前言

这篇迁移指南旨在帮助您分析从现有的SXX32F103器件移植到MH2103A器件所需的步骤。本文档收集了最重要的信息，并列出了需要注意的重要事项。

要将应用程序从SXX32F103系列移植到MH2103A系列，用户需要分析硬件移植、外设移植和固件移植。

支持型号列表：

支持型号	MH2103Axxxx

目录

MH2103A系列微控制器基本兼容SXX32F103系列,同时强化许多功能,有些许地方与SXX32F103不同，详述于本文档。

1. 快速替换 SXX32F103 芯片

●步骤一：比对外设规格、Flash容量、SRAM容量等，解焊SXX32F103，换成MH2103A对应型号

●步骤二：使用ISP或KEIL,下载SXX32F103 HEX文件或BIN文件。

●步骤三：如果有需要，下载SXX32F103 HEX文件或BIN文件以外的资料或进行系统校正。

●步骤四：查看程序能否正常运行。

●步骤五：其他问题快速排查请参考2外设使用区别。

●步骤六：如果经过上述步骤后程序仍无法正常运行，请参考本文件其他章节，或连络代理商及MH支持人员协助解决。

2. MH2103A 外设使用差别

2.1 系统功能

BOOT 管脚使用差异

●描述：

当 BOOT0,BOOT1 悬空时

SXX32F103：识别为低，但是抗干扰能力较弱，容易被外界环境干扰

MH2103A：为浮空状态，建议外部固定为高/低

自举模式选择引脚		自举模式	别名使用
BOOT1	BOOT0		
X	0	用户 Flash	选择用户 Flash 作为自举空间
0	1	系统存储器	选择系统存储器作为自举空间
1	1	嵌入式 SRAM	选择嵌入式 SRAM 作为自举空间

DEBUG 状态下，将 JTAG 关断或者将 JTRST 释放，现象差异

●描述：

情况 1：DEBUG 状态下，将 SW 使能，JTAG 关断，如下图使用

```
GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable,ENABLE);
```

SXX32F103：JTAG 关断，正常使用 SW 进行 DEBUG 仿真

MH2103A：DEBUG 仿真时程序复位

情况 2：DEBUG 状态下，将 SW、JTAG 使能，JREST 释放，如下图使用

```
GPIO_PinRemapConfig(GPIO_Remap_SWJ_NoJTRST,ENABLE);
```

SXX32F103：正常使用 SW/JTAG 进行 DEBUG 仿真

MH2103A：DEBUG 仿真时程序复位

某些烧录器下载失败问题

●描述 1：

不支持某些 SXX32F103 专用烧录器下载

因为 MH2103A 和 SXX32F103 ARM M3 Core 版本、SW、JTAG IDCODE 不同

	SXX32F103	MH2103A
Core ID	0x411FC231	0x412FC230
SW IDCODE	0x1BA01477	0x2BA01477
JTAG IDCODE	0x3BA00477	0x4BA00477

当使用某些 SXX32F103 专用烧录器时，会判断 Core ID 和 SW IDCODE、JTAG IDCODE 等信息，故不支持某些 SXX32F103 专用烧录器

●解决方法：

使用不判断 Core ID 和 SW IDCODE、JTAG IDCODE 的烧录器

●描述 2：

对于某些烧录器在 RESET 拉低的情况下进行 SW/JTAG 交互，是不支持的

因为 MH2103A 当 RESET 拉低时，SW 和 JTAG 是无法使用的

Eg：WizPro200ST8 编程器

●解决方法：

1. 配置烧录器，在烧录时将 RESET 拉高
2. 悬空芯片 NRST 管脚，不和烧录器相连

●描述 3：

对于某些烧录器在编程时，烧录失败并提示擦除错误。

因为某些烧录器擦除使用的是全擦，并且超时时间是基于 SXX32F103 的全擦时间定义的；

而 MH2103A 全擦时间较长，从而引发擦除超时并导致烧录失败

Eg：SmartPRO T9000-PLUS 编程器

●解决方法：

1. 配置烧录器，擦除选择 Page 擦
2. 换用其他烧录器进行烧录

●描述 4：

MCU 处于读保护/写保护状态下，烧录器通过 SW/JTAG 解除读写保护，再下载程序失败

因为 MH2103A 通过 SW/JTAG 发起的系统复位不会重新加载选项字节

●解决方法：

通过其他系统复位、外部 RESET 复位、上下电复位重新加载选项字节后，再进行下载

第三方某些 ISP 下载失败问题

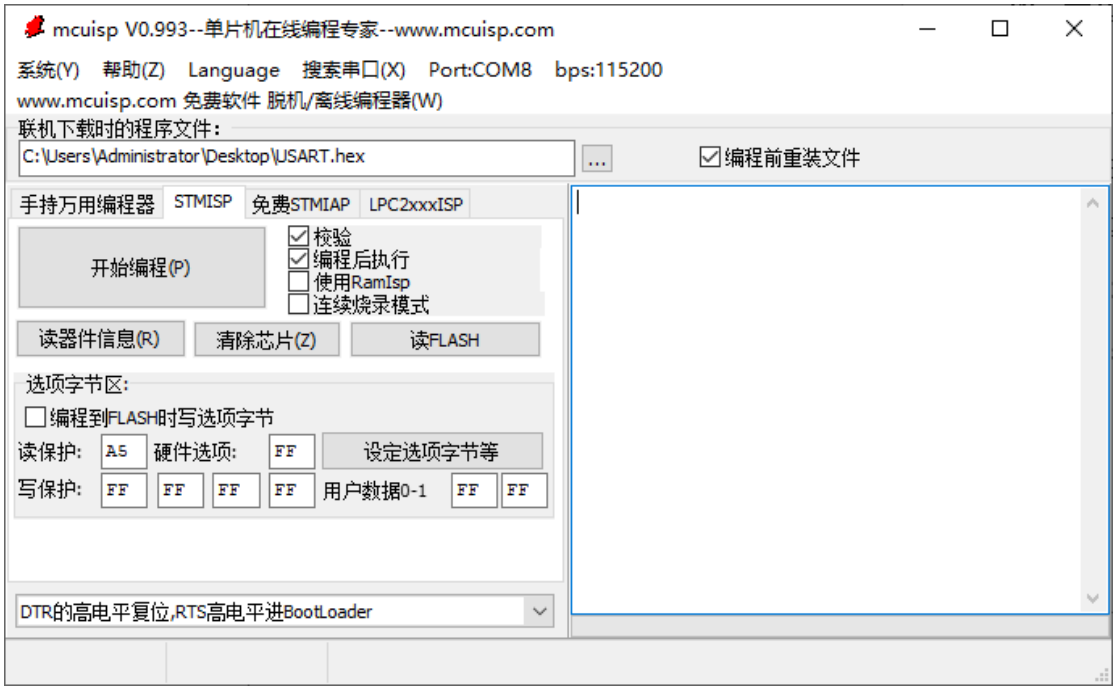
●描述：

对于某些 ISP 在编程时，烧录失败并提示擦除错误。

是因为某些 ISP 擦除使用的是全擦，并且超时时间是基于 SXX32F103 的全擦时间定义的；

而 MH2103A 全擦时间较长，从而引发擦除超时并导致烧录失败，故不支持此类 ISP

Eg：mcuisp V0.993



非 32bit 对齐访问 APB 总线时，现象差异

●描述：

访问 APB 总线时，必须 32bit 对齐，否则无法访问

Eg：

当 ADC 转换完成后，若按照非 32bit 对齐访问 ADC DR 寄存器

SXX32F103：可获取正确值

MH2103A：无法获取正确值，获取的值均为 0

●解决方法：

访问 APB 总线的寄存器时，按照 32bit 对齐，均可正常访问

中断控制器差异

SXX32F103：

1. 最多支持 60 个可屏蔽中断通道

2. 16 个可编程的优先等级(使用了 4 位中断优先级)

MH2103A：

1. 最多支持 71 个可屏蔽中断通道

2. 8 个可编程的优先等级(使用了 3 位中断优先级)

系统复位不会重新加载 BOOT 管脚

●描述：

SXX32F103：系统复位后，会重新加载 BOOT 管脚，进入相应的启动模式

MH2103A：系统复位后，不会重新加载 BOOT 管脚，保持上一次 BOOT 管脚的状态，进入相应的启动模式

系统复位涉及到以下几种情况：

软件复位、IWDG 复位、WWDG 复位、通过 SW/JTAG 发起的复位、nRST_STANDBY/nRST_STOP 为 0 时进 STANDBY/STOP 产生的复位、ISP 使能/关闭读写保护产生的复位

●解决方法：

可通过外部 RESET 复位、上下电复位重新加载 BOOT 管脚

2.2 ADC

ADC 配置连续触发后，关闭 ADON 或者进行 ADC 软复位，ADC 转化差异

●描述：

SXX32F103：关闭 ADON 或者进行 ADC 软复位，ADC 转换停止

MH2103A：

1. 关闭 ADON，ADC 连续转化未停止

2. 进行 ADC 软复位， ADC 转换停止，再次启动转换，前几个量化值异常

●解决方法：

1. 软件关 ADON 之前，将连续转换配置为单次转换，等待上一次采样周期个 ADC_CLK

2. 连续转化过程中，进行 ADC 软复位后，再次启动需要等待上一次采样周期个 ADC_CLK

ADC 连续两次使能 ADON 再进行外部事件的软件触发或者连续两次使能外部事件的软件触发，现象差异

●描述：

情况 1：当使能 ADC 后，再次使能 ADC，并且在 EOC 置位前，进行外部事件的软件触发

Eg:

```
ADC_Cmd(ADC1, ENABLE);
ADC_Cmd(ADC1, ENABLE);
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

注意：

当 ADC 使能后，再次使能 ADC，SXX32F103 和 MH2103A 均会触发一次 ADC 转化。

情况 2. 当有 2 次连续外部事件的软件触发时

Eg:

```
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

SXX32F103：EOC 正常置位，会忽略一次触发

MH2103A：会导致转化异常，EOC 不置位

●解决方法：

1. 使能 ADC 后，再次使能 ADC，等待 EOC 置位，读取量化值

Eg：

```
ADC_Cmd(ADC1, ENABLE);
ADC_Cmd(ADC1, ENABLE);
while(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC) == RESET);
ADC_GetConversionValue(ADC1);
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

2. 外部事件的软件触发，等待 EOC 置位，读取量化值

Eg：

```
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
while(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC) == RESET);
ADC_GetConversionValue(ADC1);
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
while(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC) == RESET);
ADC_GetConversionValue(ADC1);
```

双 ADC 不同模式差异

●描述：

双 ADC 不同模式下

1. 同步注入、交替触发模式下

   SXX32F103：注入组通道/规则组通道被触发后，均正常转化

   MH2103A：注入组通道被触发后，正常转化；规则组通道被触发后，不进行转化

2. 同步规则、快速交叉、慢速交叉模式下

   SXX32F103：注入组通道/规则组通道被触发后，均正常转化

   MH2103A：规则组通道被触发后，正常转化；注入组通道被触发后，不进行转化

2.3 TIM

通用定时器 TIM2-TIM5 的 Channel3 差异

●描述：

SXX32F103：通用定时器 TIM2-TIM5 的 Channel3 支持输入、输出功能

MH2103A：通用定时器 TIM2-TIM5 的 Channel3 只支持输入功能

●解决方法：

选择 TIM2-TIM5 的其他通道,或者选择其他 TIM 的通道进行输出使用

TIM2 重映射差异

●描述：

当 TIM2_REMAP[1:0] = 01

SXX32F103: TIM2_CH1_ETH - PA15;TIM2_CH2 - PB3

MH2103A: TIM2_CH1_ETH - PA0;TIM2_CH2 - PA1

当 TIM2_REMAP[1:0] = 10

SXX32F103: TIM2_CH1_ETH - PA0;TIM2_CH2 - PA1

MH2103A: TIM2_CH1_ETH - PA15;TIM2_CH2 - PB3

●解决方法：

使用 TIM2 重映射，注意 IO 配置的修改

TIM 使用外部信号刹车，连续两次刹车的时间间隔小于一个 TIM 时钟周期时，现象差异

●描述：

SXX32F103：刹车标志正常置位，正常清除

MH2103A：刹车标志正常置位，BIF 清除不掉、刹车中断清除不掉

●解决方法：

1. 软件配置刹车信号为普通输入 IO 中断，不使用 BIF 和刹车中断
2. 在 BIF 置位或者刹车中断中，进行软复位

CEN 被清除(单脉冲模式下，发生更新事件/ 软件写 0)，修改 CNT 值，再次使能 CEN，现象差异

●描述：

情况 1：

使能单脉冲模式，开启更新中断

```
TIM_SelectOnePulseMode(TIMx,TIM_OPMode_Single);
TIM_ITConfig(TIMx, TIM_IT_Update, ENABLE);
TIM_Cmd(TIMx, ENABLE);
```

在更新中断中配置 CNT，使能 CEN

```
if (TIM_GetITStatus(TIMx, TIM_IT_Update) != RESET)
{
    TIM_ClearITPendingBit(TIMx, TIM_IT_Update);
    TIM_SetCounter(TIMx,NewCnt);
    TIM_Cmd(TIMx, ENABLE);
}
```

注意：

单脉冲模式下，SXX32F103 和 MH2103A 的 TIM 计数器自动地在产生下一个更新事件 UEV 时停止。

情况 2：

在中断中软件关闭 TIM，修改 CNT 值，软件再使能 TIM

Eg:

```
void TIM3_IRQHandler(void)
{
    if (TIM_GetITStatus(TIMx, TIM_IT_Update) != RESET)
    {
        TIM_Cmd(TIMx, DISABLE);
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
        TIM_SetCounter(TIMx,1999);
        TIM_Cmd(TIMx, ENABLE);
    }
}
```

在上述两种情况下

SXX32F103：在更新中断重新使能 CEN 后，计数器会从新配置的 NewCnt 进行计数

MH2103A：在更新中断重新使能 CEN 后，计数器会不从新配置的 NewCnt 进行计数，会重新加载 ARR 的数值进行计数

●解决方法：

在修改 CNT 时，需要确保 CEN 是处于使能状态，否则配置 CNT 无效

2.4 CAN

CAN 发送时间戳时，时间戳填充报文的位置差异

●描述：

CAN 配置时间触发通讯模式，在最后 2 个数据字节发送时间戳时

SXX32F103：时间戳[7:0]在数据包的 8Byte，时间戳[15:8]在数据包的 7Byte

MH2103A：时间戳[7:0]在数据包的 7Byte，时间戳[15:8]在数据包的 8Byte

●解决方法：

其他 CAN 节点使用接受到的时间戳时，软件进行翻转

CAN 休眠模式下配置过滤器差异

●描述：

SXX32F103：CAN 过滤器可以在 CAN SLEEP 模式下配置

MH2103A：CAN 过滤器可以在 CAN SLEEP 模式下不可配置

●解决方法：

配置 CAN 过滤器时，需要确保 CAN 处于非休眠模式

如下图顺序调用库函数即可:

```
CAN_Init(CAN1, &CAN_InitStructure);
CAN_FilterInit(&CAN_FilterInitStructure);
```

2.5 FLASH

FLASH 写保护第一块后，第一次擦除其他 Page，现象差异

●描述：

SXX32F103：写保护第一块后，擦除其他 Page，可正常擦除

MH2103A：写保护第一块后，第一次擦除其他 Page，擦除失败，并上报 WRPRTERR 写保护错误

●解决方法：

在配置 Page 擦除之前，先配置一次 Strt，如下图所示

```
status = FLASH_WaitForLastBank1Operation(EraseTimeout);
if(status == FLASH_COMPLETE)
{
  FLASH->CR = CR_STRT_Set;

  FLASH->CR|= CR_PER_Set;
  FLASH->AR = Page_Address;
  FLASH->CR|= CR_STRT_Set;
```

2.6 USART

USART 智能卡模式时钟输出差异

●描述：

SXX32F103：USART 智能卡模式时钟输出，无需配置 USART TE(发送使能)/RE(接收使能)

MH2103A：USART 智能卡模式时钟输出，需要配置 USART TE(发送使能)/RE(接收使能)，否则无时钟输出

●解决方法：

在初始化 USART 时，将 USART TE(发送使能)/RE(接收使能) 使能

2.6 SPI/IIS

IIS 在主接收模式、并且处于 PCM 标准模式下时，关闭 I2SE 现象差异

●描述：

SXX32F103：可通过配置 I2SE，停止主机输出时钟

MH2103A：不可通过配置 I2SE，停止主机输出时钟

●解决方法：

1. 通过配置 I2SMOD，停止主机输出时钟
2. 通过软复位模块方式，停止主机输出时钟

SPI 使用 DMA 传输数据

●描述：

SPI 使用 DMA 传输数据时建议外设 DAM 使能和 DMA 通道使能同时 ENABLE/DISABLE

Eg:

```
DMA_Cmd(FLASH_SPI_RX_DMA_CHANNEL, ENABLE);
DMA_Cmd(FLASH_SPI_TX_DMA_CHANNEL, ENABLE);
SPI_I2S_DMACmd(FLASH_SPI_MASTER, SPI_I2S_DMAReq_Tx, ENABLE);
SPI_I2S_DMACmd(FLASH_SPI_MASTER, SPI_I2S_DMAReq_Rx, ENABLE);
```

```
SPI_I2S_DMACmd(FLASH_SPI_MASTER, SPI_I2S_DMAReq_Tx, DISABLE);
SPI_I2S_DMACmd(FLASH_SPI_MASTER, SPI_I2S_DMAReq_Rx, DISABLE);
DMA_Cmd(FLASH_SPI_TX_DMA_CHANNEL, DISABLE);
DMA_Cmd(FLASH_SPI_RX_DMA_CHANNEL, DISABLE);
```

SPI RXE 置位，读取 DR 后，DR 中的数据保留

●描述：

SXX32F103：当 RXE 置位，读操作将返回接收缓冲区里的数据，并且会将 DR 清 0

MH2103A：当 RXE 置位，读操作将返回接收缓冲区里的数据，不会将 DR 清 0

●解决方法：

不影响正常收发数据

SPI 接收 FIFO 引起的不兼容问题

●描述：

1. SPI 配置为 Master 两线全双工模式

```
SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_256;
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_InitStructure.SPI_CRCPolynomial = 7;
SPI_Init(SPIx, &SPI_InitStructure);
```

2. 如下图操作

```
//发送Data0,Data1; 不操作接收DR
SPI_I2S_SendData(SPIx, Data0);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_TXE) == RESET);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_BSY) == RESET);
SPI_I2S_SendData(SPIx, Data1);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_TXE) == RESET);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_BSY) == RESET);

//读取1次DR,将之前接收到的数据清除
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_RXNE) == RESET);
SPI_I2S_ReceiveData(SPIx);

//发送数据Data2;并读取DR,获取接收的数据receive
SPI_I2S_SendData(SPIx, Data2);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_TXE) == RESET);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_BSY) == RESET);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_RXNE) == RESET);
receive = SPI_I2S_ReceiveData(SPIx);
```

SXX32F103：发送 Data2 所接收到的数据 receive,是正常数据

MH2103A：发送 Data2 所接收到的数据 receive,是异常数据

MH2103A SPI 具有 16Byte 接收 FIFO。上述发送 Data0，Data1 所接收到的数据缓存到了接收 FIFO 中，读取 1 次 DR 的操作并没有将 FIFO 中无用的数据全部清除，导致后续发送 Data2 所接收到的数据不是期望数据

●解决方法：

作为 Master 每次发送数据后，都读取一次 DR，确保 FIFO 中无残留数据即可

Eg:

```
//发送Data0,Data1; 每次发送数据后都读取1次DR
SPI_I2S_SendData(SPIx, Data0);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_TXE) == RESET);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_BSY) == RESET);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_RXNE) == RESET);
SPI_I2S_ReceiveData(SPIx);

SPI_I2S_SendData(SPIx, Data1);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_TXE) == RESET);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_BSY) == RESET);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_RXNE) == RESET);
SPI_I2S_ReceiveData(SPIx);

//发送数据Data2;并读取DR,获取接收的数据receive
SPI_I2S_SendData(SPIx, Data2);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_TXE) == RESET);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_BSY) == RESET);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_RXNE) == RESET);
receive = SPI_I2S_ReceiveData(SPIx);
```

SPI Master 在 BUSY 状态下关开 SPE ，现象差异

●描述：

如下图操作：

```
SPI_I2S_SendData(SPI1,0xAA);
while(SPI_I2S_GetFlagStatus(SPI1,SPI_I2S_FLAG_BSY) == RESET);
SPI_Cmd(SPI1, DISABLE);
SPI_Cmd(SPI1, ENABLE);

while(SPI_I2S_GetFlagStatus(SPI1,SPI_I2S_FLAG_TXE) == RESET);
SPI_I2S_SendData(SPI1,0x55);
```

SXX32F103：0xAA 和 0x55 均可正常发送

MH2103A：0xAA 正常发送，0x55 无法发送

MH2103A SPI 在 BUSY 状态下，若进行关开 SPE，则关 SPE 在数据发送完成(非 BUSY 状态下)生效，开 SPE 动作忽略不生效

●解决方法：

软件在通讯过程中(BUSY 状态)，避免操作 SPE。在操作 SPE 前判断是否处于 BUSY 状态。

3. 芯片标识说明

每款 MH2103A 型号都有自己的芯片标识，用于区分其他厂商。

标识存于基地址 0x1FFFF7E8 处

标识说明如下：

型号	型号标识
MH2103A CCT6	0x1A5A5CCX
MH2103A RPT6	0x1A6A5CDX
MH2103A VET6	0x1A8A6DDX
MH2103A VGT6	0x1A8A6EDX

历史版本

版本	变更
1.00	最初版本
1.01	新增系统功能、FLASH、USART、SPI/IIS 相关移植点
1.02	新增SPI 相关移植点
1.03	新增ADC、TIM相关移植点
1.04	新增非32bit对齐访问APB总线差异点
1.05	新增中断控制器差异
1.06	新增双ADC不同模式差异