AN2632

tinyAVR® 1 系列 IEC 60730 B 类合规指南

特性

- B 类组件测试要求列表
- 用于一般 B 类测试的固件库

说明

IEC 60730 是针对家用电器的安全标准，涉及产品设计和操作的许多方面。其他针对安全关键型设备的标准（例如 IEC 60335）也引用了这一标准。家电必须在系统范围内符合该标准才能通过安全认证。

本应用笔记是关于该标准附录 H 的符合性指南。附录 H 主要涉及电子控制方面的安全要求。本应用笔记将提供适用于 IAR Embedded Workbench®和 AVR® GCC 的经认证固件库和使用示例。固件库涵盖该标准的大部分常规要求。至于如何运行测试，何时运行测试，以及可能需要哪些附加测试，则将取决于应用和实现的选择。一般而言，任何部分出现故障都不应该造成危险。

1. IEC 60730 附录 H 中的定义一章介绍了 IEC 60730 标准的一些定义。2. B 类要求和 3. 组件测试这两章介绍了 B 类软件的要求。 4. tinyAVR 1 系列的 B 类库介绍了 tinyAVR® 1 系列的 B 类库和文件。5. 寄存器、6. 程序计数器、7. 中断处理、8. 系统时钟、9. 存储器和 10. 模拟 I/O 这几章详细介绍了库中的嵌入式自检。 11. tinyAVR 1 系列的其他安全功能介绍了 tinyAVR® 1 系列中包含的其他安全功能。

目录

1. IEC 60730 附录 H 中的定义

1.1 软件类

IEC 60730 安全标准的附录 H 定义了三类家电控制软件：

- A 类——与设备安全性无关的控制功能（H.2.21.1）。
- B 类——包含用于在家电发生故障（软件故障除外）时防止危险的代码的软件（H.2.21.2）。
- C 类——包含用于在不使用其他保护设备的情况下防止危险的代码的软件（H.2.21.3）。

保护控制功能中使用的软件为 B 类或 C 类。本应用笔记涉及适用于大多数家电的 B 类控制。

1.2 控制结构

可以根据定义的三种结构之一设计 B 类控制（H.11.12.2）：

- 支持功能测试的单通道——测试数据在操作之前引入功能单元的单通道结构（H.2.16.5）。
- 支持定期自检的单通道——在操作期间定期测试控制组件的单通道结构（H.2.16.6）。
- 无比较的双通道——包含两个相互独立的功能方法以执行指定操作的结构（H.2.16.1）。

术语“通道”是指家电中 MCU 的数量。单通道结构往往是首选结构，因为它的成本最低。

支持功能测试的单通道结构意味着仅在制造时测试系统。仅当家电不使用任何需要定期测试的组件时，才能使用该结构。定期测试是一种更安全的选项，因为此方法可在产品操作期间检测故障。测试之间的时间间隔通常必须短于相关组件中的故障导致危险所需的时间。

无比较的双通道本质上是这样一种结构：两个 MCU 独立运行不同任务，并且任一 MCU 可以检查另一个 MCU 是否正常运行。一种方法是严格使用一个 MCU 进行监控，而不是在两者之间共享控制任务。在这种情况下，较低成本的器件通常可以用作监控器。

本应用笔记重点介绍单通道结构。

2. B 类要求

对于符合 B 类要求的家电，控制软件必须检测并处理为表 2-1 中的系统组件指定的故障。除了这些测试和故障检测之外，还必须提供适当的软件文档以通过认证。这包括程序序列、控制和数据流、时序、故障树和一般设计原理。

表 2-1. 组件和测试故障/错误概述（表 H.11.12.7）

组件	要测试的组件	测试检测的典型错误
1	CPU	-
1.1	寄存器	卡住
1.3	程序计数器	卡住
2	中断处理和执行	无中断/中断过于频繁
3	时钟	频率错误
4	存储器(1)	-
4.1	不变存储器	所有单个位故障
4.2	可变存储器	DC 故障
4.3	寻址	卡住
5	内部数据路径(1)	-
5.1	数据	卡住
5.2	寻址	地址错误
6	外部通信	-
6.1	数据	汉明距离过长
6.3	时序	时间点错误
7	I/O 外设	-
7.1	数字 I/O	H.27.1 中指定的故障条件(2)
7.2.1	A/D 和 D/A 转换器	H.27.1 中指定的故障条件(2)
7.2.2	模拟多路开关	寻址错误
9	定制芯片（ASIC、GAL 和门阵列等）	任意输出超出静态和动态功能规范

注：

1. 这些对于 AVR 基本相同，因为 SRAM、闪存和 EEPROM 均为内部组件。
2. 该表列出了各种外部组件，并指出是否必须检测短路和/或开路故障。

其中一些测试不可避免地取决于应用。例如，对于 I/O 外设，需要检查输入/输出信号的合理性。而这又取决于应用及其实现。因此，本应用笔记提供的固件库无法满足表 2-1 中的所有要求。

3. 组件测试

本章将介绍表 2-1 中与 tinyAVR® 1 系列相关的部分组件的可接受故障检测措施和注意事项。
注： 如果功能测试未在家电中使用的组件对应的可接受措施下列出，则不能使用单通道结构进行功能测试。

3.1 CPU 寄存器——组件 1.1

测试目的：检测寄存器中的卡住位。

CPU 寄存器是 MCU 最重要的部分，必须进行测试，因为存在故障的寄存器无法正确操作。

可接受的故障检测措施是使用静态存储器测试或带奇偶校验的字保护的功能测试和/或定期自检。随附的库中选择了静态存储器测试，因为奇偶校验需要硬件实现，而 tinyAVR® 1 系列不提供该选项。静态存储器测试可以作为功能测试执行，也可以作为定期自检执行，具体取决于应用要求。

3.2 程序计数器——组件 1.3

测试目的：检测程序计数器中的卡住位。

要使任意软件在 MCU 上成功运行，程序计数器正常工作至关重要。不能针对卡住位对程序计数器直接进行测试，因为任何此类测试都要依赖于程序计数器才能正常工作。

可接受的故障检测措施包括功能测试、定期自检、独立时隙监视或程序序列的逻辑监视。

首选解决方案是使用看门狗对应用程序进行间接时隙监视。如果发生程序计数器故障，看门狗定时器将在错误的时间复位或根本不复位，最终导致器件复位。由于它是一个不可或缺的安全功能，因此必须在常规模式和窗口模式下测试 tinyAVR® 1 系列看门狗能否正常工作，然后才能依赖它来捕捉程序计数器故障。

看门狗测试完成后，应始终将其使能为窗口模式。关闭周期应至少与开启周期一样长，即至少占总周期的 50%。

3.3 中断——组件 2

测试目的：验证中断是否发生并以预期速率处理。

大多数应用依赖于中断进行操作，因此必须验证这些中断是否在预期时间发生并进行了相应处理。

可接受的故障检测措施包括中断的功能测试和/或时隙监视。建议使用时隙监视，因为这有助于在使用家电时检测错误操作。任意中断测试还会间接测试中断控制器。

3.4 时钟——组件 3

测试目的：检测系统时钟频率的意外偏差。

为了使 MCU 正常工作并具有正确的时序，必须验证系统时钟的频率是否符合规范。

可接受的故障检测措施是频率或时隙监视。要求是检测到的振荡频率不会导致应用的正常执行出现问题。

tinyAVR® 1 系列事件系统允许定时器/计数器捕捉由外部时钟信号或实时计数器（Real-Time Counter，RTC）触发，从而实现参考时钟和系统时钟之间的频率比较。

3.5 静态 RAM——组件 4.2、4.3 和 5

测试目的：检测 SRAM 和数据总线上的卡住位和耦合故障，以及任何寻址问题。

内部 SRAM 用于数据的易失性存储，与此相关的任何故障对于家电控制而言都可能是灾难性的。

可接受的故障检测措施包括定期测试和/或数据冗余，例如奇偶校验位。数据冗余十分繁琐，而且没有专门的硬件实现，因此定期测试（例如，March 算法）会是理想之选。

如果无法一次性测试整个 SRAM，则被测试的存储器段必须有所重叠，以便检测耦合故障。

3.6 闪存和 EEPROM——组件 4.1

测试目的：检测非易失性存储器中的所有单个位故障。

在所有 AVR 单片机中，应用软件存储在闪存中，而 EEPROM 存储器可用于存储特定于器件的设置和常量等。为了确保器件安全运行，必须检查这些非易失性存储器是否损坏。

可接受的故障检测措施为使用单个或多个校验和与/或单个位数据冗余的定期自检，例如，硬件中的奇偶校验。建议使用多个校验和，因为这样可以一次校验一个闪存段或 EEPROM 段。之后，该方法将计算目标存储器范围的校验和，然后将结果与存储在非易失性存储器中其他位置的参考校验和进行比较。

3.6.1 有关自编程的注意事项

建议不要在恶劣环境中对闪存进行自编程，因为写入期间的电源故障等事故可能会损坏闪存。

如果必须进行自编程，建议在受锁定位保护的自举程序中执行此操作，以避免意外自改写。在这种情况下，自举程序应当在执行闪存应用程序段中的任何代码之前对闪存应用程序段运行 CRC 检查。

所有 tinyAVR® 1 系列器件的闪存中都包含自举程序段。

3.7 外部通信——组件 6

测试目的：验证所传输的数据以及通信序列和时序是否正确。

与外部器件的通信是许多应用的重要组成部分。但同时也是潜在的故障源，因为通信往往容易受到噪声的影响，并且通信线路的任何一端都可能出现工作异常的情况。因此，必须采取相应的措施以确保其中一个器件中出现噪声或故障时不会导致另一个器件也出现故障。

检测所传输数据中的故障时，可接受的措施为多位数据冗余，例如重复、CRC、汉明码或协议测试。检测通信时序中的故障时，可接受的措施为时隙监视或调度传输。检测通信序列中的故障时，可接受的措施与通信时序类似，只是另外还包括逻辑监视。

简而言之，应使用支持超时、调度和传输冗余且基于状态机的通信驱动程序。

3.8 输入/输出外设——组件 7

测试目的：验证输入/输出是否符合预期以及信号是否正确连接。

所有应用都需要使用模拟和/或数字 I/O，以便检测外设或 MCU 本身的故障。

可接受的故障检测措施为合理性检查，这意味着软件会验证其是否获得了预期的输入并能够在任何给定时间提供所需的输出。

4. tinyAVR® 1 系列的 B 类库

该库的目的是简化基于 tinyAVR® 1 系列单片机的安全可靠应用程序的设计。此外，该库已通过认证，可简化客户的设计和认证流程。

该库经过专门开发，其灵活性足以将自检模块嵌入到许多不同的应用程序中。用户负责配置和使用该库，以使应用程序符合 IEC 60730 B 类标准。

库代码支持 AVR GCC 编译器和 IAR™。其中包含许多示例以说明如何将自诊断程序嵌入用户应用程序中。

该库的源代码已准备好使用 Doxygen（http://www.doxygen.org）自动生成文档。此文档补充了本文档中提供的信息。

源代码不依赖于 Atmel START 代码。某些代码需要配置外设，配置方式可能与这些外设在应用中运行所需的设置不同。测试完成后，并非所有这些配置都会复位。在某些测试中，可通过添加一个函数来复位任何已更改的寄存器。这不是必须使用的，但可以利用它来概览测试后有哪些寄存器发生了改变。

在 ATTINY817 Xplained PRO 上对测试示例进行编译和测试，使用连接到 EXT1 连接器的 OLED1 Xplained 提供按钮和状态 LED。

4.1 错误处理

为了使测试模块尽可能通用，已经定义了许多可由用户配置的错误处理程序。错误分为严重错误和非严重错误，并且在错误处理程序中有相应的默认值。

严重错误是指那些无法处理的错误，例如，当应该返回寄存器自诊断程序结果的寄存器有一个卡住位时。默认情况下，严重错误会挂起 CPU，使其执行无限循环。这会导致看门狗复位，并且在看门狗定时器（以下称为 WDT）发出的系统复位之后，还可以配置将采取的措施。

非严重错误是指那些即使会阻止应用程序正常工作，也可以由程序处理的错误。例如，如果模拟测试失败，程序仍然可以采取一些措施使系统处于安全状态。默认情况下，非严重错误会将全局错误标志置 1。该标志称为 `classb_error`，主应用程序可以使用它将系统置于安全状态。这是示例中遵循的方法。

在所有测试中，如果未检测到错误，`classb_error` 标志为 0；如果检测到错误，标志不为 0。错误标志被指定 NO_INIT 属性，这样可以防止它在 SRAM 中使用的存储空间在启动时被默认值改写。只要器件没有断电，值就会一直保持不变。

`classb_error` 标志位于该库中，在检测到错误时写入 1。可以稍作更改，使错误变量中包含检测到哪类错误的信息。该库中包含的测试尚未实现上述逻辑，因为大部分错误处理视应用而定。

4.2 源文件

源文件可通过访问 www.microchip.com 获取。

项目包含以下文件：

- 通用文件：
  - oled1_xpro_attiny817.h——用于示例的定义以及用于配置示例的按钮和 LED 的代码。
  - classb_error_handler.h——错误处理程序变量和错误处理函数。
  - classb_compiler.h——使 B 类库在 IAR 和 GCC 中都能工作的代码。
- 模拟测试：
  - classb_analog.h——测试 ADC、DAC 和 Vref。
  - main_analog.c——模拟测试的示例代码。
- CRC 测试：
  - main_crc.c——CRC 测试的演示应用程序。
  - classb_crc.h——CRC 测试的定义。
  - classb_crc_hw.h——HW CRCSCAN 模块的简单接口。
  - CRC_16bit_alg.h——基于算法的 16 位 CRC 扫描的代码。

  - CRC_16bit_lookup.h——使用查找表执行的 16 位 CRC 扫描的代码。
  - CRC_32bit_alg.h——基于算法的 32 位 CRC 扫描的代码。
  - CRC_32bit_lookup.h——使用查找表执行的 32 位 CRC 扫描的代码。
- 频率测试：
  - classb_freq.h——频率测试代码。
  - classb_rtc——RTC 配置代码。
  - main_frequency.c——示例代码。
- 中断监视器：
  - classb_interrupt_monitor.h——中断监视器测试代码。
  - main_interrupts.c——示例代码。
- 寄存器测试：
  - classb_cpu.h——用于寄存器测试的宏。
  - classb_cpu_gcc.c——通过 GCC 编译时使用的测试代码。
  - classb_cpu_gcc_asm.s——GCC 寄存器测试的汇编代码。
  - classb_cpu_iar.c——通过 IAR 编译时使用的测试代码。
  - classb_cpu_iar_asm.s——IAR 寄存器测试的汇编代码。
  - main_registers.c——示例代码。
- SRAM 测试：
  - classb_sram.c——MarchX 测试代码。
  - classb_sram.h——定义。
  - main_sram.c——示例代码。
- WDT 测试：
  - classb_wdt_test.c——WDT 测试函数。
  - classb_wdt_test.h——用于在初始化之前和进入主程序之前运行测试的定义和属性。
  - main_wdt.c——示例代码。

注： 一些*.h 文件包含函数定义。这样做的原因在于，GCC 随后能够更轻松地优化代码。使用内联函数时尤为如此。这样一来，GCC 便能够实现与-flto 优化标志相当的代码优化。不同的是，-flto 选项目前不会保留调试信息。但在许多情况下，-flto 选项仍会提供更优的代码长度。如果将-flto 选项与-Os 相结合，则实现的代码长度优化在某些情况下可以与 IAR 相媲美。

5. 寄存器

在该自诊断程序中，将测试 CPU 寄存器是否存在卡住位以及各个寄存器内的每个位间（而非寄存器间）是否存在耦合故障。基本算法的工作原理如下：寄存器的内容压入堆栈，这样便可在测试后恢复寄存器。寄存器设置为已知值并经过验证。随后，强制寄存器位改变状态，并再次验证寄存器内容。

如果任一验证后寄存器的内容不正确，则错误标志置 1。测试完成后，寄存器恢复，原始值压入堆栈。将测试两种类型的寄存器：寄存器文件寄存器（R0-R31）和 I/O 寄存器。它们位于不同的存储空间中，使用不同的指令访问。

寄存器按以下顺序进行测试（GCC 编译器实现）：

1. 返回值寄存器：R24 和 R25。
2. 辅助寄存器：R31 和 R30。
3. 堆栈指针：SPL 和 SPH.
4. 状态寄存器：SREG（中断标志除外）。
5. 寄存器：R0 至 R23 和 R25 至 R29。

在前三个步骤中，针对对自诊断程序十分关键的寄存器执行测试：

- 返回值寄存器：用于提供此自检的结果。
- 辅助寄存器 1 和 2：用于存储必须保留的寄存器的值。
- 堆栈指针：为返回主程序所必需。
- 状态寄存器：由 CPU 指令使用的标志。

辅助寄存器很关键，因为需要使用这些寄存器来测试堆栈指针。如果这些寄存器中存在错误，则默认情况下器件将保持执行无限循环。但是，也可以调用错误处理程序。

在最后的步骤中，将测试其余寄存器。如果出现错误，自诊断程序将调用错误处理程序并返回测试值。但是，建议配置与关键寄存器相同的行为，以便器件在任何寄存器中出现错误时保持执行无限循环。如果任何寄存器测试失败，则在大多数情况下不建议继续执行代码。

自诊断代码使用汇编语言编写，因为在汇编代码中进行寄存器操作更加简单。

示例应用程序将 LED 1 点亮。主程序是一个无限循环，它将多次运行测试，直到发现错误。如果发现错误，则将满足循环退出条件，并且 LED 1 将熄灭，除非辅助寄存器中存在错误。

注： 在实际应用程序中，建议不要使用此行为。测试应在启动时运行，如果发现错误，则不应激活任何功能。通常，如果寄存器发生故障，WDT 将复位器件，因为器件不太可能复位 WDT。

为了模拟错误，可以在自诊断程序中设置断点。应用程序在断点处停止后，可以修改寄存器的内容以模拟卡住位。然后可以恢复执行，自诊断程序将检测错误。

根据经过修改的寄存器，或者 CPU 直接进入自诊断程序内的无限循环，或者全局错误变量设置为 1，这会导致 LED 熄灭，然后进入无限循环。

6. 程序计数器

程序计数器使用看门狗定时器（WDT）进行测试，以便对应用程序进行间接时隙监视。

WDT 是一个用于监视程序是否正常运行的系统功能，允许从跑飞或死锁代码等错误情况下恢复。WDT 是时钟独立于 CPU 的定时器。它配置为预定义的超时周期，并且会在使能后持续运行。

如果 WDT 未在超时周期内复位，则会发出系统复位。通过执行应用程序代码中的 WDT 复位指令（WDR）来复位 WDT。此外，tinyAVR® 1 系列中的 WDT 还具有窗口模式。因此可以在总超时周期内定义时隙或窗口，必须在此时隙或窗口期间复位 WDT。如果在此窗口外复位 WDT，无论过早（窗口关闭）还是过晚，都将发出系统复位。与正常模式相比，这还可以捕捉因错误导致 WDT 复位指令持续执行的情况。因此，要求 B 类软件使用此窗口模式，并且关闭周期至少为总周期的 50%。

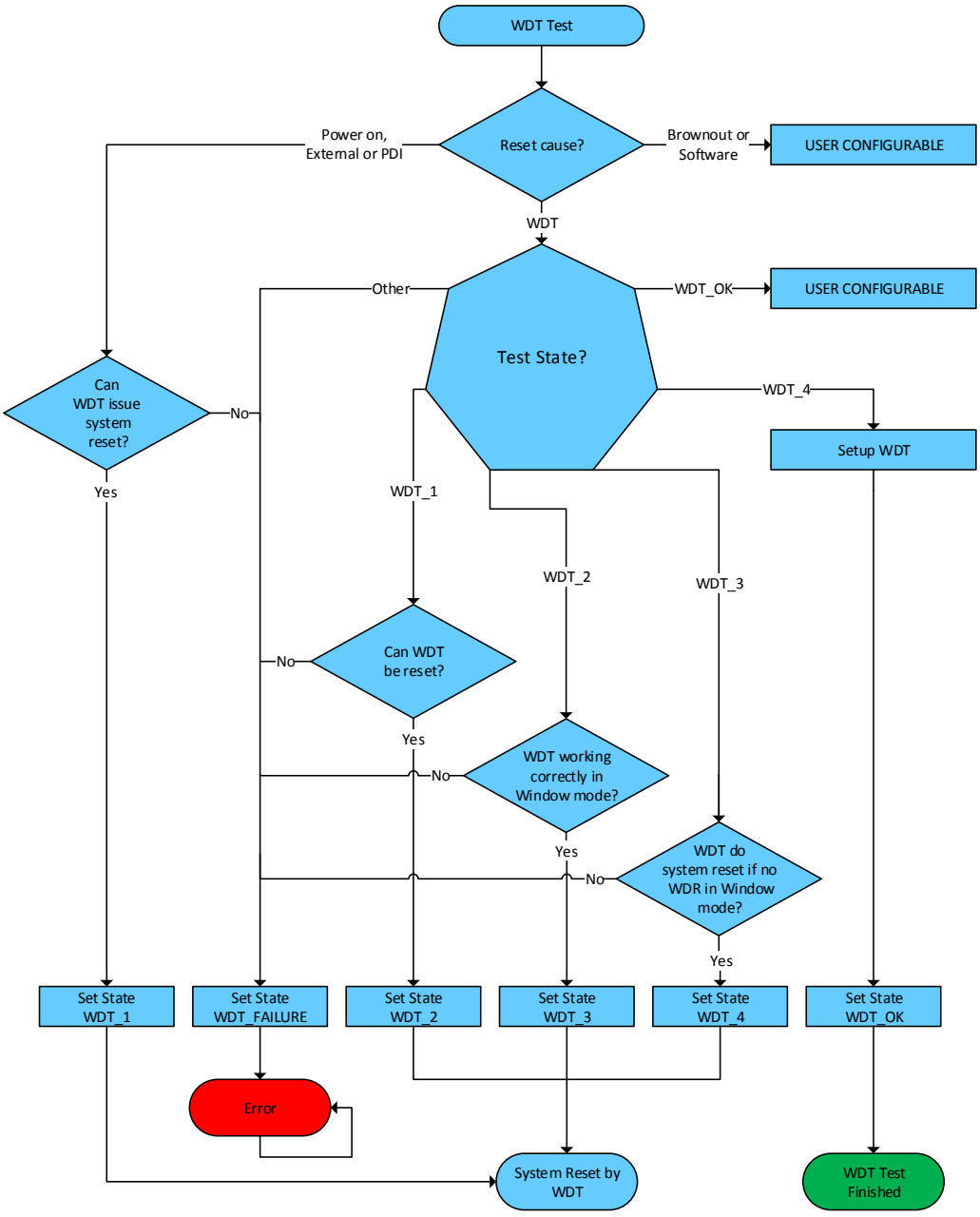
如果使能，WDT 将在工作模式和所有休眠模式下运行。它通过独立于 CPU 的时钟源异步运行，即使主时钟发生故障，它也将继续运行并发出系统复位。

tinyAVR® 1 系列中有一个配置更改保护机制，可确保无法意外更改 WDT 设置。

注： 对于 WDT 和主时钟，可以使用相同的 32 kHz 时钟。如果这样做，WDT 不再满足时隙监视或频率监视的 B 类要求。

由于 WDT 是 tinyAVR® 1 系列中不可或缺的安全功能，因此我们设计了一个在正常模式和窗口模式下测试该模块的自诊断程序。此类测试将于复位后在应用程序的预初始化部分中先于主函数执行。图 6-1 中给出了测试流程图。

图 6-1. 看门狗测试


WDT Test
Reset cause?
Power on, External or PDI
Brownout or Software
USER CONFIGURABLE
WDT
Other
WDT_OK
USER CONFIGURABLE
Test State?
Can WDT issue system reset?
No
Yes
WDT_4
Setup WDT
WDT_1
WDT_2
WDT_3
Can WDT be reset?
No
Yes
WDT working correctly in Window mode?
No
Yes
WDT do system reset if no WDR in Window mode?
No
Yes
Set State WDT_1
Set State WDT_FAILURE
Set State WDT_2
Set State WDT_3
Set State WDT_4
Set State WDT_OK
Error
System Reset by WDT
WDT Test Finished

自诊断程序验证：

- 在正常模式和窗口模式下，WDT 超时后都会发出系统复位
- 在正常模式和窗口模式下均可使用 WDR 指令复位 WDT
- 在窗口模式下 WDT 过早或过晚复位后，器件复位

流程图显示在 WDT 测试期间器件多次复位。自诊断程序使用器件的 SRAM 变量和复位标志来跟踪测试阶段。用户可以配置在发生欠压复位或软件复位时要执行的操作，也可以配置在测试处于“WDT_OK”状态时如何处理由看门狗引起的复位。

自诊断程序使用定时器计数器 A（TCA）来检查 WDT 振荡器的时序。TCA 具有独立于 WDT 时钟的时钟源。TCA 用于估算 WDT 的周期，程序检查该估算值是否处于区间（T/2, T3/2）内，其中 T 是 WDT 的标称周期。TCA 的时钟来自系统时钟。系统时钟通常为内部 20 MHz 时钟。该时钟在温度和电压漂移性能上比 WDT 使用的时钟稳定。

注： TCA 通过该自诊断程序进行隐式测试；如果 TCA 和 WDT 之间的频率差异大于 50%，则设置错误状态。

预期（无错误）执行流程如下：

1. 上电复位或外部复位后，检查 WDT 是否可以发出系统复位：将测试状态设置为“WDT_1”，并通过 WDT 超时复位系统。
2. 检查 WDT 是否可以复位：将测试状态设置为“WDT_2”并在 WDT 超时前将其复位。
3. 检查窗口模式是否正常工作：将 WDT 更改为窗口模式。首先，等待窗口开启并在超时前复位 WDT。然后，将状态更改为“WDT_3”并在窗口关闭时发出复位。这应导致系统复位。
4. 将测试状态设置为“WDT_4”，并通过 WDT 超时复位系统，同时 WDT 配置为窗口模式。
5. 根据应用程序设置来设置 WDT。将测试状态设置为“WDT_OK”并继续执行主函数。

第一步是确保 WDT 可以发出系统复位。方法是设置 WDT 并等待其发出系统复位。此外，TCA 用于估算看门狗周期，测试的后期阶段需要用到此周期。方法是为 TCA 配置一个较短的周期（约 1 ms），然后在系统复位之前一直对 TCA 周期数进行计数。如果程序等待系统复位的时间超过理论上的最大超时周期，则会设置错误状态。

第二步是确保 WDT 可以复位并检查 WDT 的时序。设置错误状态。测试状态将处于错误状态，直到完成测试的这一步。检查估算的 WDT 周期是否大于或等于理论最小值。检查 WDT 和 TCA 之间的频率差异是否处于区间内，以此来确认两个模块是否按预期工作。接下来，设置 WDT 并将 TCA 设置为等待约¾个 WDT 周期（WDT 同步延时）。这将检查 WDT 是否早于预期超时。然后 WDT 复位，程序再次等待¾个周期以发出新的 WDT 复位。原因在于，如果复位 WDT 的机制中存在任何问题，则在程序第二次等待时会发生系统复位。提早系统复位会使测试处于错误状态。在验证 WDT 复位后，测试状态设置为“WDT_2”，程序等待 WDT 超时并发出系统复位。如果 TCA 的计数时间过长，则会发出错误。

第三步检查 WDT 在窗口模式下是否正常工作。这包括将 WDT 配置为窗口模式，等待 WDT 处于开启窗口，然后再发出 WDT 复位。如果 WDT 仅处于开启窗口，则会复位 WDT 而不是整个器件。如果此时复位器件，器件将在启动备份时被检测到。程序继续运行，在 WDT 复位后，WDT 将返回到关闭窗口。测试状态设置为“WDT_3”，当 WDT 处于关闭窗口时将执行 WDT 复位。当窗口关闭时，WDT 应发出系统复位。如果未发生系统复位，则设置错误状态。

在第四步中，WDT 再次配置为窗口模式并保持运行。如果 WDT 按预期运行，则 WDT 将在通过关闭窗口和打开窗口后超时并执行系统复位。如果未在合理时间内发生系统复位，则设置错误状态。

在第五步（也是最后一步）中，程序只需将 WDT 设置为窗口模式并设置“WDT_OK”状态。请注意，在此之后，主应用程序负责根据配置的设置复位 WDT。

如果测试设置错误状态，则可以调用用户可配置的错误处理程序。默认情况下，器件将保持执行无限循环。这被认为是最安全的选择，因为正常工作的 WDT 对于实现可靠的软件应用程序至关重要。

声明计数器和测试状态变量，以便编译器不会在复位后将它们初始化。这样，便可在复位前后使用这些计数器和测试状态变量。tinyAVR® 1 系列有一个存储复位原因的寄存器，用于确定它是否为测试的第一次迭代。

演示应用程序将为按钮设置中断，点亮 LED，并且保持执行只要未设置 `classb_error` 变量 WDT 就会复位的循环。如果设置了此变量，则 LED 将熄灭并且应用程序结束。

按下按钮时，程序将停止复位 WDT，这会使 WDT 超时，从而引起系统复位。这种“意外”的系统复位应由自诊断程序捕捉，在本演示中，自诊断程序配置为设置变量 `classb_error`、设置 WDT 并继续执行主应用程序。按下按钮后，LED 将熄灭，应用程序将不会做出响应，直到上电复位或外部复位。

6.1 如何测试自诊断程序

自诊断程序的第一步将设置错误状态（存在系统复位时除外）。如果 WDT 出现问题，导致无法发出系统复位，则只需不启动 WDT 即可复制该问题。此时会设置错误状态，并且器件会挂起。

可以通过以下方法模拟 WDT 或定时器/计数器的振荡器频率故障：设置断点并修改 `tc_count` 变量的值，使其超出置信区间。

可以通过删除复位 WDT 的代码行来模拟 WDT 复位机制中的故障。给定程序的结构后，除非 WDT 遵循强加的时间限制，否则自诊断程序的第二步将设置错误状态。

可以通过删除 WDT 窗口模式的设置来模拟该模式下的故障。随后代码将复位 WDT 并等待 1/4 个 WDT 周期。此时，WDT 周期将大于或等于估计的 WDT 周期（总超时周期是开启周期和关闭周期的总和）。在设置错误状态之前，不会复位器件。

可以通过演示应用程序来测试 WDT 发出的系统复位所对应的已配置操作。按下按钮将导致 WDT 超时。随后自诊断程序将执行配置的操作，在本演示程序中为设置 WDT，并设置变量 `classb_error`，然后熄灭 LED。

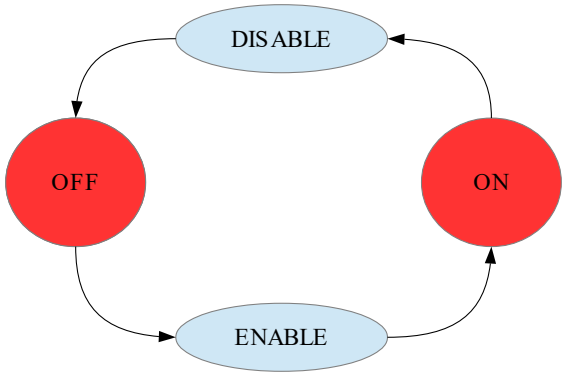
7. 中断处理

中断指示外设的状态变化，可用于改变程序执行。嵌入式应用使用中断来实现实时响应事件等目的。因此，系统可以检测中断功能中的错误是很重要的。具体来说，B 类应用程序应能够检测中断是否未按预期经常执行（或根本未执行）。

所选方法是使用实时计数器（RTC，其时钟独立于 CPU 时钟）进行时隙监视。简而言之，每次执行任何要监视的中断时都必须使计数器递增。RTC 会定期产生中断，以便检查中断计数器。如果任何计数器超出中断特定的可配置范围，则会调用错误处理程序。

中断监视器会检查已注册中断和已激活中断的频率。注册中断意味着为中断监视器提供有关待检查中断的以下信息：标识符、预期频率和偏差容差。中断监视器使用这些信息创建数据结构。激活中断告知监视器它应该开始检查待监视的已注册中断：可以使用状态变量为每个已注册的中断单独开启和关闭中断监视器。各状态之间可能的转换如图 7-1 所示。

图 7-1. 中断状态



对于任何中断，默认状态均为 OFF。在此状态下，中断管理器不检查中断的频率。当主应用程序希望监视器开始检查中断频率时，应将中断状态设置为 ENABLE。下次执行中断监视器时，会将中断状态更改为 ON。这样可确保中断计数器与中断监视器同步。中断计数器恰好在中断监视器的一个周期后开始递增。同样，如果主应用程序决定不再监视中断，则应将中断状态设置为 DISABLE。中断监视器将在下次执行时将状态更改为 OFF。

注： 每个 RTC 周期应该只有一个允许/禁止请求。

实现的中断监视器有两项功能可进一步提高主应用程序的稳健性。第一项功能是，中断计数器仅在中断为 ON 时才会递增，因此在中断为 OFF 时计数器应为零。所有已注册的中断都由中断计数器进行相关检查。如果检测到错误，则将调用错误处理程序。第二项功能是，如果定义了常量 CLASSB_STRICT，则允许 ON 状态的中断或禁止 OFF 状态的中断时将调用错误处理程序。

RTC 会定期调用中断监视功能。所有已注册的中断都按照其状态进行处理。然后会检查活动中断的频率。如果没有错误，则中断监视器将复位计数器。如果发现错误，则将调用错误处理程序，同时监视器立即结束（这是可配置的）。为确保一致性，应检查非活动中断的计数器是否为零。如果主应用程序已请求激活中断，则其状态设置为 ON，反之亦然。请注意，当状态为 OFF 时，中断计数器设置为零。

要监视中断，应遵循以下步骤：

1. 应通过为中断提供标识符在 `classb_int_identifiers` 中声明中断。
2. 主应用程序必须通过调用 `classb_intmon_reg_int()` 来注册中断。随后将为中断创建一个结构。
3. 必须将 RTC 设置为定期产生中断并回调中断监视器。
4. 每次执行必须监视的中断时应调用 `classb_intmon_increase()`。
5. 主应用程序必须请求监视器开始检查中断，具体通过使用 `classb_intmon_set_state()` 将中断状态更改为 ENABLE 来完成。
6. 如果某些时候不应再监视中断，主应用程序可以将状态更改为 DISABLE。

在示例应用程序中，定时器/计数器（TC）设置为定期产生溢出中断，由监视器检查。此外，应用程序还设置了两个按钮中断。第一个用于更改 TC 中断的频率。第二个用于在监视器中禁止中断。LED 点亮表示应用程序正常工作。只要没有按下按钮，应用程序就会保持执行循环，同时 LED 指示灯会点亮。如果按下第一个按钮，TC 中断的频率将改

变，监视器应将错误标志置 1。随后，主应用程序将退出循环并熄灭 LED。按下第二个按钮可以在监视器中禁止 TC 中断。在这种情况下，按下第一个按钮仍然会更改中断频率，但监视器不会产生错误。

中断监视器依靠正常运行的 RTC 来工作。RTC 还支持 CPU 频率测试，并在相关软件模块中对 RTC 进行检查。因此，可以假定 RTC 中没有故障。为了提高应用程序的可靠性，已注册的中断数量（可以在中断内进行测试）可达计数器的最大值。如果计数器达到此值，则可以假定中断监视器不在工作。

可以通过以下多种方式来测试监视器。

- 可以设置中断，然后修改其频率以产生错误。这种方式在在示例应用程序中已实现。
- 可以在调试时修改中断计数器。这种方式对于活动中断和非活动中断均适用。
- 如果定义了符号 CLASSB_STRICT，则程序可以激活或禁止中断两次，从而产生错误

8. 系统时钟

自诊断程序采用实时计数器（RTC）和定时器/计数器（Timer/Counter，TC）实现。选择 TC 是因为该外设的时钟域与 CPU 相同。不过，RTC 也可以由独立的时钟源提供时钟。例如，CPU 由内部 20 MHz 振荡器提供时钟，而 RTC 由内部 32 kHz 振荡器提供时钟。

RTC 设置为定期产生中断。在该中断中，软件 16 位 TC 溢出计数器会与其预期值进行比较，如果在给定的可配置范围内，则清零计数器值。否则，将调用错误处理程序。

在 TC 溢出中断中，16 位溢出计数器变量会递增。使用软件计数器值的原因在于，TC 在大多数情况下的运行频率远高于 RTC。TC 的溢出速度比 RTC 快得多。如果溢出计数器高于可配置阈值，则调用错误处理程序。鉴于溢出计数器通过 RTC 中断清零，溢出计数高于阈值将意味着 RTC 的频率与 TC 的频率不匹配。

自检模块将检测系统时钟、内部 32 kHz 振荡器、RTC 或 TC 的故障。因此，漏检错误的风险得以降低，RTC 和 TC 时钟的故障会给出其频率的正确比值。

示例应用程序与前面的示例类似。默认情况下，tinyAVR 1 系列器件依靠内部 20 MHz 振荡器（预分频至 3.3 MHz）运行，该频率为系统频率，在设置自诊断程序的参数时需要加以考虑。如果按下按钮，则 TC 周期会发生变化。这将会导致测试失败，LED 也将随之熄灭。

可以在与 RTC 相关的文件中配置 RTC 频率和 RTC 中断周期。请注意，只要时钟源独立于 CPU 时钟并且根据 RTC 设置定义常量 RTC_INTERUP_PERIOD_TIME，不同的 RTC 设置便将与自诊断程序兼容。此外，还可以配置 TC 模块、预分频比、测试容差（百分比）和系统频率。配置的系统频率必须与主应用程序设置的实际系统频率相对应。自诊断程序不会根据该设置更改时钟系统。

可通过以下多种方法来测试该自诊断程序。

- 按下按钮后，示例应用程序可能会更改系统频率
- 可以修改系统频率常量 F_CPU，使其与实际系统频率不匹配。与此同时，还可以修改容差值进一步测试。
- 为了模拟 RTC 或 TC 中的问题，可以将设置函数注释掉

9. 存储器

本章介绍存储器的标准要求和已实现的自诊断测试。9.1 不变存储器介绍不变存储器以及 tinyAVR® 1 系列中的内部闪存和 EEPROM。9.2 可变存储器介绍可变存储器，即 SRAM。

9.1 不变存储器

已实现循环冗余校验（Cyclic Redundancy Check，CRC）以测试不变存储器。这是一种用于查找数据中意外错误的错误检测技术。该方法通常用于确定数据传输以及数据和程序存储器中存在的数据的正确性。

CRC 算法处理输入数据流或数据块，并生成稍后可用于检测错误的输出校验和。执行此操作的两种常用方法包括：

- 计算并存储数据的校验和。为了检测错误，将重新计算相同数据的校验和并将其与前一个校验和进行比较。如果它们不同，则表示存在错误。
- 计算数据的校验和并将其附加到数据段。数据计算得到的校验和加上包含的 CRC 校验和应该得到恒定的 CRC 值。如果新的校验和不是正确的值，则表示数据已更改，并且存在错误。

应用于任意长度数据块的 n 位 CRC 通常会检测不超过 n 位的任何单个错误突发，并会检测 $1/(1-2^{-n})$的所有较长错误突发。如果数据中存在错误，应用程序应采取一些纠正措施。

提供基于硬件和软件的自诊断模块。支持两种常用的 CRC 标准：

- 16 位 CRC CCITT
- 32 位 CRC IEEE® 802.3

所有 tinyAVR® 1 系列器件都可以使用该软件实现。在这种情况下，CPU 读取数据并计算 CRC 校验和。可以在两个软件实现之间进行选择：

- 查找表：该方法使用 CRC 查找表来加速计算。查找表需要 512（适用于 16 位）或 1024（适用于 32 位）字节的闪存。
- 直接计算：该方法使用多项式除法计算每个字节的校验和。该方法不占用闪存中的空间，但速度慢于查找表方法。

在软件实现中，使用的 32 位 CRC 多项式为 0xEDB88320，初始余数为 0xFFFFFFFF，生成的校验和经过位取反和求补。使用的 CCITT 16 位 CRC 多项式为 0x1021，初始余数为 0x0000。在这种情况下，校验和既不会经过位取反也不会经过位求补。

tinyAVR® 1 系列中的硬件实现只能检查闪存的内容，而不能检查 EEPROM 或 SRAM 的内容。计算的 CRC 是 16 位宽，并且 CRC 机制依赖于被检查闪存的最后部分的 CRC 校验和。可以通过 ISR 或需要在 CRC 完成后进行检查的状态标志来发出故障信号。CRC 模块可完全访问闪存，并且 CPU 不会同时运行。在启动时通过熔丝设置启动 CPU 之前，也可以使 CRC 运行，我们建议这样操作。

CRC 扫描时间取决于扫描的闪存大小和主时钟频率。扫描将在三个主时钟周期内处理 16 位数据。此外，启动和停止大约 20 个主时钟周期的开销很小。如果需要在应用程序运行期间测试闪存，则必须在应用程序可以处理单片机无响应的情况下完成，而无论闪存扫描多长时间。

以下函数可用于计算存储在 EEPROM 中的数据的校验和：

- CLASSB_CRC16_EEPROM_SW
- CLASSB_CRC32_EEPROM_SW

为计算闪存内容的 CRC，将使用硬件模块。可以在 classb_crc_hw.h 中找到该模块的驱动程序。

注： 由于已配置硬件和软件实现，因此等效 CRC 算法会产生相同的校验和。但是，处理时间存在明显差异。

示例应用程序使用器件上的 CRCSCAN 模块来检查闪存的完整性。执行测试时 CPU 不运行。配置按钮，以便在按下该按钮时写入闪存中的页。完成此操作后，CRC 扫描将失败，激活的不可屏蔽中断（NMI）将执行并熄灭 LED。
注： NMI 无法禁止。发现错误后，它将一直保持有效状态，退出 ISR 后将立即重新进入 ISR 代码，直到 WDT 触发并复位器件。

注： 为了使 CRC 不失败，需要计算 CRC 校验和并将其附加到闪存。这可以通过在 Studio 中添加编译后命令来实现。可在 AN2521 “CRCSCAN on Devices in the tinyAVR® 1-Series” 中找到此命令以及添加它的方法。

再次按下按钮将不起任何作用。为了检查所有等效的 CRC 计算是否会得出相同的校验和，可以调用它们并比较结果。注意，用于软件实现（查找或直接计算）的算法由相应头文件中的设置选择，并且在一次执行期间只能调用一种算法。由于选择一种算法或另一种算法将修改闪存中的内容，因此应比较不同 EEPROM 执行的校验和。

9.2 可变存储器

已选择 March 算法来测试可变存储器。具体实现的 March 算法称为 March X 测试，可用如下方式描述：

⇳(w0);⇑(r0,,w1);⇓(r1,w0);⇳(r0)

第一阶段是以任意顺序将 0 写入所有存储单元。第二阶段包括三项操作（逐位执行），从最低地址开始：

- 读取某个位并验证其是否为 0。如果为 1，则表示发生了故障
- 向其存储单元写入 1
- 为下一个位重复上述操作

第三阶段也包括三项操作，但以相反的地址顺序完成（相对于第二阶段）：

- 读取某个位并验证其是否为 1。如果为 0，则表示发生了故障
- 向其存储单元写入 0
- 为下一个位重复上述操作

第四阶段也是最后一个阶段以任意顺序验证所有位是否为 0。请注意，第二阶段和第三阶段中实际使用的地址顺序无关紧要，只要这两个阶段的顺序完全相反即可。此测试与更为常见的 March C 测试很类似，只是跳过了步骤 3 和步骤 4。

March X 可以检测到以下故障：

- 地址解码器故障
- 单个单元故障：卡住、数据跳变或数据保持故障
- 存储单元之间的故障：部分（并非全部）可能存在的耦合故障（Coupling Fault，CF）。

注： 检测所有可能存在的耦合故障很难及时完成。具体原因有很多，其中一个来自 SRAM 的分区。由于测试需要在 SRAM 的分区上运行，因此测试无法检查分区之间是否存在某些 CF。即使各分区有所重叠，也仍然无法保证一定检测得到，只能说降低了漏检 CF 的风险。

上述 March 测试是针对面向位的存储器（Bit-Oriented Memory，BOM）定义的。AVR 中的 SRAM 是面向字的存储器（Word-Oriented Memory，WOM）。将 r0、r1、w0 和 w1 分别替换为 r^D、$r^{\overline{D}}$、w^D 和 $w^{\overline{D}}$，其中 D 可以是任意数据背景，BOM March 测试将转换为涵盖字间 CF 的 WOM March 测试。在我们的实现中，选择了数据背景 D = 0x00。

必须考虑存储器单元阵列行中各个位的物理位置。所提出的自诊断程序可配置为附加具有背景序列{0x55, 0xAA, 0x33, 0xCC, 0x0F, 0xF0}的额外 March 元素。这将增加不受限制的字内 CF 模型中考虑的字内状态 CF 的覆盖范围。

为了能够对 SRAM 中的应用数据运行测试，存储器将被划分成若干个存储段（数量可配置）逐个进行测试。测试的最简单行为出现在各存储段之间没有重叠时。在这种情况下，除了最后一个存储段之外，所有存储段的大小都相同。第一个存储段（称为缓冲区）保留，用于在测试期间存储其他被测存储段的内容。鉴于实现的 March 测试具有破坏性，因此这一项很有必要。

鉴于 March X 算法一次在一个存储段上运行，当各存储段之间存在用户可配置的重叠时，将会降低漏检字间 CF 的概率。每次测试一个存储段时，前一存储段也会有一部分被再次测试。请注意，这一点不适用于缓冲区，因为它是第一个存储段。缓冲区的大小需要根据前面的情况进行扩展（第二个存储段的大小相应地减小）。

我们提供了一个示例应用程序，用于说明如何将存储器测试嵌入到应用程序中。指示系统行为是否正确的 LED 指示灯将点亮，然后程序保持执行循环，只要没有错误，就会测试 SRAM 存储器。

可以按照如下方式测试自诊断程序：

- 在实现 March X 算法的函数中设置若干个断点
- 修改一些存储单元的内容以构建不同类型字间耦合故障的模型

然后，测试模块会将错误标志置 1，从而使应用程序退出主循环并且 LED 将熄灭。

9.3 更大涵盖范围

考虑到 SRAM、闪存和 EEPROM 是 tinyAVR® 1 系列的内部存储器，前述自诊断程序涵盖了存储器寻址和内部数据路径的标准规范（表 H.11.12.7 中的子组件 4.3 和组件 5）。

10. 模拟 I/O

库中可用的模拟测试仅对内部信号执行。库中提供了两种不同的测试功能：

1. 器件内部连接的 DAC 所产生电压的 ADC 读数。每个外设的参考电压由带隙电压生成。
2. 这与上面类似，但将 V_{DD} 作为 ADC 参考电压。

所有测试的运行均无需符合 B 类标准。

测试 1 和测试 2 非常相似，但是如果器件以稳定的 V_{DD} 运行，则测试 2 的覆盖范围更广一些，因为它更有可能找到带隙电压的偏差。由于测试 2 需要更高的 V_{DD} 精度，因此不得将其用于电池应用或测试过程中 V_{DD} 可能存在较大噪声的应用中。如果 V_{DD} 有噪声，或者在不同电路中差异很大，也可能需要从 ADC 获得更高的可接受转换值范围。

测试 1 比测试 2 和测试 3 更简单。由于 DAC 和 ADC 均从带隙电压获得参考电压，因此无法检测带隙是否出错。它可以检测带隙提供的两个独立参考发生器是否不同或出错。

所有测试都将测试 DAC、ADC 和参考电压系统，但是，如果测试失败，则无法分辨其中哪一项失败。例如，如果 ADC 是惟一使用的元件，测试用于验证 ADC 的行为是否正确，则 DAC 或 DAC 的参考发生器中存在的错误可能导致测试失败。这可能导致应用程序即使在安全的情况下也无法执行。

示例代码在连续循环中执行测试 1 和测试 2，两个 LED 分别指示测试状态。如果测试失败，则会熄灭相应的 LED。

可以通过不同方式验证测试程序：

- 可以提高 ADC 的时钟速度
- 可以调整可接受的转换范围，这样测试便会失败
- 可以改变 DAC 或 ADC 的参考电压

在本示例中，BUTTON1 中断将 ADC 的参考电压更改为 1.1V（测试 1），而 BUTTON2 将 ADC 和 DAC 参考电压更改为 1.1V（测试 2）。这将导致 ADC 转换结果超出预期限值，测试应失败。

11. tinyAVR 1 系列的其他安全功能

tinyAVR® 1 系列具有一些可用于提高安全性的附加功能，但不能直接满足任何 B 类要求：

- 如果不遵循定时代码序列，则配置更改保护（Configuration Change Protection，CCP）会阻止更改某些 I/O 寄存器以及写入/读取非易失性存储器
- 应用程序软件可以轻松读取和验证熔丝
- 欠压检测（Brown-Out Detection，BOD）、具有中断的可编程电压监视器（Voltage Level Monitor，VLM）和上电复位（Power-On Reset，POR）可防止器件在低于安全电压的条件下工作
- 时钟系统会在切换时钟源之前检查振荡器的稳定性

12. 附录——术语和缩写

- 汉明距离——对于二进制数，它可以表示为 A XOR B 的结果中 1 的数量。
- 静态存储器测试——非易失性存储器的测试。
- 时隙监视——对设定时间段内所发生事件执行的某种连续测试。
- March 算法——按顺序执行 RAM 写入并在稍后分多个阶段验证 RAM 中内容的算法。
- 耦合故障（CF）——不同存储单元之间的连接，一个单元发生变化会导致两个单元均发生变化。

13. 致谢

在本文档中，“IEC 60730 标准”以及本标准中的所有其他定义和部分均指：IEC 60730-1 3.2 版，Copyright© 2007 IEC Geneva, Switzerland, www.iec.ch。

本文作者感谢国际电工委员会（International Electrotechnical Commission，IEC）允许从其国际出版物 IEC 60730-1 3.2 版（2007）中复制信息。

14. 参考资料和推荐文献

- “IEC 60730-1: Automatic Electrical Controls for Household and Similar Use”，国际电工委员会，3.2 版，2007 年 3 月。
- 《Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI》，作者：Michael Lee Bushnell 和 Vishwani D. Agrawal
- “A Designer’s Guide to Built-In Self-Test”，C. E. Stroud，Kluwer Academic Publishers，2002 年
- AVR040：EMC Design Considerations
- AVR042：AVR Hardware Design Consideration
- AN2521“CRCSCAN on Devices in the tinyAVR® 1-Series”

15. 版本历史

文档版本	日期	备注
D	2019 年 12 月	删除了第 11 章中的“内部温度传感器可用于检测超出器件规范的工作条件”。
C	2019 年 7 月	根据客户反馈更新了程序计数器主题
B	2018 年 10 月	删除了 START 的链接，增加了 microchip.com 的链接。在“tinyAVR 1 系列的其他安全功能”一章中增加了 VLM。
A	2018 年 2 月	初始文档版本

Microchip 网站

Microchip 网站（http://www.microchip.com/）为客户提供在线支持。客户可通过该网站方便地获取文件和信息。我们的网站提供以下内容：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及归档软件
- **一般技术支持**——常见问题解答（FAQ）、技术支持请求、在线讨论组以及 Microchip 设计伙伴计划成员名单
- **Microchip 业务**——产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、Microchip 销售办事处、代理商以及工厂代表列表

产品变更通知服务

Microchip 的产品变更通知服务有助于客户了解 Microchip 产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请访问 http://www.microchip.com/pcn，然后按照注册说明进行操作。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（ESE）
- 技术支持

客户应联系其代理商、代表或 ESE 寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过 http://www.microchip.com/support 获得网上技术支持。

Microchip 器件代码保护功能

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下，Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿意与关心代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。

代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

本出版物中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担

商标

ISBN：978-1-5224-5789-3

质量管理体系

有关 Microchip 的质量管理体系的信息，请访问 http://www.microchip.com/quality。

全球销售及服务网点

美洲	亚太地区	亚太地区	欧洲
公司总部 2355 West Chandler Blvd. Chandler, AZ 85224-6199 电话：480-792-7200 传真：480-792-7277 技术支持： http://www.microchip.com/support 网址： http://www.microchip.com **亚特兰大** 德卢斯，佐治亚州 电话：678-957-9614 传真：678-957-1455 **奥斯汀，德克萨斯州** 电话：512-257-3370 **波士顿** 韦斯特伯鲁，马萨诸塞州 电话：774-760-0087 传真：774-760-0088 **芝加哥** 艾塔斯卡，伊利诺伊州 电话：630-285-0071 传真：630-285-0075 **达拉斯** 阿迪森，德克萨斯州 电话：972-818-7423 传真：972-818-2924 **底特律** 诺维，密歇根州 电话：248-848-4000 **休斯顿，德克萨斯州** 电话：281-894-5983 **印第安纳波利斯** 诺布尔斯维尔，印第安纳州 电话：317-773-8323 传真：317-773-5453 电话：317-536-2380 **洛杉矶** 米慎维荷，加利福尼亚州 电话：949-462-9523 传真：949-462-9608 电话：951-273-7800 **罗利，北卡罗来纳州** 电话：919-844-7510 **纽约，纽约州** 电话：631-435-6000 **圣何塞，加利福尼亚州** 电话：408-735-9110 电话：408-436-4270 **加拿大 - 多伦多** 电话：905-695-1980 传真：905-695-2078	**澳大利亚 - 悉尼** 电话：61-2-9868-6733 **中国 - 北京** 电话：86-10-8569-7000 **中国 - 成都** 电话：86-28-8665-5511 **中国 - 重庆** 电话：86-23-8980-9588 **中国 - 东莞** 电话：86-769-8702-9880 **中国 - 广州** 电话：86-20-8755-8029 **中国 - 杭州** 电话：86-571-8792-8115 **中国 - 香港特别行政区** 电话：852-2943-5100 **中国 - 南京** 电话：86-25-8473-2460 **中国 - 青岛** 电话：86-532-8502-7355 **中国 - 上海** 电话：86-21-3326-8000 **中国 - 沈阳** 电话：86-24-2334-2829 **中国 - 深圳** 电话：86-755-8864-2200 **中国 - 苏州** 电话：86-186-6233-1526 **中国 - 武汉** 电话：86-27-5980-5300 **中国 - 西安** 电话：86-29-8833-7252 **中国 - 厦门** 电话：86-592-2388138 **中国 - 珠海** 电话：86-756-3210040	**印度 - 班加罗尔** 电话：91-80-3090-4444 **印度 - 新德里** 电话：91-11-4160-8631 **印度 - 浦那** 电话：91-20-4121-0141 **日本 - 大阪** 电话：81-6-6152-7160 **日本 - 东京** 电话：81-3-6880-3770 **韩国 - 大邱** 电话：82-53-744-4301 **韩国 - 首尔** 电话：82-2-554-7200 **马来西亚 - 吉隆坡** 电话：60-3-7651-7906 **马来西亚 - 槟榔屿** 电话：60-4-227-8870 **菲律宾 - 马尼拉** 电话：63-2-634-9065 **新加坡** 电话：65-6334-8870 **台湾地区 - 新竹** 电话：886-3-577-8366 **台湾地区 - 高雄** 电话：886-7-213-7830 **台湾地区 - 台北** 电话：886-2-2508-8600 **泰国 - 曼谷** 电话：66-2-694-1351 **越南 - 胡志明市** 电话：84-28-5448-2100	**奥地利 - 韦尔斯** 电话：43-7242-2244-39 传真：43-7242-2244-393 **丹麦 - 哥本哈根** 电话：45-4485-5910 传真：45-4485-2829 **芬兰 - 埃斯波** 电话：358-9-4520-820 **法国 - 巴黎** 电话：33-1-69-53-63-20 传真：33-1-69-30-90-79 **德国 - 加兴** 电话：49-8931-9700 **德国 - 哈恩** 电话：49-2129-3766400 **德国 - 海尔布隆** 电话：49-7131-72400 **德国 - 卡尔斯鲁厄** 电话：49-721-625370 **德国 - 慕尼黑** 电话：49-89-627-144-0 传真：49-89-627-144-44 **德国 - 罗森海姆** 电话：49-8031-354-560 **以色列 - 若那那市** 电话：972-9-744-7705 **意大利 - 米兰** 电话：39-0331-742611 传真：39-0331-466781 **意大利 - 帕多瓦** 电话：39-049-7625286 **荷兰 - 德卢内市** 电话：31-416-690399 传真：31-416-690340 **挪威 - 特隆赫姆** 电话：47-72884388 **波兰 - 华沙** 电话：48-22-3325737 **罗马尼亚 - 布加勒斯特** 电话：40-21-407-87-50 **西班牙 - 马德里** 电话：34-91-708-08-90 传真：34-91-708-08-91 **瑞典 - 哥德堡** 电话：46-31-704-60-40 **瑞典 - 斯德哥尔摩** 电话：46-8-5090-4654 **英国 - 沃金厄姆** 电话：44-118-921-5800 传真：44-118-921-5820