7FFF;                    // new reload value
         wdgConfig.inputFrequency   = IFXSCU_WDT_FDIV_64;        // new frequency divider
         wdgConfig.dr               = 0;
         wdgConfig.ur             = 0;
         wdgConfig.par            = 0;
         wdgConfig.tcr            = 0;
         wdgConfig.tctr           = 0;

        /* Configure CPUx WDT with modified parameters */
        IfxScuWdt_initCpuWatchdog(wdt, &wdgConfig);
}
```

w ww.i nf ineon.com

Infineon Technologies AG 出版