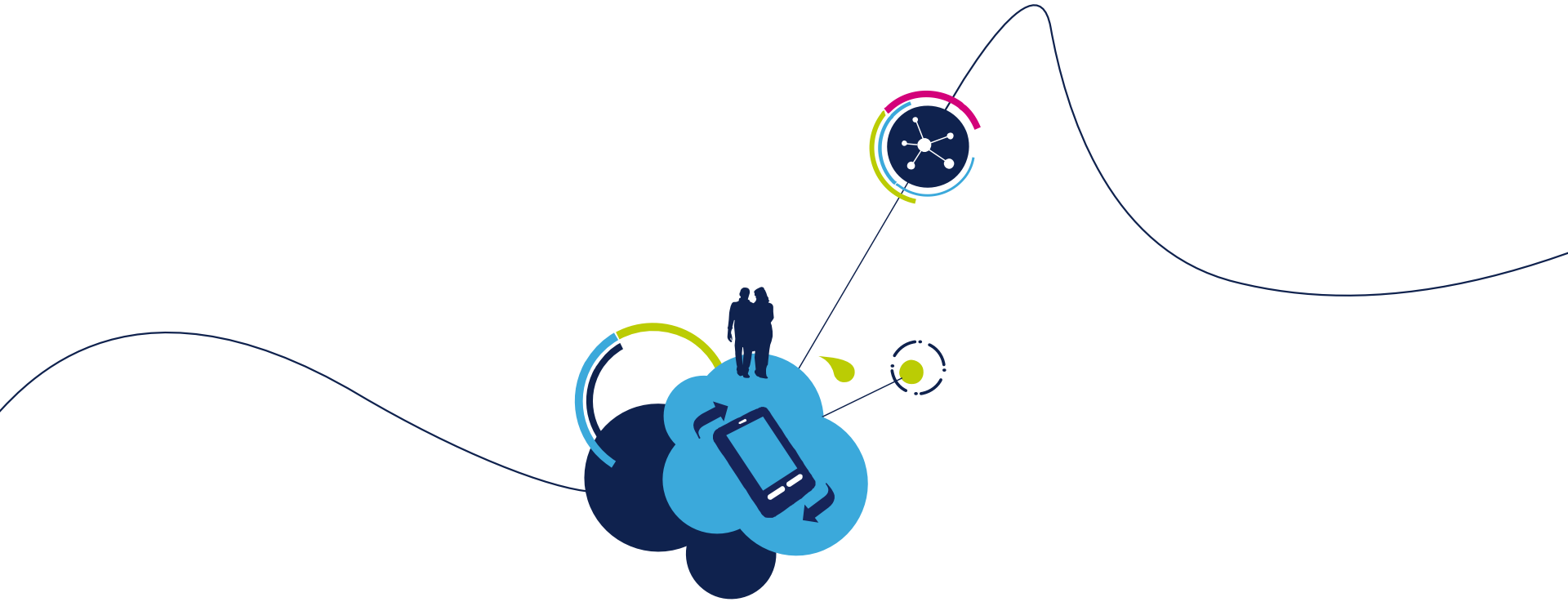
ST的Class B软件库介绍

life.augmented

目录



什么是Class B?

软件评估

软件评估的目的

- 对软件运行时的风险控制措施进行评估
- 确保电器使用安全

软件评估的内容包括

- 对硬件结构和软件结构的综合检查
- 对软件开发过程的评估

软件评估的对象

- 家用电器如果同时具备以下两个条件，就应当进行软件评估：
  - 使用可编程电子电路，即嵌入式微控制器MCU
  - 可编程电子电路具有安全保护功能。

  比如具有过热控制的电磁炉，带自动门锁控制的洗衣机等

- 如果MCU仅实现产品功能，安全保护由硬件进行，这类家电不需要进行软件评估。

标准和软件分类

- 2004年，IEC颁布了IEC60335-1：2001(Ed.4.0)的修订版A1：2004（Ed.4.1),首次提出家用电器软件评估要求。
- IEC60335-1附录R，参照了IEC60730附录H的H.2，H.7，H.11.12进行修改。定义了软件评估的要求和检查方法。
- 国标GB4706.1-2005参照了IEC60335-1:2004（Ed.4.1)也引入了家用电器软件评估的要求

IEC60730的附录H(H.2.22)中对软件进行了分类

A类软件：软件仅实现产品的功能，不涉及产品的安全控制。比如室用恒温器的软件，灯光控制的软件…

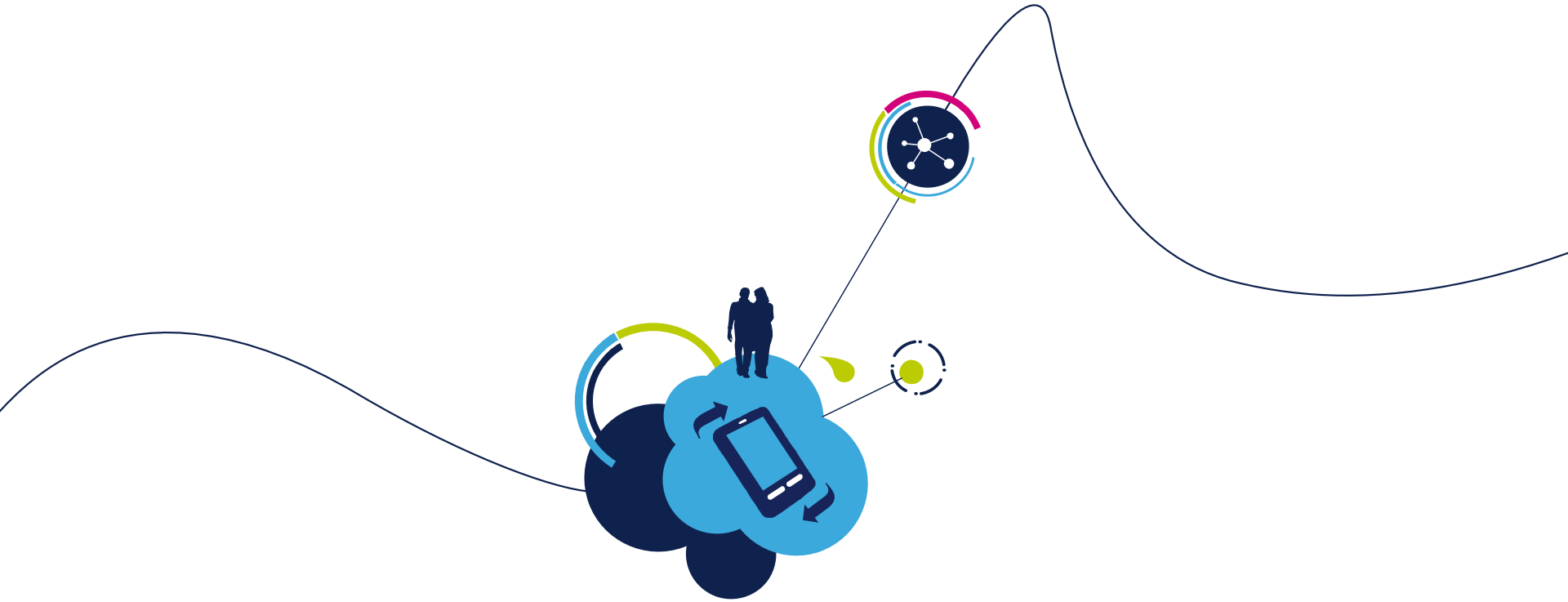
B类软件：软件的设计要防止电子设备的不安全操作；例如带电控门锁的洗衣机软件，带电机过热检测的洗碗机水泵驱动…

C类软件：软件的设计为了避免某些特殊的危险；例如automatic burner controls, gas fired controlled dryer（主要针对一些会引起爆炸的设备）

IEC 60730
表H中定义

ClassB软件需要检测的故障/错误

需要检测的组件		故障/错误	MCU相关故障	应用相关故障	ST提供库
1.CPU	1.1 寄存器	滞位(Stuck at)	是	否	√
	1.3 程序计数器	滞位(Stuck at)	是	否	√
2.中断		没有中断或者中断太频繁	否	是	X
3.时钟		错误的频率	是	否	√
4.存储器	4.1 非易失存储器	所有的单比特错误	是	否	√
	4.2 易失存储器	DC fault	是	否	√
	4.3 寻址（与非易失和易失存储器相关的）	滞位(Stuck at)	是	否	√
5.内部数据路径	5.1 数据	滞位(Stuck at)	是	否	√
	5.2 寻址	错误的地址	是	否	√
6.外部通信	6.1 数据	汉明距离3	否	是	X
	6.2 寻址	错误的地址	否	是	X
	6.3 时序	错误的时序	否	是	X
7.输入输出	7.1 数字I/O	H27中定义的错误	否	是	X
	7.2 模拟输入输出（AD,DA,模拟复用器）	H27中定义的错误； 错误的寻址	否	是	X



ST Class B软件库结构

ClassB程序

Class B程序分为两个部分： 启动时的自检和运行时的自检

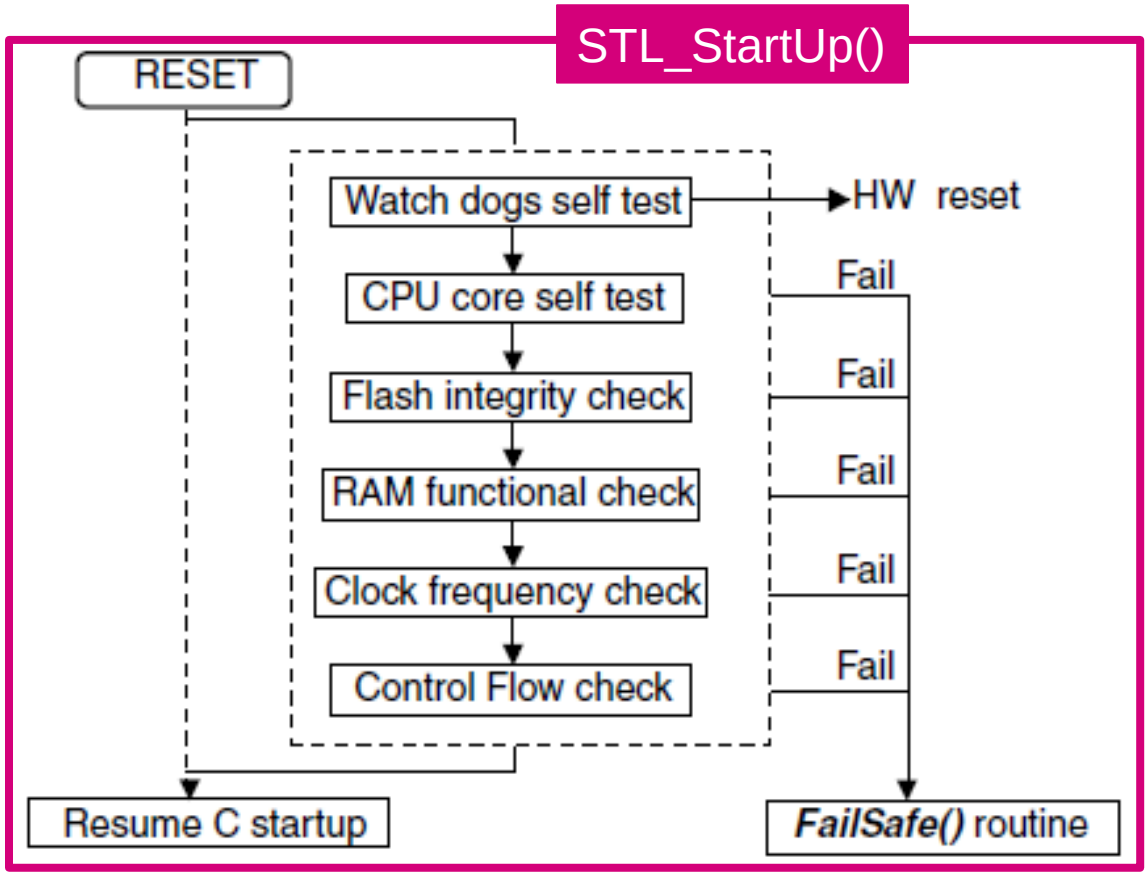
启动时的自检包括：

- 看门狗自检
- CPU寄存器自检
- Flash完整性自检
- RAM功能自检
- 系统时钟自检
- 控制流自检

运行时的周期自检包括：

- 局部CPU寄存器自检
- 堆栈边界检测
- 系统时钟自检
- Flash自检
- 控制流自检
- 局部RAM自检(在中断服务程序中进行）

启动时的自检流程



STM8中相关的文件

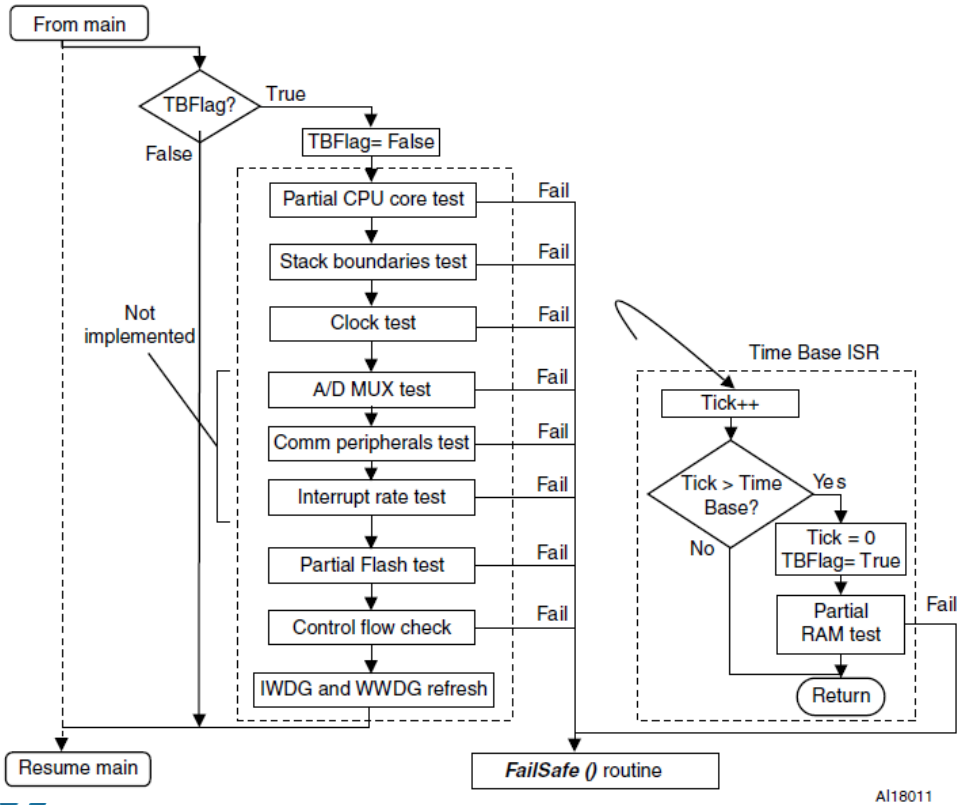
- stm8s_stl_startup.c
- stm8s_stl_cpustart_CSMC.s
- _classb_cksumxx.s
- stm8s_stl_fullRam_CSMC.s
- stm8s_stl_clockstart.c

STM32中相关的文件

- stm32fxxx_STLstartup.c
- stm32fxxx_STLcpustartIAR.s
- stm32fxxx_STLcrcxx.c
- stm32f0xx_STLfullRamMc.c
- stm32fxxx_STLclockstart.c

运行时周期自检流程

- 通过timer产生一个周期定时中断，启动Run time自检
- 执行Run time自检前必须调用**STL_InitRunTimeChecks()** 进行初始化
- 除了对RAM的检测，其他的都在主循环里进行
- 对RAM的检测，在周期定时中断服务程序里进行

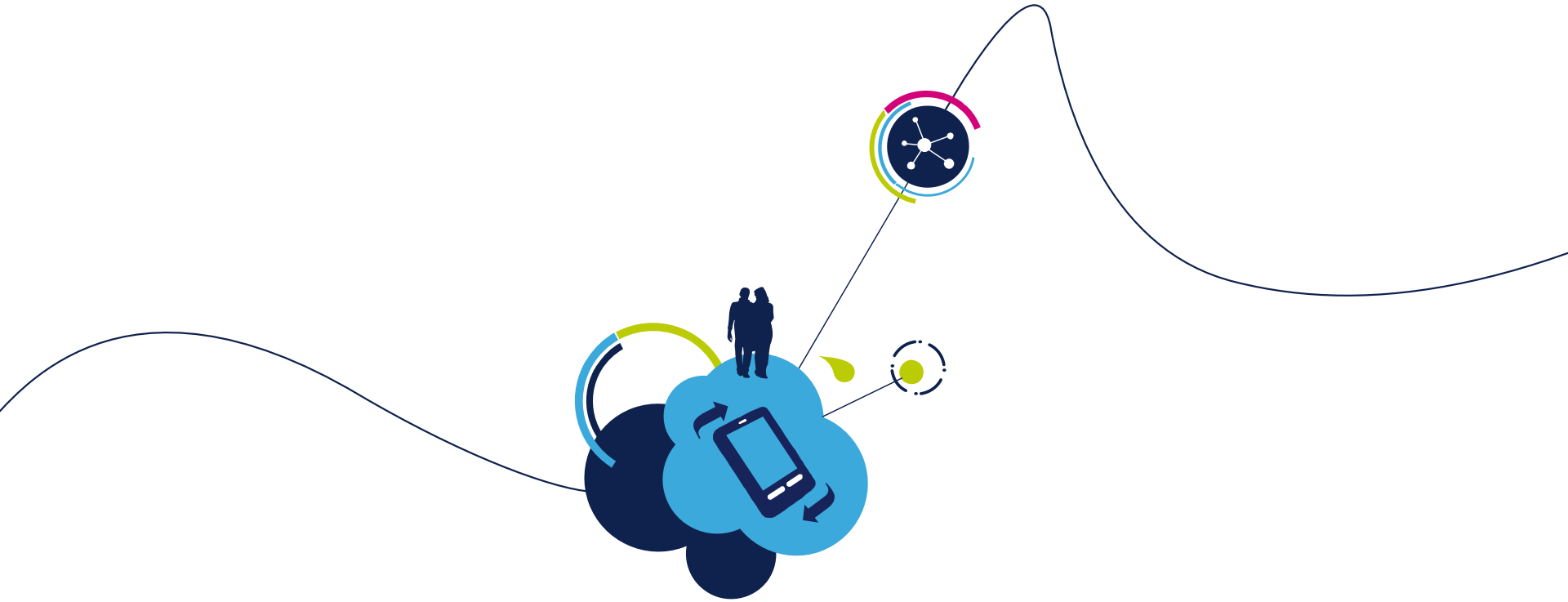


STM8中相关的文件

- stm8x_stl_main.c
- stm8x_stl_cpurun_CSMC.s
- stm8x_stl_crcrun.c
- _block_cksumxx.s
- stm8x_stl_transpRam_Mc.c
- stm8x_stl_transpRam_Mx.c
- stm8x_stl_clockrun.c

STM32中相关的文件

- stm32fxxx_STLmain.c
- stm32fxxx_STLcpurunIAR.s
- stm32fxxx_STLcrcxxRun.c
- stm32fxxx_STLtranspRamMc.c
- stm32fxxx_STLtranspRamMx.c
- stm32fxxx_STLclockrun.c



故障&措施具体介绍
——MCU相关故障检测方法

life.augmented

1.CPU

- 1. CPU
  - 1.1 CPU寄存器
  - 1.3 程序计数器
- 3.系统时钟
- 4.存储器
  - 4.1 非易失存储器---Flash
  - 4.2 易失存储器---RAM
  - 4.3 寻址
- 5.内部数据路径
  - 5.1 数据
  - 5.2 寻址

1.1 CPU寄存器
故障：滞位(Stuck-at)

寄存器的某一位或几个位总为0或者1，故障状态固定不变。

IEC60730 H表格中给出的可选的检测措施：

- 启动时的功能测试 或者 周期自检
- 测试方法：
  - H.2.19.6 静态存储器测试
    - ✓ H.2.19.6.1 Checkerboard 存储器测试
    - ✓ H.2.19.6.2 marching存储器测试
  - H.2.19.8.2 带有一位冗余的字保护

CPU寄存器故障检测——STM32

启动时：

- 标志位检测：Z(zero), N(negative), C(carry), V(overflow)
- 寄存器功能检测：R0~R12，PSP,MSP
- 调用STL_StartUpCPUTest()函数
- 检测到错误后，程序跳到Fail Safe程序

运行时：

- 仅检测寄存器R0~R12
- 调用STL_RunTimeCPUTest()函数
- 检测到错误后，程序跳到Fail Safe程序

对应的代码位置：

- IAR：stm32fxxx_STLcpustartIAR.s
  stm32fxxx_STLcpurunIAR.s
- Keil：stm32fxxx_STLcpustartKEIL.s
  stm32fxxx_STLcpurunKEIL.s

CPU寄存器故障检测——STM8

启动时：

- 检测标志位：Z(zero),N(negative),C(carry)……
- 检测寄存器 A, X, Y, SP
- 调用STL_StartUpCPUTest()函数
- 检测到错误后，程序跳到Fail Safe程序

运行时：

- 检测寄存器 A, X, Y
- 调用STL_RunTimeCPUTest()函数
- 检测到错误后，程序跳到Fail Safe程序

stm8s_stl_cpustart_CSMC.s

```
_STL_StartUpCPUTest:

  LDW X,_CtrlFlowCnt
  ADDW  X,#3     ; CtrlFlowCnt += CPU_INIT_CALLEE
  LDW _CtrlFlowCnt,x
  ; If X not functional, corruption will be detected later
;
  ; Check flags of code condition register
  CLR A           ; Set Z(ero) Flag
  JRNE ErrorCPU   ; Fails if Z=0
  LD A,#1         ; Reset Z Flag
  JREQ ErrorCPU   ; Fails if Z=1
;
  SUB A,#2        ; Set N(egative) Flag (A=0xFF)
  JRPL ErrorCPU   ; Fails if N=0
  ADD A,#2        ; Reset N and set C Flags (Res=0x101)
  JRMI ErrorCPU   ; Fails if N=1
;
```

对应的代码位置：

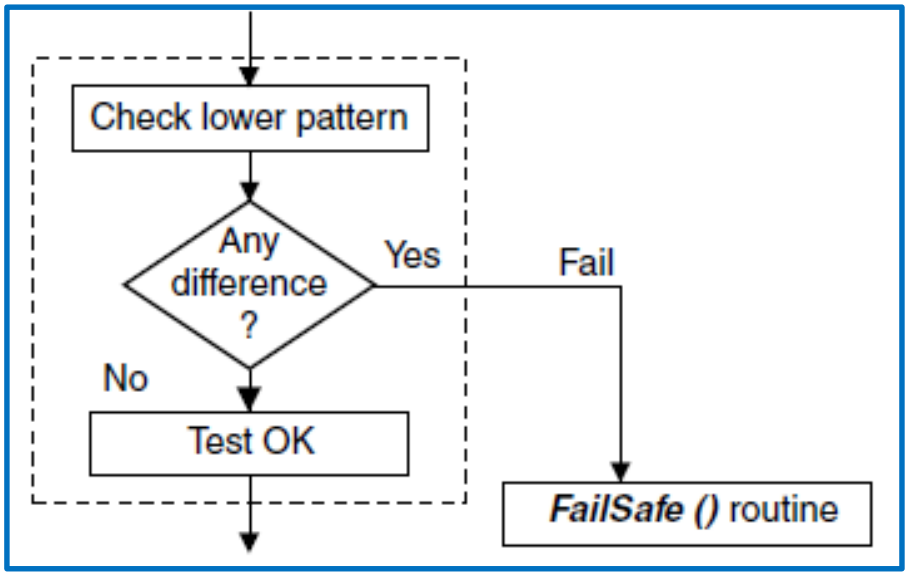
- Cosmic下使用
  stm8s_stl_cpustart_CSMC.s/stm8s_stl_cpurun_CSMC.s
- IAR下使用
  stm8s_stl_cpustart_IAR.asm/ stm8s_stl_cpurun_IAR.asm
- Raisonance下使用
  stm8s_stl_cpustart_RAIS.asm/stm8s_stl_cpurun_RAIS.asm

堆栈溢出检测

部分STM8芯片还提供
Stack roll-over limit机制
可以防止堆栈的溢出

仅在程序运行中进行检测

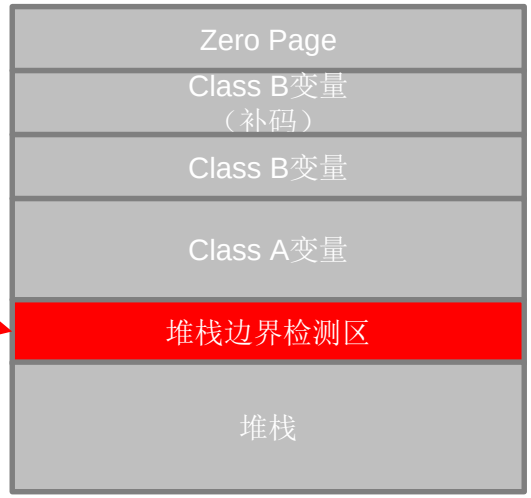
- 在紧跟堆栈区的低地址位置，定义为堆栈边界检测区。放置特殊的数值。
- 没有发生堆栈溢出时，该区域里的值是不变的。



STM8举例

```
#ifdef STL_INCL_RUN_STACK
  /* For stack overflow detection function */
  StackOverFlowPtrn[0] = 0xAAu;
  StackOverFlowPtrn[1] = 0xBBu;
  StackOverFlowPtrn[2] = 0xCCu;
  StackOverFlowPtrn[3] = 0xDDu;
#endif /* STL_INCL_RUN_STACK */
```

在RAM中的位置



该检测在STL_RunTimeCPUTest()函数中实现

1.3 程序计数器
故障：滞位(Stuck-at)

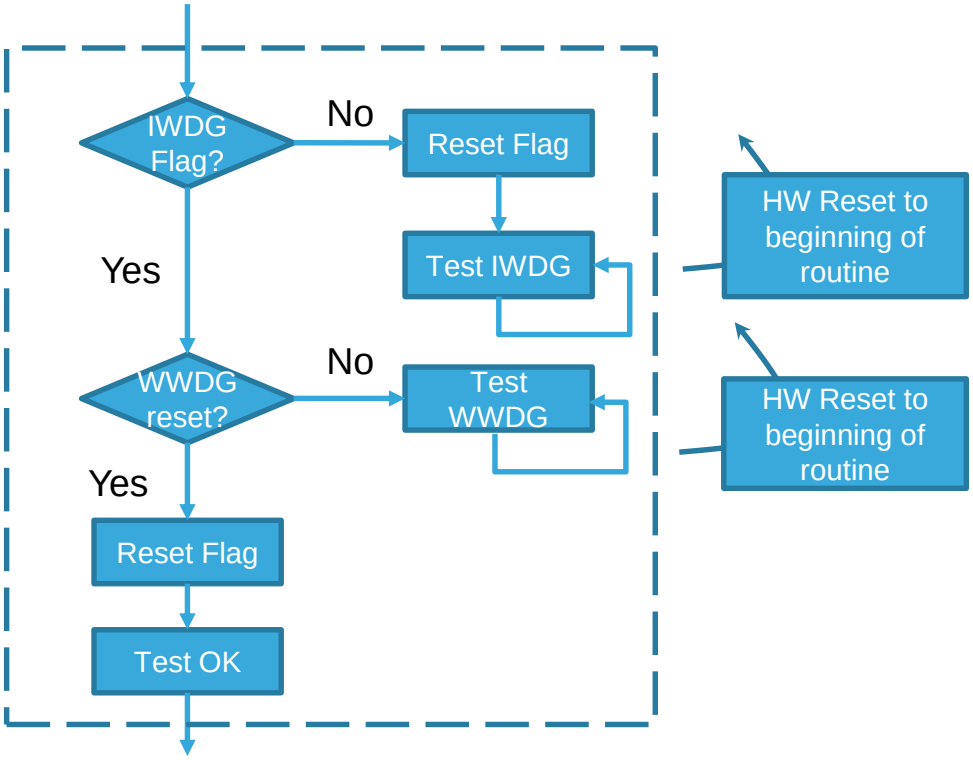
IEC60730 H表格中给出的可选的检测措施：

- 启动时的功能测试 或者 周期自检
- 测试方法：
  - H.2.18.10.4 独立的时间片监测
    - watchdog
  - H.2.18.10.2 程序顺序的逻辑监测

CPU程序计数器故障

ST芯片可提供的方法：

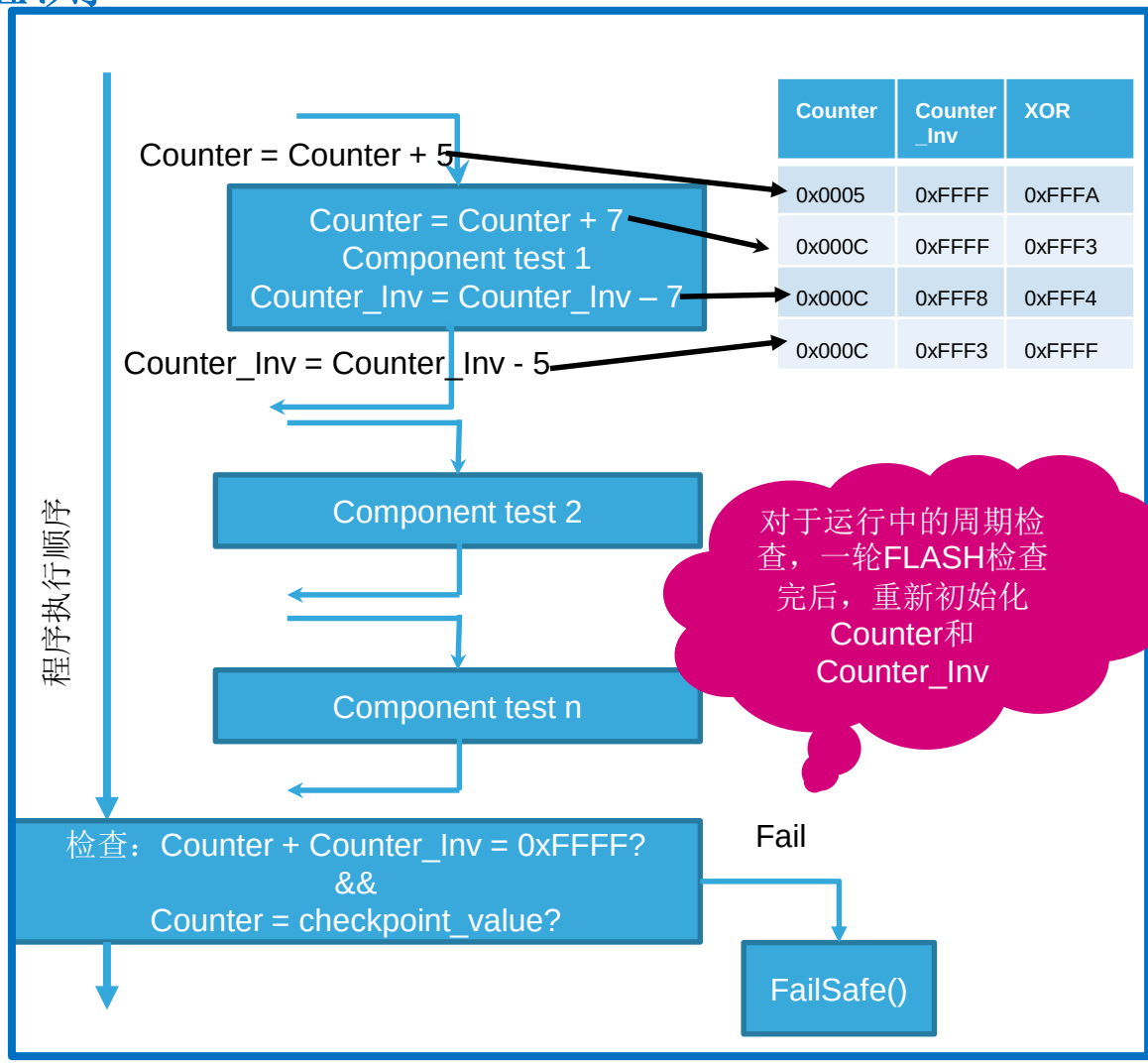
- ST芯片有两个不同时钟源的看门狗：Window watchdog and Independent watchdog
- 当PC滞位(Stuck at)在某个位置或者跑飞时，WWDG和IWDG可以对芯片复位
- ST的ClassB软件库提供了代码，在系统启动时对WWDG和IWDG进行检测，保证WWDG和IWDG工作正常。



CPU程序计数器故障

ST例程：程序控制流监测

- 检查程序是否被正确的调用（check if the block is correctly called from main flow level）
- 检查程序是否正确的完成（if the block is correctly Completed）
- 检测的方法
  - 定义两个变量用来计数，初始化为0和0xFFFF
  - 为需要检测的软件执行模块定义两个的数值,这两个值是不同且唯一的
  - 执行到要检测的软件模块时，进行图中的4个步骤
  - 软件模块执行完后，对counter和counter_Inv进行检查
  - 如果不对则进到FailSafe程序



life.augmented

3.系统时钟故障

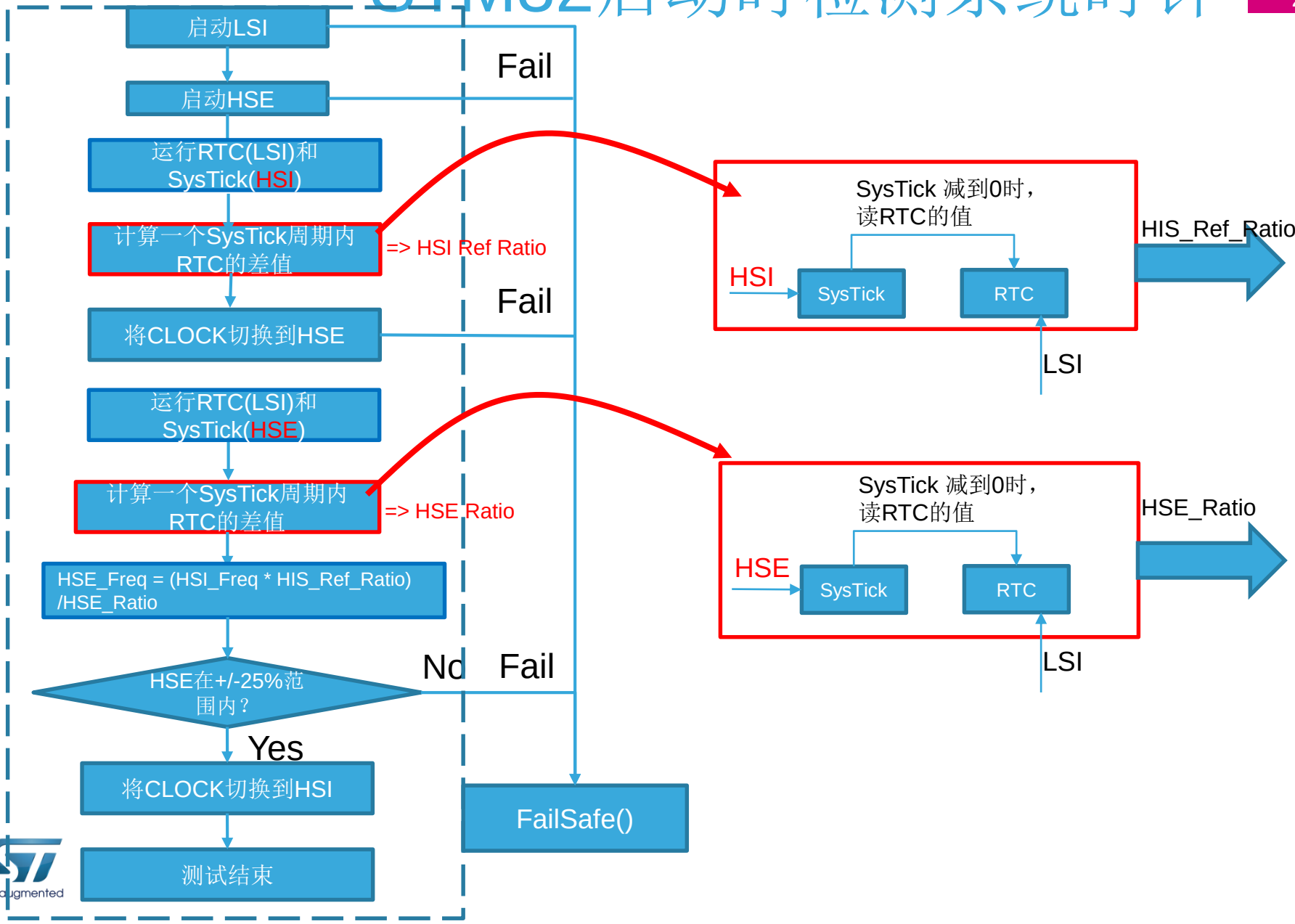
- 1. CPU
  - 1.1 CPU寄存器
  - 1.3 程序计数器
- 3.时钟
- 4.存储器
  - 4.1 非易失存储器---Flash
  - 4.2 易失存储器---RAM
  - 4.3 寻址
- 5.内部数据路径
  - 5.1 数据
  - 5.2 寻址

3.时钟
故障：错误的频率

IEC60730 H表格中给出的可选检测措施：

- H.2.18.10.1 频率监测
  - 将时钟频率与一个独立的固定频率相比较。
- H.2.18.10.4 独立的时间片监测

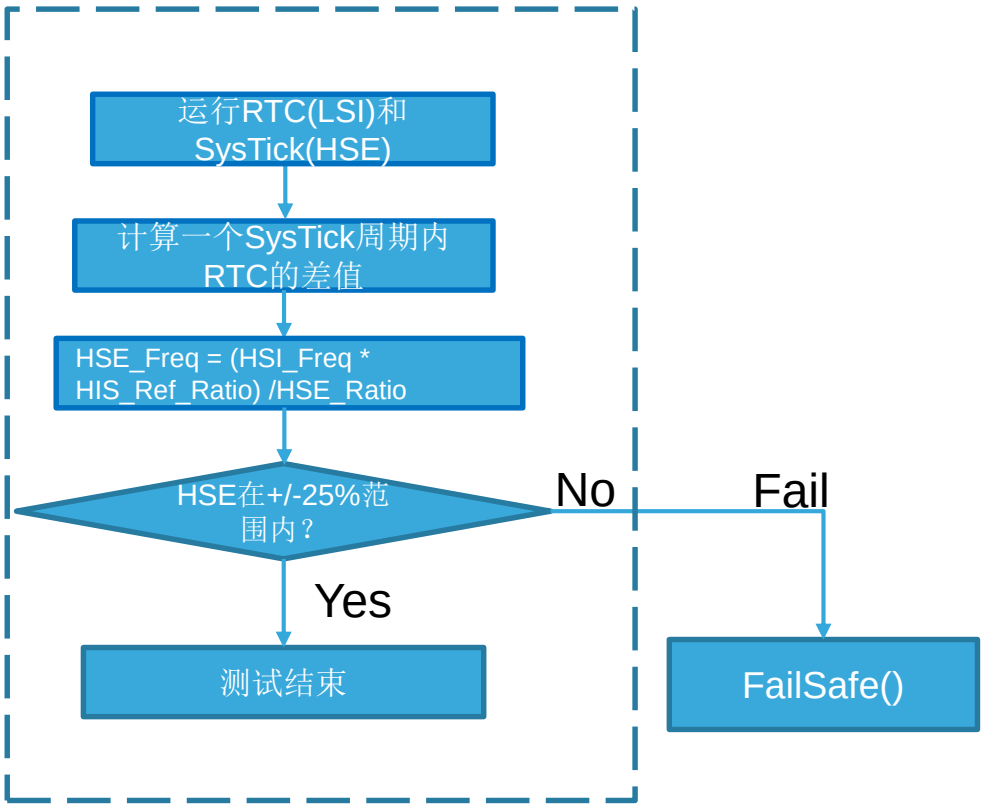
STM32启动时检测系统时钟



STM32运行时检测系统时钟

运行时监测时钟

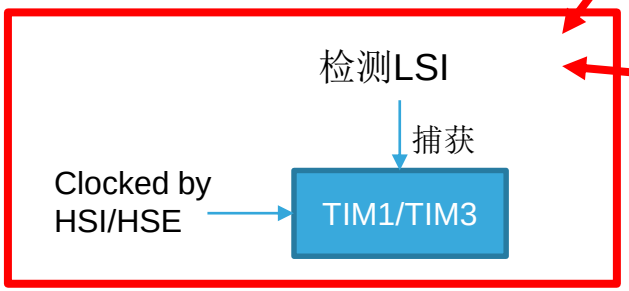
- 监测HSE系统时钟。
- 参考值使用系统启动自测时保存的值。

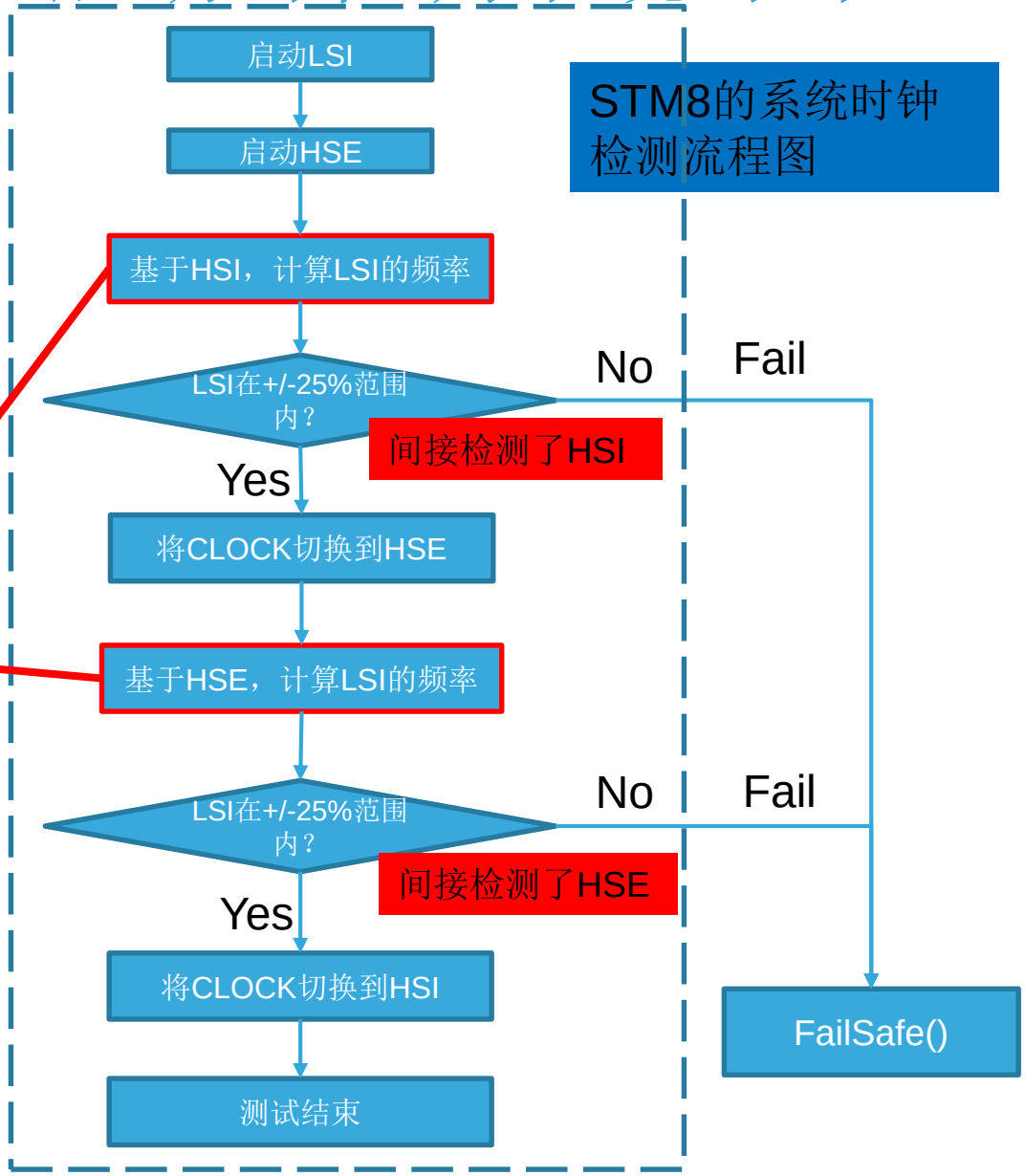


STM8启动时检测系统时钟

启动时检测时钟

- 以LSI为基准，分别检测HSE和HSI。
- 没有使用外部晶振的，可以跳过HSE的检测。

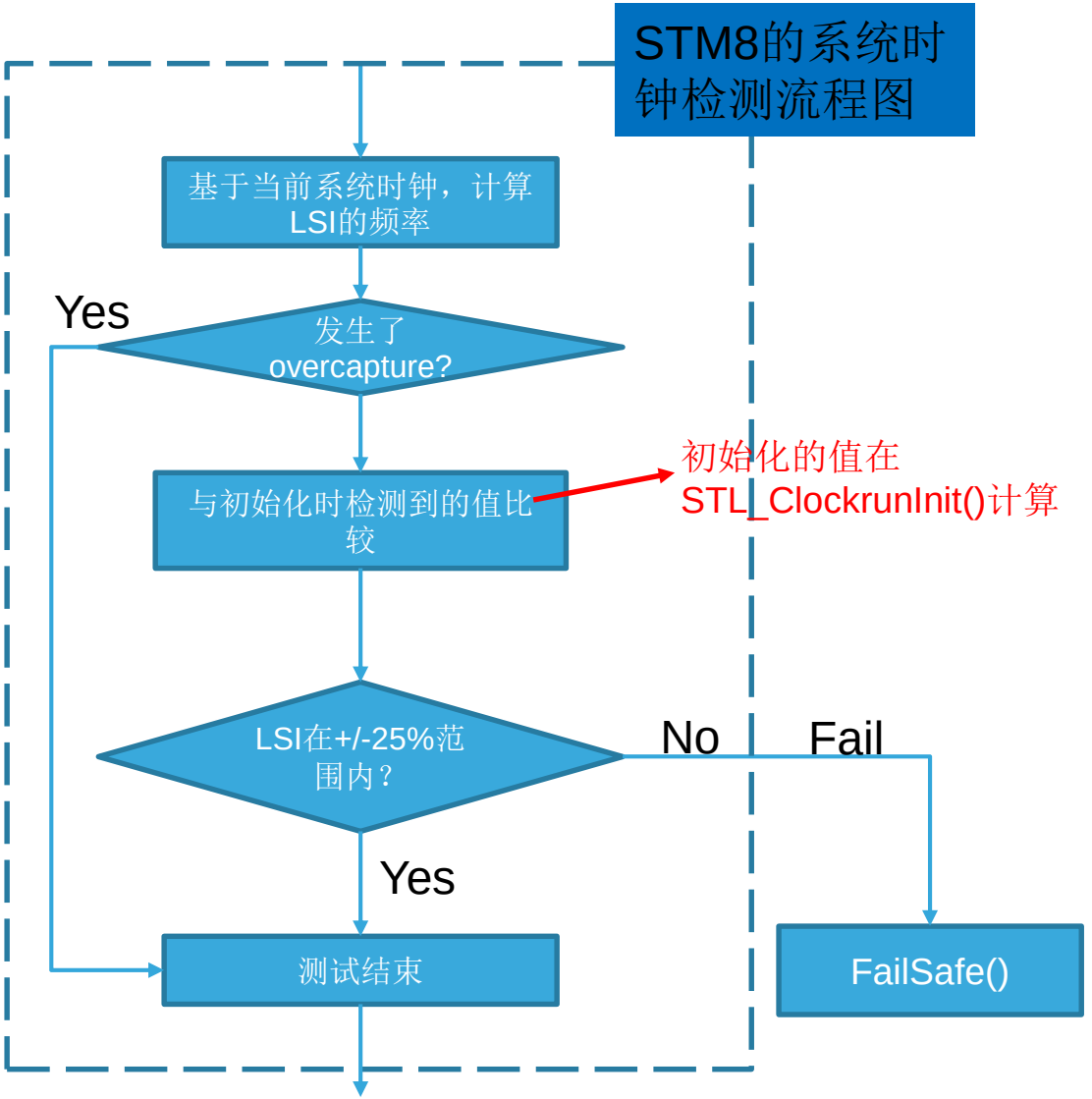




STM8运行时检测系统时钟

运行时监测时钟

- 以LSI为基准，监测当前的系统时钟。
- 参考值使用main函数中STL_ClockrunInit()中计算的值。



4.存储器

- 1. CPU
  - 1.1 CPU寄存器
  - 1.2 程序计数器
- 3.系统时钟
- 4.存储器
  - 4.1 非易失存储器---Flash
  - 4.2 易失存储器---RAM
  - 4.3 寻址
- 5.内部数据路径
  - 5.1 数据
  - 5.2 寻址

4.1非易失存储器
故障：所有的单比特错误

IEC60730 H表格中给出的可选检测措施：

- H.2.19.3.1 周期修改的检查和
- H.2.19.3.2 多重检查和
- H.2.19.8.2 带有一位冗余的字保护
- H.2.19.4.1 CRC

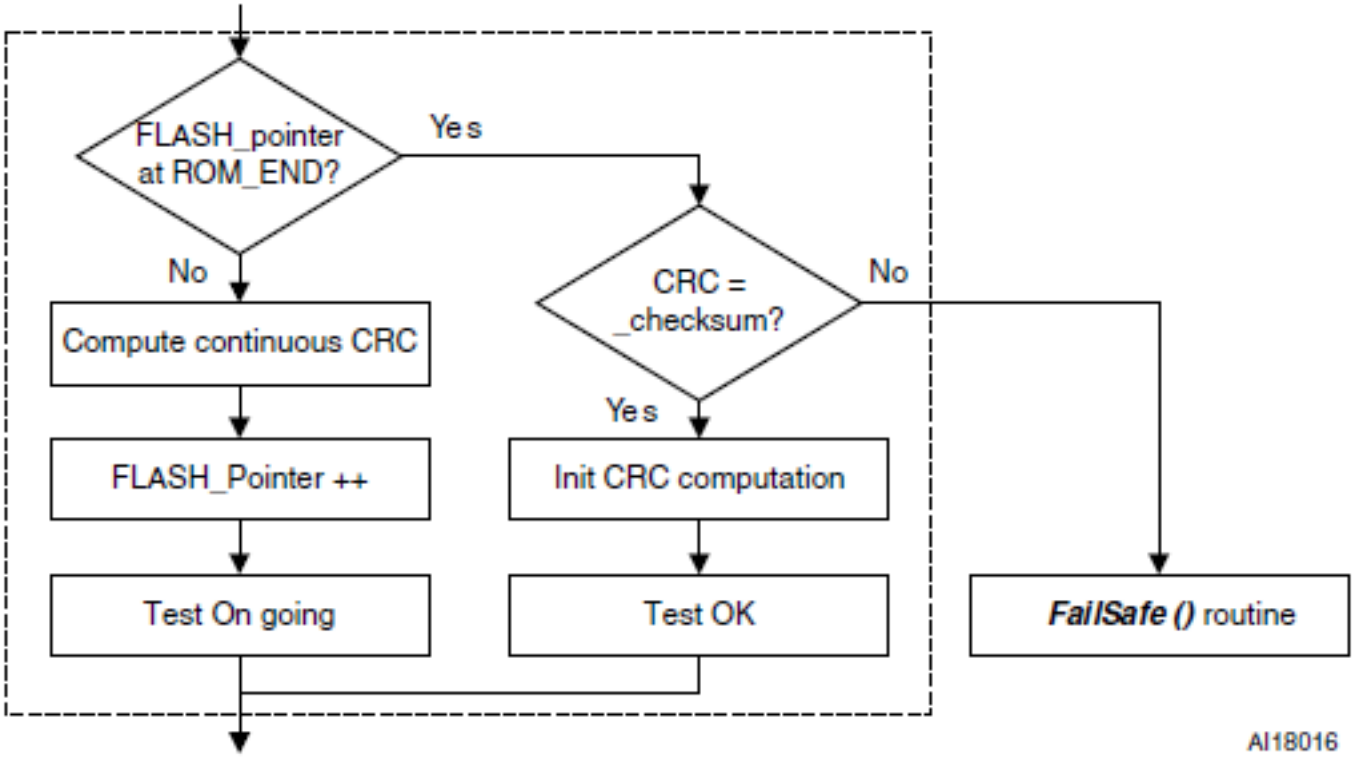
Flash故障检测

启动时对FLASH的检测方法：

- 编译时计算整个FLASH的CRC校验值，并存储在FLASH末尾位置
- 启动时，用同样的算法重新计算整个FLASH的CRC校验值（不包括前面存储在FLASH中的CRC校验值），并与存储在FLASH中的CRC校验值做比较

运行时对FLASH的检测方法：

- 对FLASH分块逐次计算出最终的CRC校验值
- 将最终结果与正确的CRC校验值做比较



Cyclic Redundancy Check(CRC)

- 一种对数据传输和存储中的错误检测技术
- 原始信息转换成二进制，被另一个固定的二进制数除（模二除法），得到的余数就是CRC校验码
- CRC码与原始信息一起发送，接受方用同样的方法对数据进行计算，然后与接受到的CRC码进行比较，不相同就认为数据在传输中被破坏
- 不论是原始信息转换成的二进制数，还是作为除数的固定的二进制数。都可以与一个系数仅为“0”或“1”的多项式对应。

  例如代码1010111对应多项式为$1*x^6+0*x^5+1*x^4+0*x^3+1*x^2+1*x+1*x^0$
- 作为除数的固定数对应的多项式称为生成多项式

  例如，CRC-8，生成多项式$x^8+x^5+x^4+1$

“*A painless guide TO CRC ERROR* DETECTION ALGORITHMS”
—— Ross N. Williams

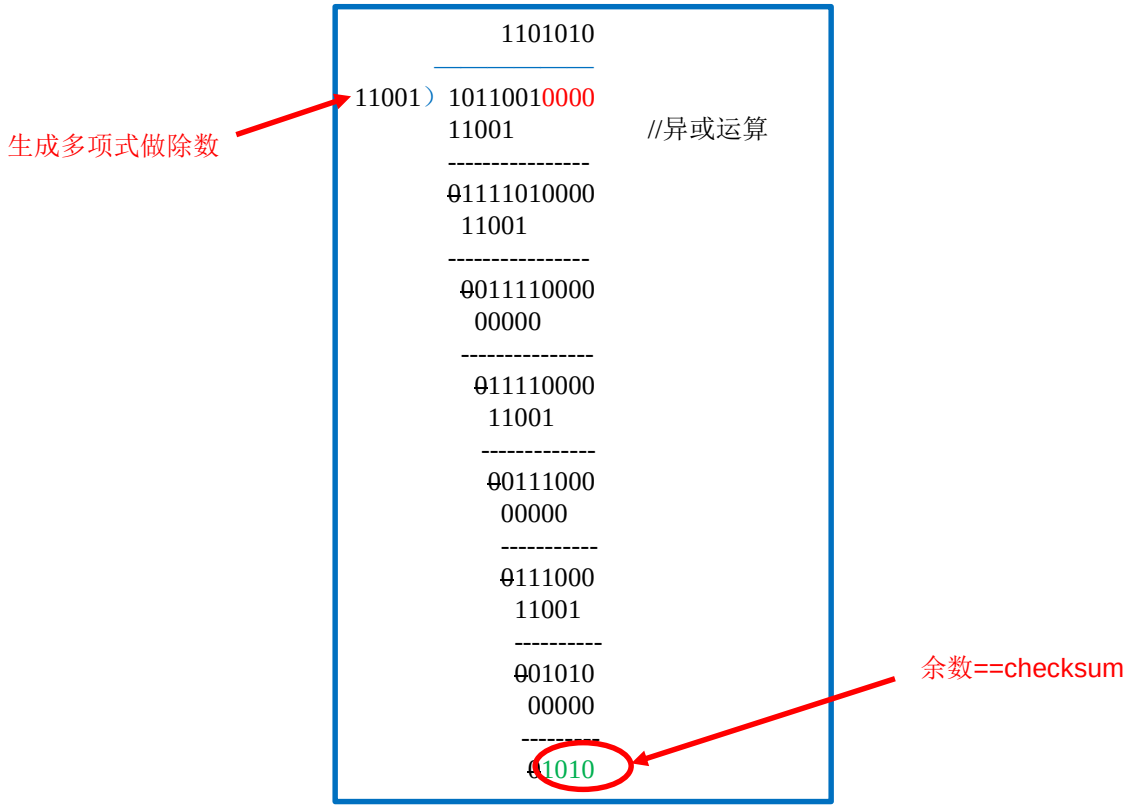
CRC校验码生成步骤

例如：原始信息为：1011001

生成多项式为：x^4+x^3+1（11001），最高次幂为R，R=4

计算CRC校验码步骤如下：

1. 将原始信息码左移R位（此时R为4）
2. 用生成多项式（11001）对移位后的信息码做模二除法，得到R位的余数，即为CRC校验码



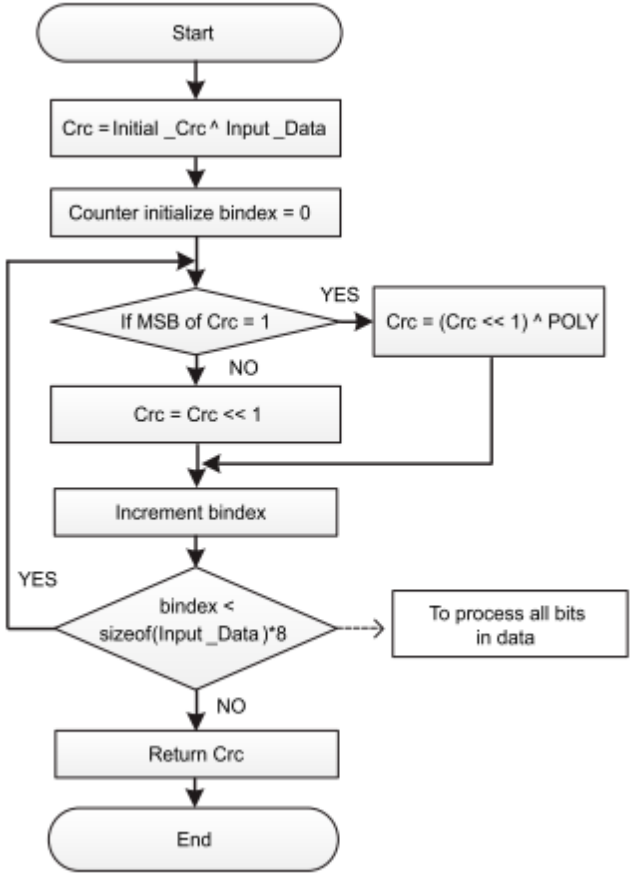
STM32全系列都有CRC外设

STM32的CRC外设

- 默认使用CRC32多项式：0x4C11DB7
  $X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X+1$

- 部分芯片支持可编程的多项式(STM32F3)
- 可对8/16/32bit位数据计算CRC值
- CRC初始值默认为0xFFFFFFFF。STM32F0、STM32F3可以修改初始值

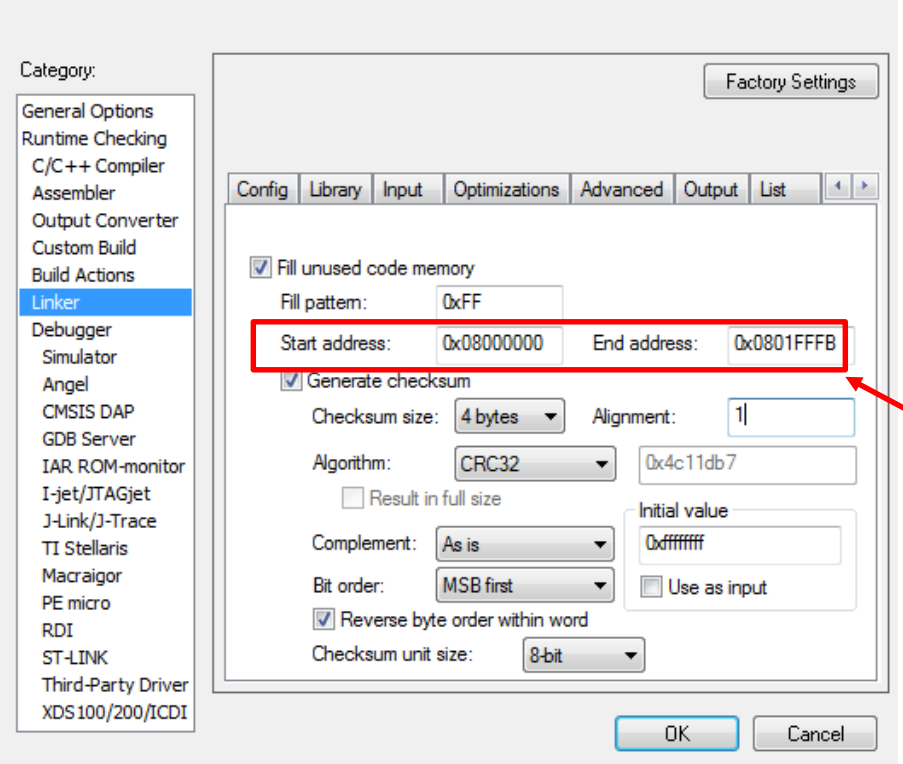
- 默认不对输入数据和输出数据进行位反转
  - 对输入数据的位反转操作可以设置为按字节/半字 /字为单元进行操作。例如输入数据为0x1A2B3C4D,
    每个字节内逐位反转，结果是0x58D43CB2
    每半字内逐位反转，结果是0xD458B23C
    每个字长内逐位反转，结果是0xB23CD458
  - 对输出数据的位反转。
    例如输出数据为0x11223344,反转后为0x22CC4488



可参考AN4187，Using the CRC peripheral in the STM32 family

IAR7.2中如何配置CRC——STM32

1.Project>Options>Linker>Checksum



需要计算checksum的FLASH地址范围

2.指定checksum在FLASH中的存储位置——修改linker文件：

eg:

place at end of ROM_region { ro section .checksum };

Keil中如何添加CRC值——STM32

- Keil 不像IAR可以很方便的支持在编译时计算CRC32，并存储在FLASH指定位置。
- 需要单独计算CRC的值，并手动添加代码将其放在FLASH末尾。
  - 修改*.sct文件

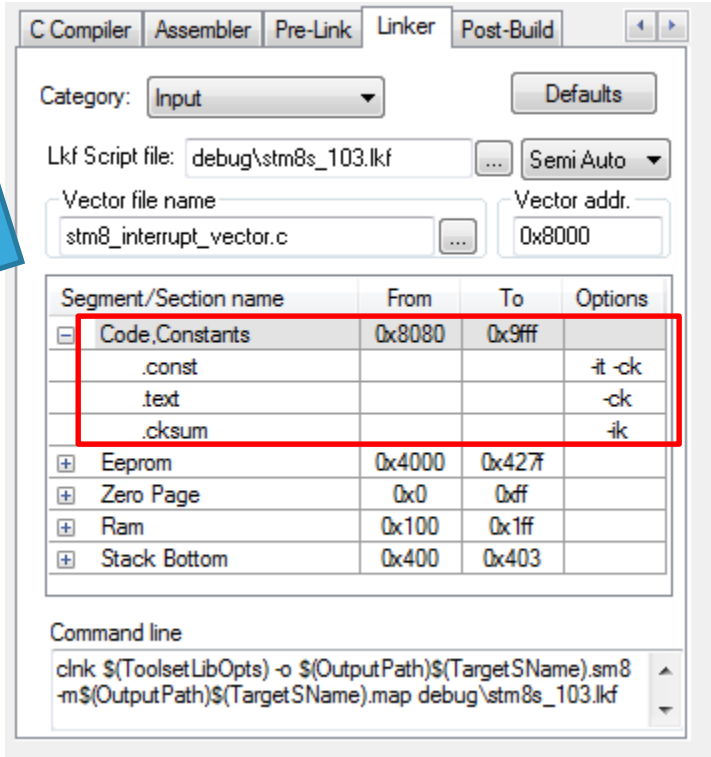
```
LR_IROM1 0x08000000 0x00010000  {    ; load region size_region
      ER_IROM1 0x08000000 0x00010000  {  ; load address = execution address
      *.o (RESET, +First)
      *(InRoot$$Sections)
      .ANY (+RO)
      *.o (CHECKSUM, +Last)
    }
```

  - 修改“startup_stm32xxxxxkeil.s”

```
;*******************************************************************************
; User Checksum - must be placed at the end of memory
;*******************************************************************************
                AREA    CHECKSUM, DATA, READONLY, ALIGN=6
                EXPORT  __Check_Sum

; Alignement here must correspond to the size of tested block at FLASH run time test (16 words ~ 64 bytes)!!!
                ALIGN

__Check_Sum     DCD     0x9D75                  ; Check sum computed externaly,

                END
```

Flash CRC校验配置——STM8

- 在不同的编译环境下，需要做相应配置以保证Flash CRC校验的正确执行：
  - Cosmic:
    - 所有需要进行校验的区域增加-ck标志
    - 通过-ik标志定义checksum段，计算出的checksum值将会存储在此。
  - IAR：通过工程选项的linker窗口的checksum选项卡下进行设置
  - Raisonance：需要在项目环境中进行校验范围和checksum存储位置的设置



4.2 易失存储器
故障：DC故障

IEC60730 H表格中给出的可选检测措施：

- H.2.19.6 周期静态存储器测试
  - H.2.19.6.1 方格（checkerboard）存储器测试
  - H.2.19.6.2 marching存储器测试
- H.2.19.8.2 带有一位冗余的字保护
  - 奇偶校验

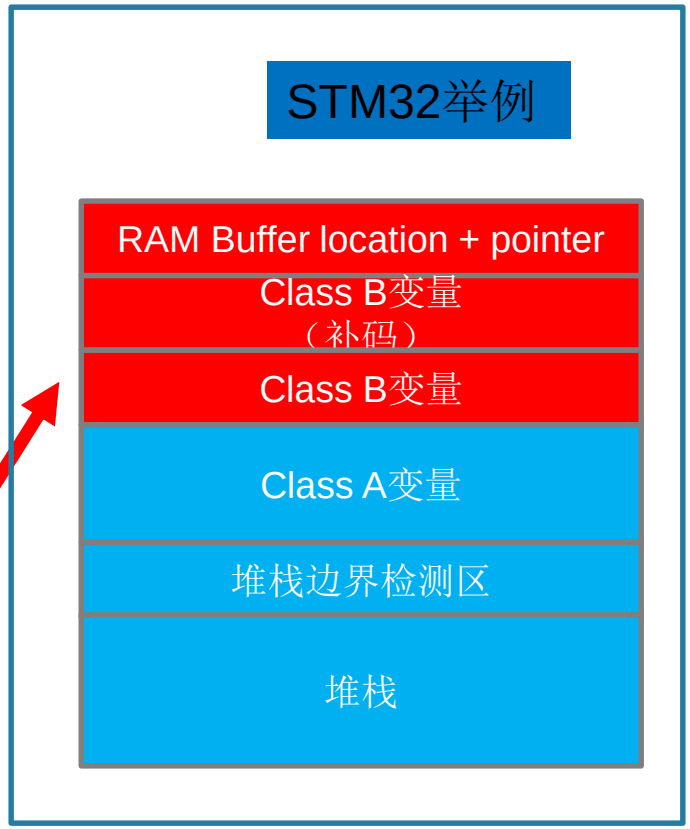
RAM故障检测

启动时：

- 对整个RAM区进行检测
- March C测试

运行时：

- 对存储ClassB变量的区域进行检测
- March C测试
- March X测试
- 关键安全信息（ClassB变量）做双反存储



March C/X算法

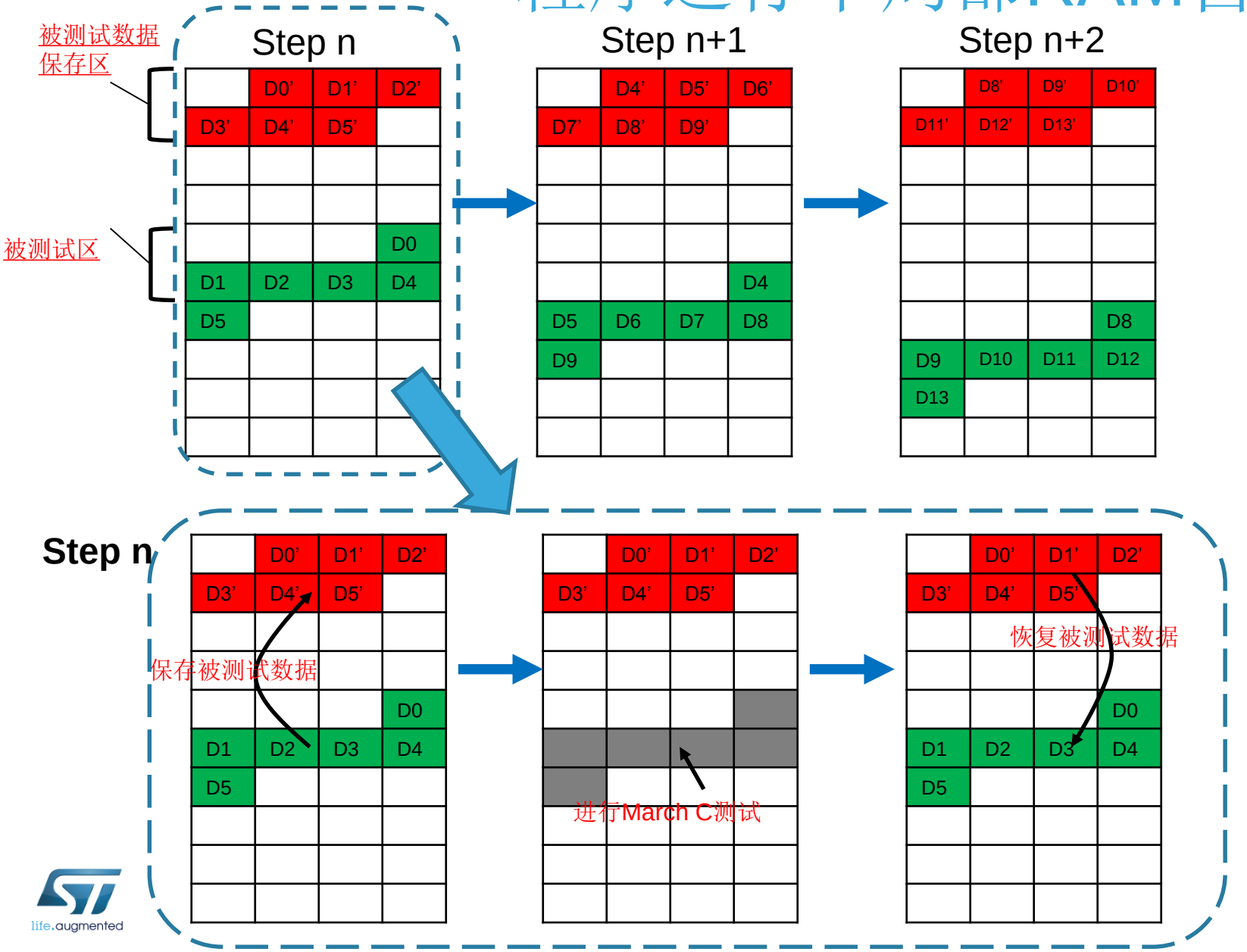
- March C算法分为6步执行
- March X算法比March C少中间的两个步骤

March X中没有第3，第4步

步骤	执行内容	操作的地址顺序
1	对所有测试单元写0	按地址增加的顺序
2	逐个检测每个单元是否为0，然后写为0xFF*	按地址增加的顺序
3	逐个检测每个单元是否为0xFF*，然后写为0	按地址增加的顺序
4	逐个检测每个单元是否为0，然后写为0xFF*	按地址递减的顺序
5	逐个检测每个单元是否为0xFF*，然后写为0	按地址递减的顺序
6	逐个检测每个单元是否为0	按地址递减的顺序

*：STM32中写入的是0xFFFFFFFF

程序运行中局部RAM自测



STM32的SRAM硬件奇偶校验功能

- 还可利用STM32的SRAM硬件奇偶校验功能
  - 可以使用STM32芯片的SRAM硬件奇偶校验功能，可以不用通过软件进行March测试
  - 对不支持硬件奇偶校验的SRAM采用软件March测试
  - 将硬件奇偶校验和软件March测试结合，甚至可以满足ClassC的需求
  - 使用硬件奇偶校验时，建议先对整个SRAM空间进行初始化，以避免出现错误

系列	产品线	SRAM硬件奇偶校验功能
STM32F0	All	支持
STM32F3	STM32F302/STM32F303/STM32F373/STM32F3X4/STM32F378/STM32F328	支持
STM32F1/STM32F2/STM32F4/STM32L0/STM32L1	All	不支持

如何增加新的ClassB变量

- 新定义的ClassB变量需要在stm32fxxx_STLclassBvar.h中按下面的格式进行声明
  - IAR：

```
__no_init EXTERN uint32_t MyClassBvar @ "CLASS_B_RAM";
__no_init EXTERN uint32_t MyClassBvarInv @ "CLASS_B_RAM_REV";
```

  - Keil:

```
EXTERN uint32_t MyClassBvar __attribute__((section("CLASS_B_RAM"), zero_init));
EXTERN uint32_t MyClassBvarInv __attribute__((section("CLASS_B_RAM_REV"), zero_init));
```

- 根据用户应用程序的需用，改变ClassB变量区的大小
  - IAR：可以通过修改linker文件来改变ClassB变量区的大小
  - Keil：需要在stm32fxxx_STLparam.h文件中进行修改

```
/* Constants necessary for Transparent March tests */
#define CLASS_B_START ((uint32_t *) 0x20000040)
#define CLASS_B_END   ((uint32_t *) 0x20000100)
```

4.3寻址错误（与非易失和易失存储器相关的）故障：Stuck-At

IEC60730 H表格中给出的可选检测措施：

- H.2.19.8.2 带有地址的一位奇偶的字保护

5 内部数据路径

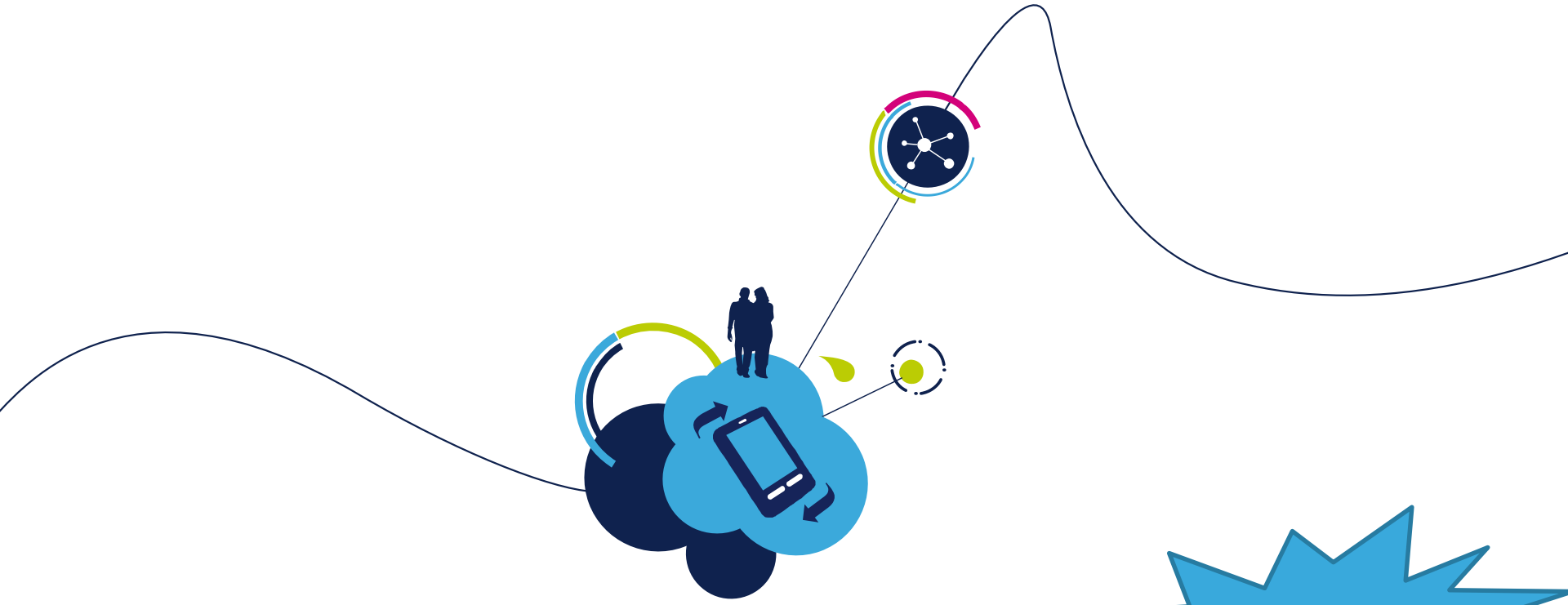
- 1. CPU
  - 1.1 CPU寄存器
  - 1.2 程序计数器
- 3.系统时钟
- 4.存储器
  - 4.1 非易失存储器---Flash
  - 4.2 易失存储器---RAM
  - 4.3 寻址
- 5.内部数据路径
  - 5.1 数据
  - 5.2 寻址

5.1 数据错误
5.2 寻址错误
故障：Stuck-At

IEC60730 H表格中给出的可选检测措施：

- H.2.19.8.2 带有地址的一位奇偶的字保护

仅针对使用了外部存储器的微控制器，单片微控制器不做要求



需要用户
自己开发

故障&措施具体介绍
——应用相关故障的检测

2. 中断
故障：没有中断或者中断太频繁

IEC60730 H表格中给出的可选检测措施：

- H.2.16.5 功能检测
- H.2.18.10.4 独立的时间片监测

可以采用的方法：
每次中断中计数器自加；周期性地、以固定时间间隔去检查计数器的值（该时间间隔由独立定时器驱动），从而检查是否在固定时间内产生了应用需要的中断次数

6.外部通信故障——数据和地址

6.1 数据
故障： 汉明距离3
6.2 寻址
故障： 错误的地址

例如：
1011101与1001001之间的汉明距离是2。
2143896与2233796之间的汉明距离是3。

IEC60730 H表格中给出的可选检测措施：

H.2.19.8.1 带有多位冗余的字保护
H.2.19.4.1 CRC-单字
H.2.18.2.2 传输器冗余
H.2.18.14 预定测试

部分外设提供CRC计算功能

可采样的方法：在数据传输中增加用于校验的冗余信息，奇偶校验，CRC校验值等，根据实际应用选择

带硬件CRC的外设

外设	硬件CRC功能	CRC多项式
SPI	硬件CRC	CRC8/CRC16
I2C	Packet Error Checking	CRC8： X^8+X^2+X+1
USB	CRC	
SDIO	CRC7	CRC7：X^7+X^3+1
Ethernet	CRC	CRC32： $X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$

6.3 时序
故障：错误的时序

IEC60730 H表格中给出的可选检测措施：

- H.2.18.10.2 逻辑检测
- H.2.18.10.4 独立的时间片监测
- H.2.18.18 预定传输

可以采用的方法：
每个通信事件中计数器自加；周期性地、以固定时间间隔去检查计数器的值（该时间间隔由独立定时器驱动），从而检查是否在固定时间内产生了应用需要的通信事件次数

7.1 数字I/O
故障：H27中定义的错误

例如:开路，对电源短路，对地短路，引脚之间短路等

IEC60730 H表格中给出的可选检测措施：

- H.2.18.13 似真检查

可以采用的方法：
必须能检测数字I/O口的任何故障。可能需要与应用中的其他部件的工作情况进行核对。（比如，加热管的开关控制引脚切换后（接通或者断开），温度传感器检测到的温度是否发生了变化）

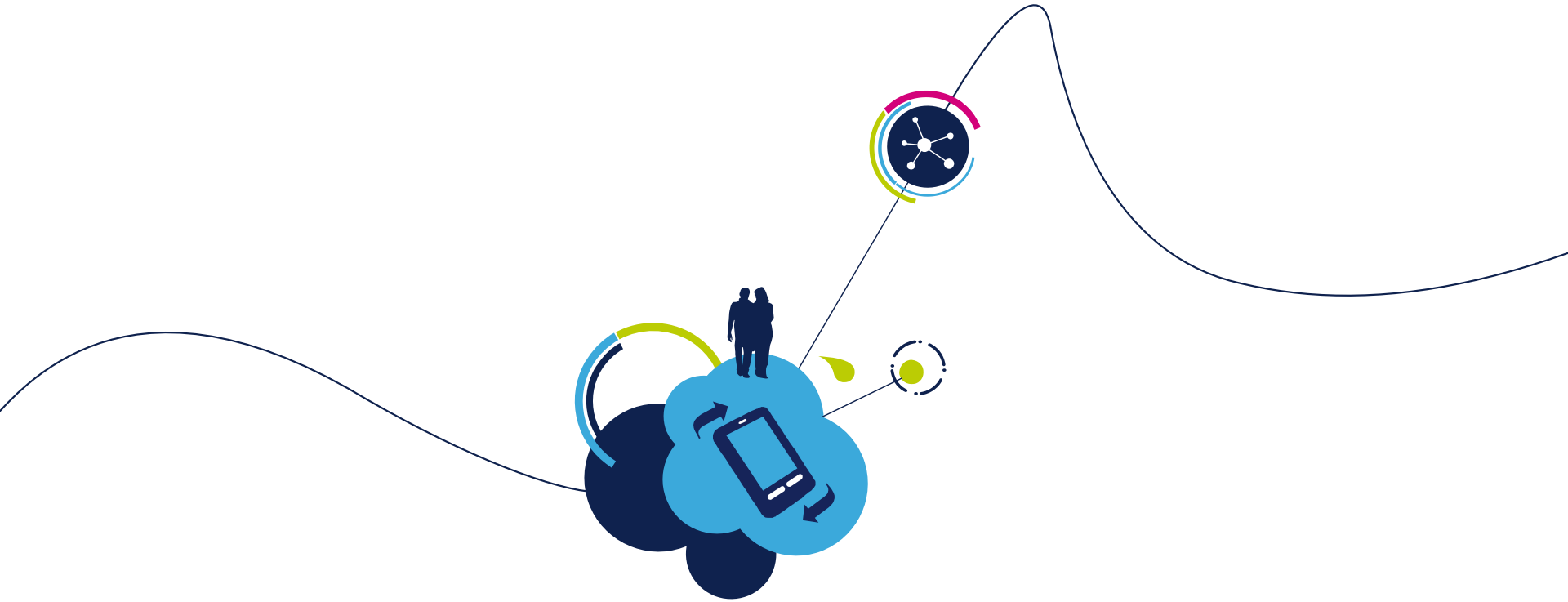
7.2 模拟I/O
故障：H27中定义的错误/寻址错误

例如:开路，对电源短路，对地短路，引脚之间短路等

IEC60730 H表格中给出的可选检测措施：

- H.2.18.13 似真检查

可以采用的方法：
使用到的pin脚应该按照一定的时间间隙进行检测。
空闲的引脚可以用来检测应用中用到的模拟端口。
内部参考源也需要进行检测。



ST ClassB库的结构和应用

life.augmented

ST ClassB软件库

ST ClassB软件包：

- 每个产品系列都有一个软件包，如IEC60355_STM32F2_V1.0.0_setup.exe。
- 安装后的软件包包括以下内容
  - 上述MCU相关故障的检测程序源码。
  - 支持不同的编译环境的参考项目
    - STM8的软件库支持Cosmic, IAR, Raisonance三种编译环境
    - STM32的软件库支持IAR, Keil两种编译环境
    - 可供参考的Class B软件结构，包括修改过的向量表，linker文件等。
    - 可以直接重用
  - 版本说明
  - VDE证书

ST Class B软件库版本信息

	系列	版本	Certified by VDE
STM8	STM8S/A	1.1.0	Certified
	STM8L/AL Medium density	1.1.0	Certified
	STM8L/AL Low density	1.1.0	Certified
STM32	STM32F0	1.0.0	Certified
	STM32F1	2.1.1	Certified
	STM32F2	1.0.0	Derivative
	STM32F3	1.0.0	Certified
	STM32F4	1.0.0	Derivative
	STM32L1	1.0.0	Derivative
	STM32L0		Not available

参考文档及软件

AN3181- Guideline for obtaining IEC 60335 in STM8 application：

http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/CD00268777.pdf?s_searchtype=keyword

STM8 ClassB软件包下载：
http://www.st.com/web/en/catalog/tools/FM147/CL1794/SC1807/SS1754/PF258214?s_searchtype=keyword#

AN3307-Guidelines for obtaining IEC60335 Class B certification in STM32：

http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/CD00290100.pdf?s_searchtype=keyword

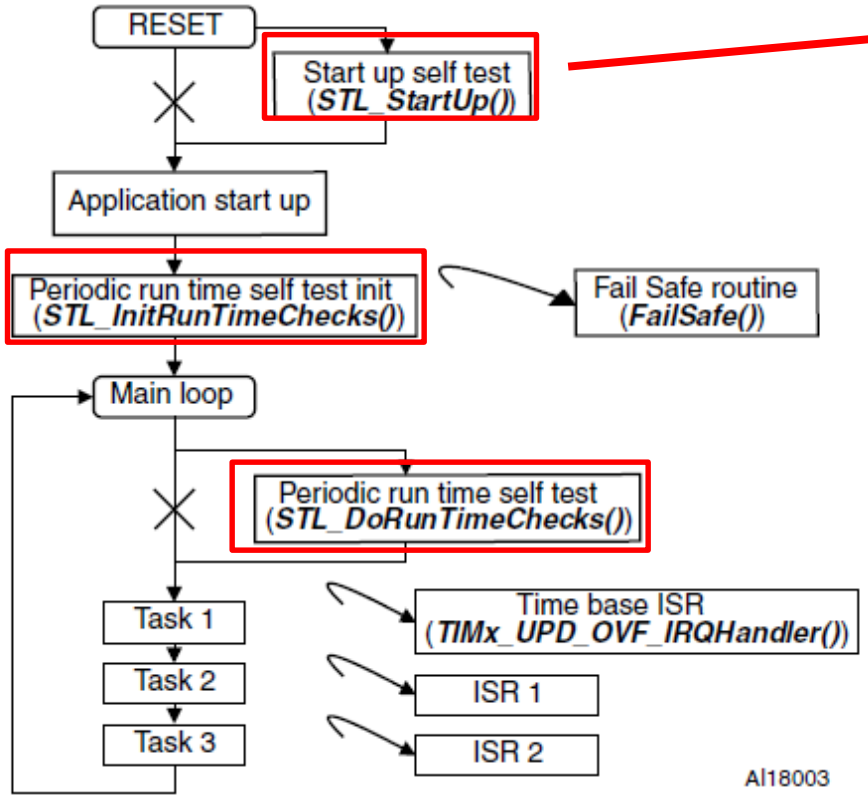
与用户程序的整合(1/3)

用户需要做的事情：

- 在执行用户程序之前，先执行STL_StartUp函数（启动自检）
- 设置WWDG和IWDG，防止其在程序正常运行时复位
- 设置启动和运行时的RAM和FLASH检测范围
  - CRC校验的范围，checksum在Flash中存储的位置
  - ClassB变量的存储地址范围
  - 堆栈边界检测区的位置
- 对检测到的故障进行处理
- 根据具体的应用，增加用户相关的故障检测内内容
- 根据具体应用定义程序运行时自检的频率
- 在进入主循环前调用STL_InitRunTimeChecks()
- 主循环中调用STL_DoRunTimeChecks()

与用户程序的整合(2/3)

- 芯片复位后，在执行初始化工作之前，必须先调用STL_StartUp函数进行启动时的自检。

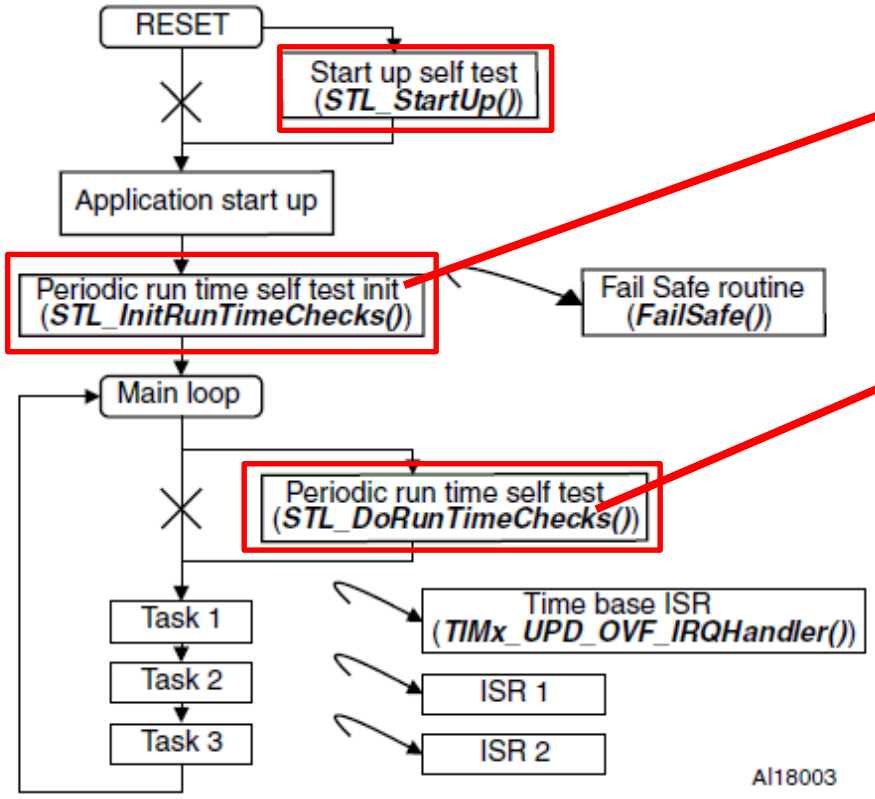


- STM8
  - IAR：修改csstartup.s文件
  - Raisonance: 修改startup.asm文件
  - Cosmic: 修改stm8_interrupt_vector.c文件
- STM32，强制将复位向量跳转执行STL_StartUp函数。
  - IAR： STL_StartUp执行完后跳回执行__iar_program_start()
  - Keil： STL_StartUp执行完后跳回执行Reset_Handler()

与用户程序的整合(3/3)

- 在main函数中加入：

  STL_InitRunTimeChecks();
  STL_DoRunTimeChecks();




```
#ifdef STL_INCL_RUN
  STL_InitRunTimeChecks();
#endif
```

```
while (1)
{
  #ifdef STL_INCL_RUN
    STL_DoRunTimeChecks();
  #else
    refresh_iwdog();  /* For demo purposes, if hardw
    refresh_wwdog(0x7f, 0x7f);
  #endif /* STL_INCL_RUN */

  #ifdef STL_VERBOSE
    #if defined(EVAL_BOARD_LCD)
      LCD_SetCursorPos(LCD_LINE1+4);
      sprintf(s," df=%02u%%", FreqDifDisplay);
      LCD_Print((u8 *)s);
    #endif /* EVAL_BOARD_LCD */
  #endif /* STL_VERBOSE */
}
```

Debug注意事项

- 在FailSafe()函数中给IWDG喂狗，避免重启。
  - IWDG使用LSI时钟，在debug时可以继续工作（STM32通过DBG_IWDG_STOP标志位设置,STM8通过DM_CR1中的WDGOFF位设置）
- 停止对FLASH进行CRC校验，以避免设置断点后出现CRC校验错误导致系统无法正确运行
- 关闭WWDG，避免系统重启
  - WWDG使用系统时钟，在debug时可以继续工作（STM32通过DBG_WWDG_STOP标志位设置， STM8通过DM_CR1中的WDGOFF位设置）
- 关闭程序控制流监测，当增加或减少了测试功能模块时。

Verbose 诊断模式

- 在Verbose诊断模式下，可以通过UART的Tx pin脚输出文字信息。
  - 115200Bd, no parity, 8-bit data,1 stop bit
- 在这种模式会占用很多的代码空间
- 用户可以在stm8x_stl_param.h中选择verbose模式的类型：
  - 启动时
  - 程序运行时
  - 出现错误时

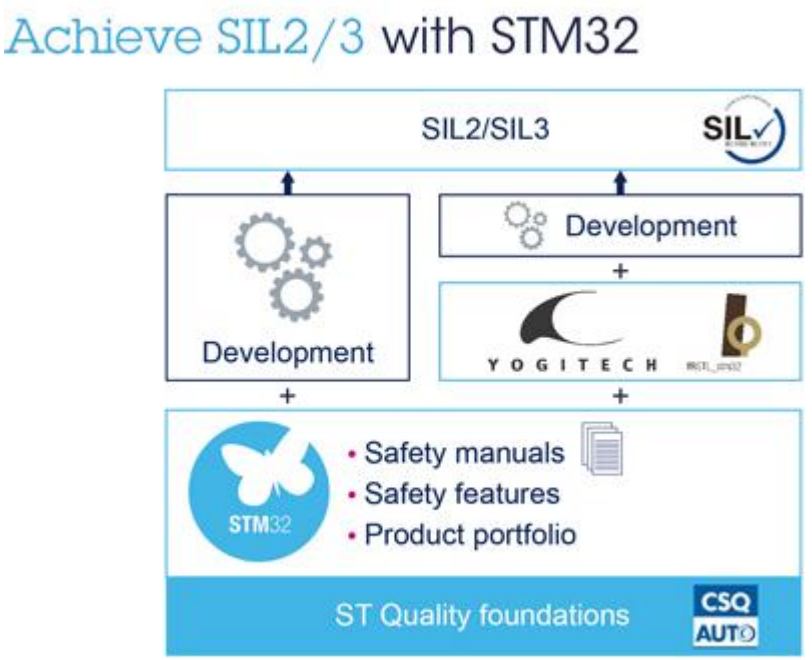
Thanks!

什么是SIL

- SIL（Safety Integrity Level）-安全完整性等级。
- 执行的基础标准是IEC 61508。
- SIL认证一共分为4个等级，SIL1、SIL2、SIL3、SIL4，包括对产品和对系统两个层次。 其中，以SIL4的要求最高。

STM32的SIL安全设计软件包

- 支持STM32F0,/F1/F2/F3/F4/L1
- 帮助STM32用户到达SIL2/3标准
- 用户可以参照免费的STM32安全手册建立自己的STL
- 也可以使用YOGITECH特许的经过TUV认证的fRSTL_STM32库文件



STM32_SafeSIL