Introduction

This migration guide is designed to help you analyze the steps required to migrate from an existing SXX32F103 device to an MH2103C device. This document gathers the most important information and lists the important things to be aware of.
To port an application from the SXX32F103 series to the MH2103C series, users need to analyze hardware migration, peripheral migration, and firmware migration.

Supported model list:

Supported model	MH2103Cxxxx

Catalogue

The MH2103C series microcontrollers are basically compatible with the SXX32F103 series, while enhancing many features, with some differences from the SXX32F103, detailed in this document.

1. Quickly replace the SXX32F103 chip

●Step 1: Compare peripheral specifications, Flash capacity, SRAM capacity, etc. , unsolder SXX32F103 and replace it with the corresponding model of MH2103C

●Step 2: Use ISP or KEIL to download the SXX32F103 HEX file or BIN file.

●Step 3: Download data other than SXX32F103 HEX file or BIN file or perform system corrections if necessary.

●Step 4: Check whether the program works properly.

●Step 5: Quick troubleshooting of other problems please refer to 2\3 electrical \ software transplantation precautions.

●Step 6: If the program does not work properly after the above steps, please refer to the other sections of this document, or contact the distributor and MH support staff for assistance.

2. Precautions for hardware migration

2.1 GPIO pin voltage withstand description

MH2103C PA11, PA12 pin withstand voltage limit is 3.6V, hardware circuit design pay attention to the voltage range.

Note:

The optional multiplexing functions of PA11 pin are CAN_RX, USART1_CTS, USBDM, TIM1_CH4;

The optional multiplexing functions of PA12 pin are CAN_TX, USART1_RTS, USBDP, TIM1_ETR;

2.2 USB pin instructions

When MH2103C uses USB, it is recommended to protect PA11 and PA12 with TVS diode.

2.3 Bootstrap mode select pin

When the BOOT0 pin is suspended, the chip has a probability of starting the execution program from the non-flash area after the reset startup. If you need to boot from user Flash, avoid dangling BOOT0.

An external 10K pull-down resistor or direct ground is recommended for BOOT0 pins, Can stably boot from user Flash.

Bootstrap mode select pin		Bootstrap mode	Aliasing
BOOT1	BOOT0		
X	0	User Flash	Select user Flash as the bootstrap space
0	1	System memory	Select system memory as the bootstrap space
1	1	Embedded SRAM	Select the embedded SRAM as the bootstrap space

3. Precautions for software migration

3.1 System module

3.1.1 Interrupt controller precautions

- Supports a maximum of 71 masked interrupt channels.
- Eight programmable priority levels (using a 3-bit interrupt priority).

3.1.2 Bootstrap mode configuration options are not reloaded during soft reset

Example of exceptions (Enabling read/write protection through an ISP) :

- BOOT0 connects to high, BOOT1 connects to low, external Reset, successfully connects to ISP.
- Set BOOT0 to low and enable read/write protection through ISP.
- After read/write protection is enabled, programs in the Flash are not executed.

Root cause:

After read/write protection is enabled through the ISP, the chip automatically initiates a system reset. During soft reset, the boot-up configuration pin is not reloaded and the BOOT pin configuration status is maintained. Therefore, the program is executed in the system memory again, resulting in no program execution in the Flash.

Solution:

After the bootstrap mode is changed, you can RESET the chip through

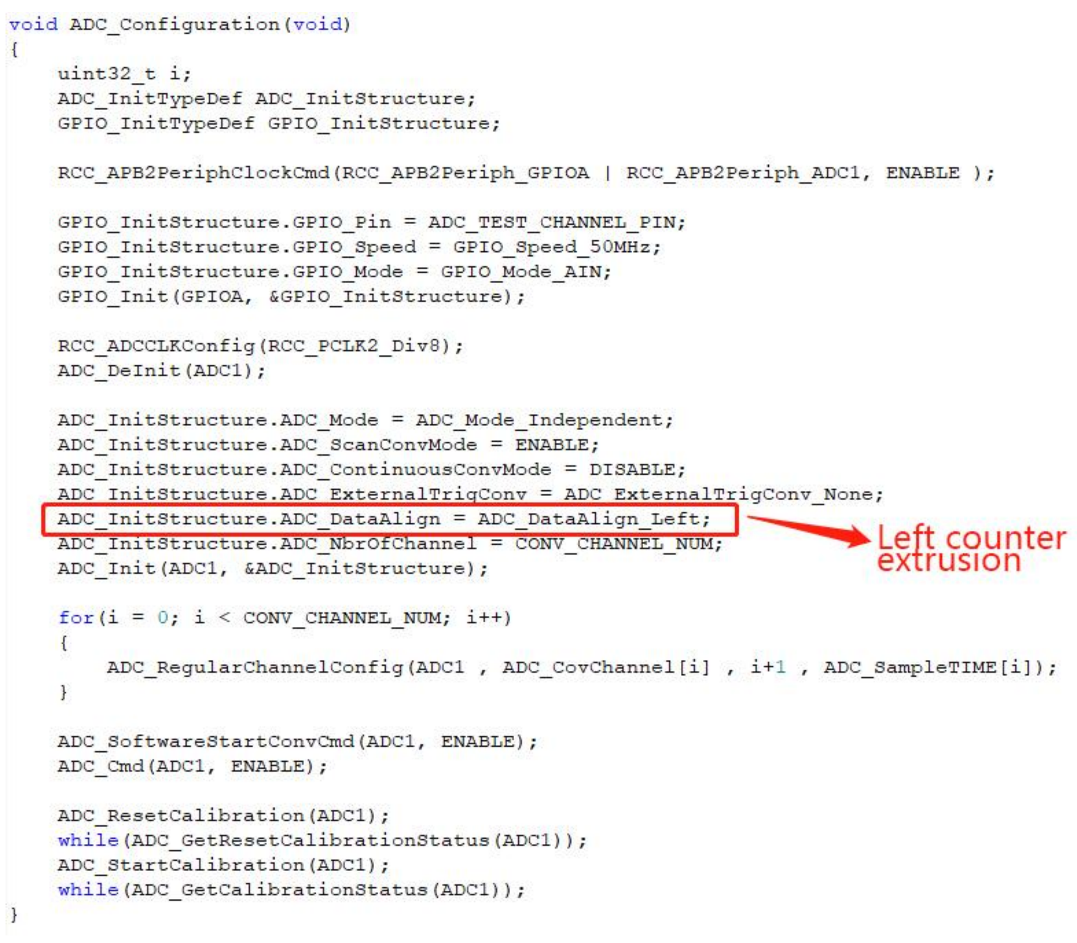
external reset or power it on or off again.

3.1.3 Non-32bit alignment access APB bus precautions

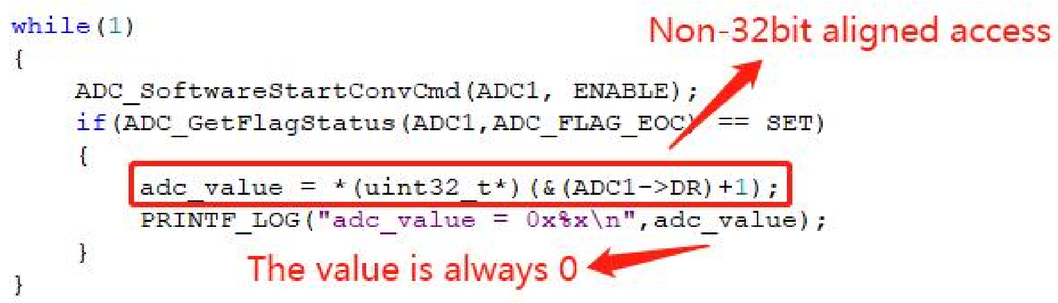
Problem description:

When the ADC is configured in left-justified mode, the converted high 8-bit data is obtained from the address 0x4001244E, and the obtained value is fixed to 0.

The program is configured as follows:



```
void ADC_Configuration(void)
{
    uint32_t i;
    ADC_InitTypeDef ADC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_ADC1, ENABLE );

    GPIO_InitStructure.GPIO_Pin = ADC_TEST_CHANNEL_PIN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    RCC_ADCCLKConfig(RCC_PCLK2_Div8);
    ADC_DeInit(ADC1);

    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = ENABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Left;
    ADC_InitStructure.ADC_NbrOfChannel = CONV_CHANNEL_NUM;
    ADC_Init(ADC1, &ADC_InitStructure);

    for(i = 0; i < CONV_CHANNEL_NUM; i++)
    {
        ADC_RegularChannelConfig(ADC1 , ADC_CovChannel[i] , i+1 , ADC_SampleTIME[i]);
    }

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

    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));
}
```



```
while(1)
{
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
    if(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC) == SET)
    {
        adc_value = *(uint32_t*)(&(ADC1->DR)+1);
        PRINTF_LOG("adc_value = 0x%x\n",adc_value);
    }
}
```

Root cause:

When accessing the APB bus, you must use 32-bit alignment; otherwise, the APB bus cannot be accessed.

Solution:

When accessing the registers of the APB bus, the alignment needs to be 32bit. If you need a field in the 32bit register, read it out in alignment and then process it.

3.1.4 Precautions to distinguish the MH2103C chip from the SXX32F103 chip

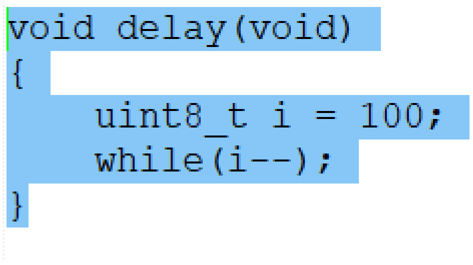
Read the base address 0x1FFFF7E8 and get a 32bit identifier to distinguish. As shown in the picture below:
The identification is described as follows:

Model number	Type identification
MH2103C CBT6	0x1C5A5BBX
MH2103C CCT6	0x1C5A5CCX

3.1.5 Software delay precautions

Problem description:

Some software delays implemented using other platforms need to be adjusted, such as the following simple software delay function:



```
void delay(void)
{
    uint8_t i = 100;
    while(i--);
}
```

The delay time for SXX32F103 to execute this function is 10us.

The delay time for MH2103C to execute this function is 7.2us.

If the application has strict requirements on the software delay time

, adjust the software delay parameters to some extent

3.1.6 TRACESWO as a Printf function

Problem description:

When TRACESWO pin (PB3) is used as the Log output, no Log is found.

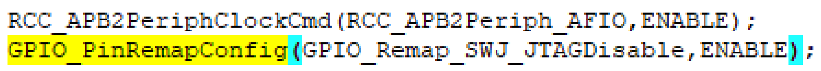
Root cause:

MH2103CPB3 takes precedence over TRACESWO as a JTDO function.

Solution:

If TRACESWO pins (PB3) are used as Log output, JTAG multiplexing needs to be configured.

As shown below:



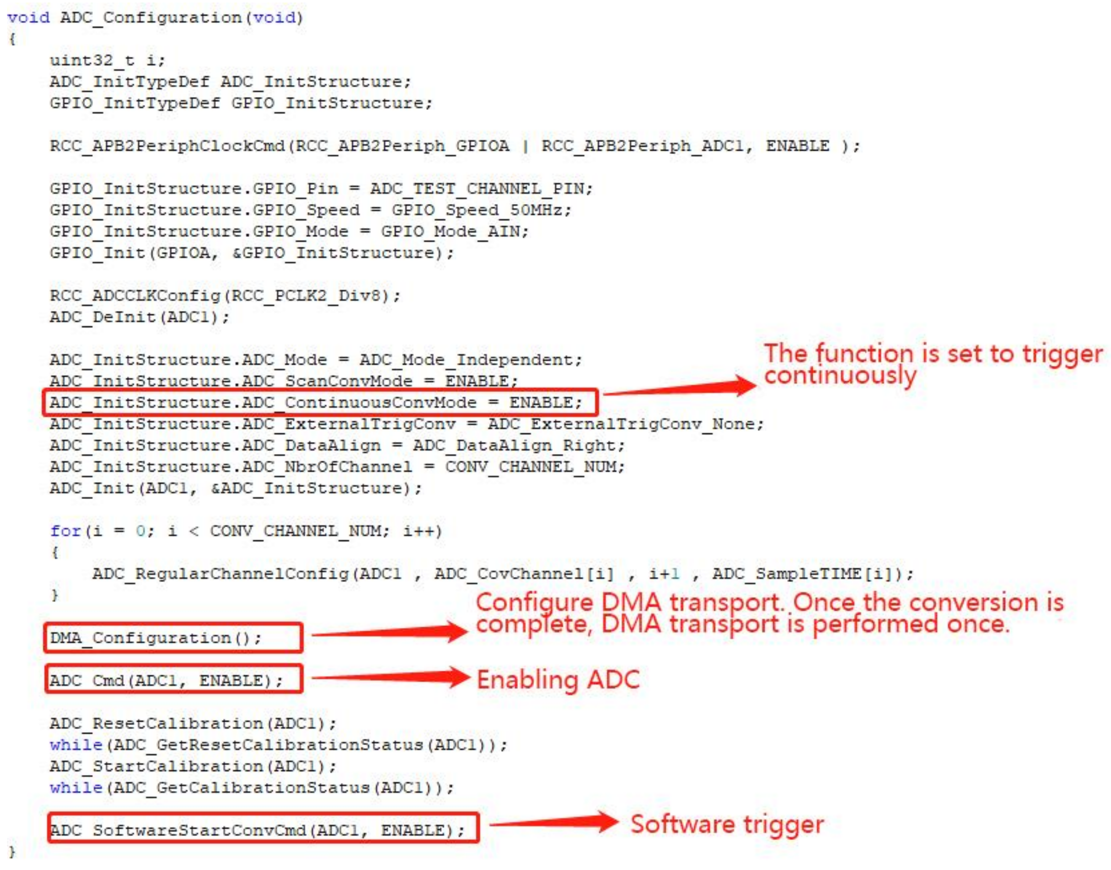
```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable,ENABLE);
```

3.2 ADC module

3.2.1 The ADC is configured to trigger continuously. Precautions for stopping the trigger

Repeat the problem as follows:

- Configure ADC for continuous sampling and enable DMA handling. After the software is triggered, ADC conversion completes once, DMA handling completes once, and DMA completion interrupts once.
- Configure to turn off ADC enablement in DMA completion interrupt.
- MH2103C will continue to convert the ADC after disabling the ADC, and will continue to enter the DMA completion interrupt



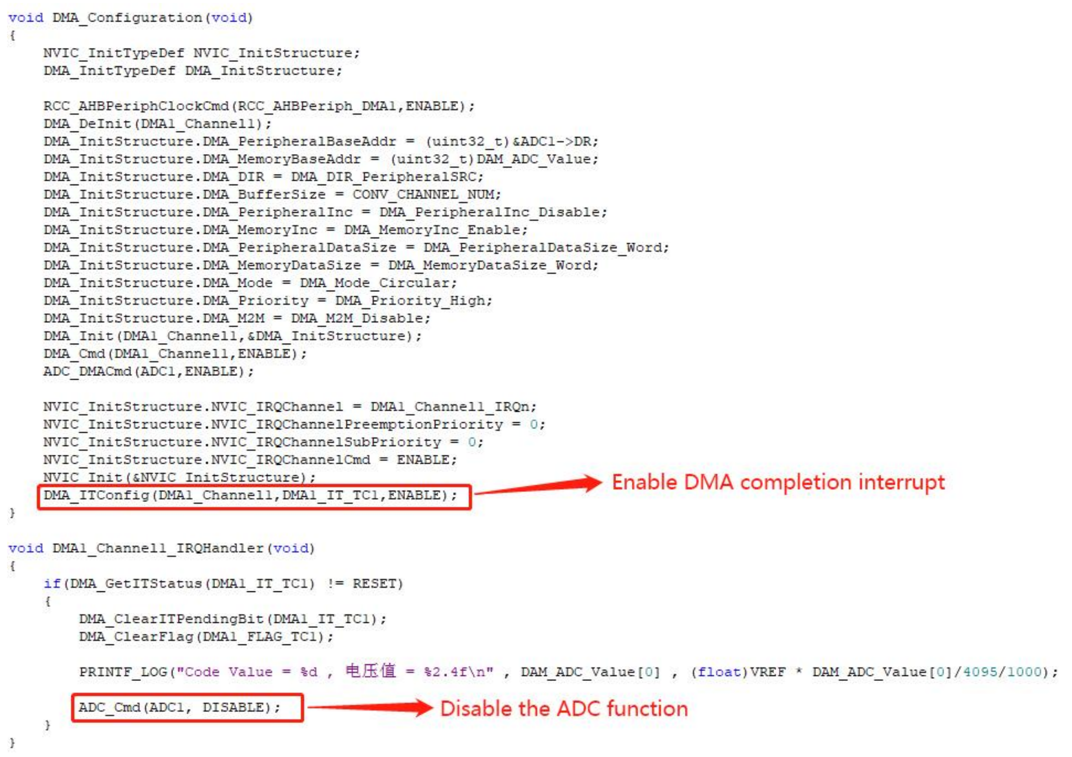
```
void ADC_Configuration(void)
{
    uint32_t i;
    ADC_InitTypeDef ADC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_ADC1, ENABLE );

    GPIO_InitStructure.GPIO_Pin = ADC_TEST_CHANNEL_PIN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    RCC_ADCCLKConfig(RCC_PCLK2_Div8);
    ADC_DeInit(ADC1);

    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = ENABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = CONV_CHANNEL_NUM;
    ADC_Init(ADC1, &ADC_InitStructure);

    for(i = 0; i < CONV_CHANNEL_NUM; i++)
    {
        ADC_RegularChannelConfig(ADC1 , ADC_CovChannel[i] , i+1 , ADC_SampleTIME[i]);
    }

    DMA_Configuration();

    ADC_Cmd(ADC1, ENABLE);

    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));

    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

The function is set to trigger continuously

Configure DMA transport. Once the conversion is complete, DMA transport is performed once.

Enabling ADC

Software trigger



```
void DMA_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    DMA_InitTypeDef DMA_InitStructure;

    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1,ENABLE);
    DMA_DeInit(DMA1_Channel1);
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR;
    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DAM_ADC_Value;
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
    DMA_InitStructure.DMA_BufferSize = CONV_CHANNEL_NUM;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    DMA_Init(DMA1_Channel1,&DMA_InitStructure);
    DMA_Cmd(DMA1_Channel1,ENABLE);
    ADC_DMACmd(ADC1,ENABLE);

    NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
    DMA_ITConfig(DMA1_Channel1,DMA1_IT_TC1,ENABLE);
}

void DMA1_Channel1_IRQHandler(void)
{
    if(DMA_GetITStatus(DMA1_IT_TC1) != RESET)
    {
        DMA_ClearITPendingBit(DMA1_IT_TC1);
        DMA_ClearFlag(DMA1_FLAG_TC1);

        PRINTF_LOG("Code Value = %d , 电压值 = %2.4f\n" , DAM_ADC_Value[0] , (float)VREF * DAM_ADC_Value[0]/4095/1000);

        ADC_Cmd(ADC1, DISABLE);
    }
}
```

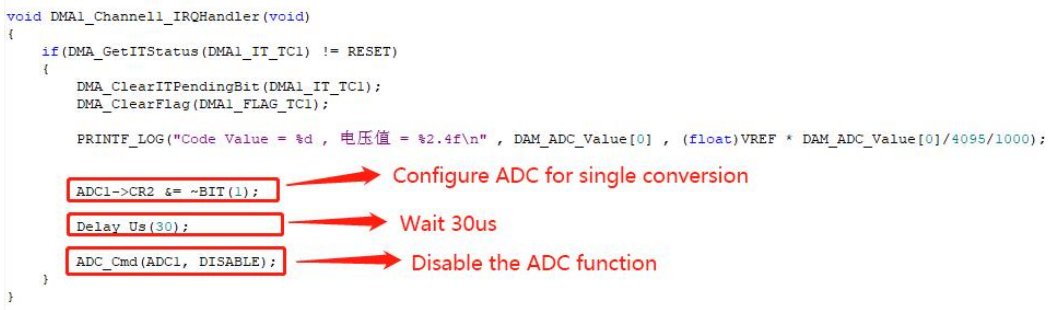
Solution:

Configure ADC for single conversion and wait until the last sampling period is ADC_CLK before disabling ADC.

As shown in the picture below:

The ADC clock configured in the above example is 9M and the sampling period is 239.5.

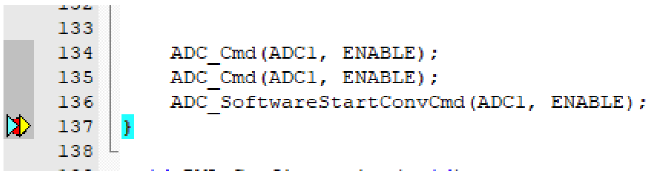
Therefore, the waiting time is (239.5+12.5)*1/9000 ≈27.922us, so the waiting time of 30us in the figure below is sufficient.



```
void DMA1_Channel1_IRQHandler(void)
{
    if(DMA_GetITStatus(DMA1_IT_TC1) != RESET)
    {
        DMA_ClearITPendingBit(DMA1_IT_TC1);
        DMA_ClearFlag(DMA1_FLAG_TC1);

        PRINTF_LOG("Code Value = %d , 电压值 = %2.4f\n" , DAM_ADC_Value[0] , (float)VREF * DAM_ADC_Value[0]/4095/1000);

        ADC1->CR2 &= ~BIT(1);
        Delay_Us(30);
        ADC_Cmd(ADC1, DISABLE);
    }
}
```

3.2.2 Software trigger Precautions for enabling ADON to perform external events twice in a row

Problem description:



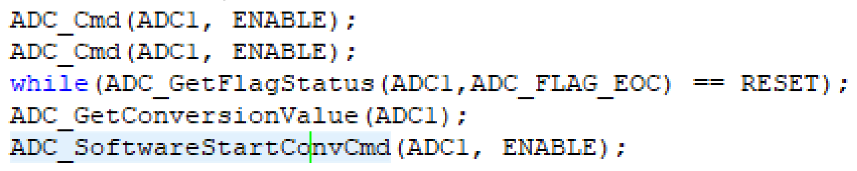
```
    ADC_Cmd(ADC1, ENABLE);
    ADC_Cmd(ADC1, ENABLE);
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

After the above operation, MH2103C will fail to clear 0 after EOC setting, and abnormal ADC conversion value will occur.

Solution:

After enabling ADC, enable ADC again, wait for EOC to be set, and read the quantized value.

As shown in the picture below:



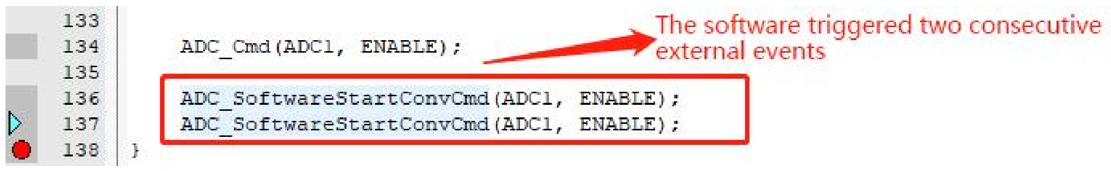
```
ADC_Cmd(ADC1, ENABLE);
ADC_Cmd(ADC1, ENABLE);
while(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC) == RESET);
ADC_GetConversionValue(ADC1);
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

Attention:

When the ADC is enabled and the ADC is enabled again, both SXX32F103 and MH2103C trigger an ADC conversion.

3.2.3 The ADC enables software triggers for external events twice in a row

Problem description:



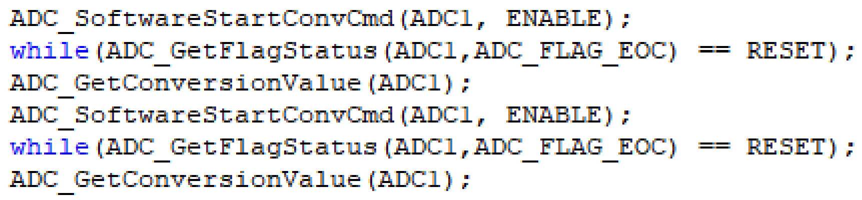
```
ADC_Cmd(ADC1, ENABLE);

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

After the above operation, MH2103C will fail to clear 0 after EOC setting, and abnormal ADC conversion value will occur.

Solution:

The software triggers the external event, waits for the EOC to set, and reads the quantized value.

As shown in the picture below:



```
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);
```

3.2.4 Precautions for using different modes of dual ADC

Synchronous injection, alternate trigger mode:

SXX32F103: After the injection group channel or rule group channel is triggered, the conversion is normal.

MH2103C: Normal transformation after the injection group channel is triggered; After the rule group channel is triggered, no conversion is performed.

Synchronous regular, fast crossover, slow crossover mode:

SXX32F103: After the injection group channel or rule group channel is triggered, the conversion is normal.

MH2103C: Normal transformation after the rule group channel is triggered; After the injection group channel is triggered, no conversion is performed.

3.2.5 Precautions for ADC and DMA configuration

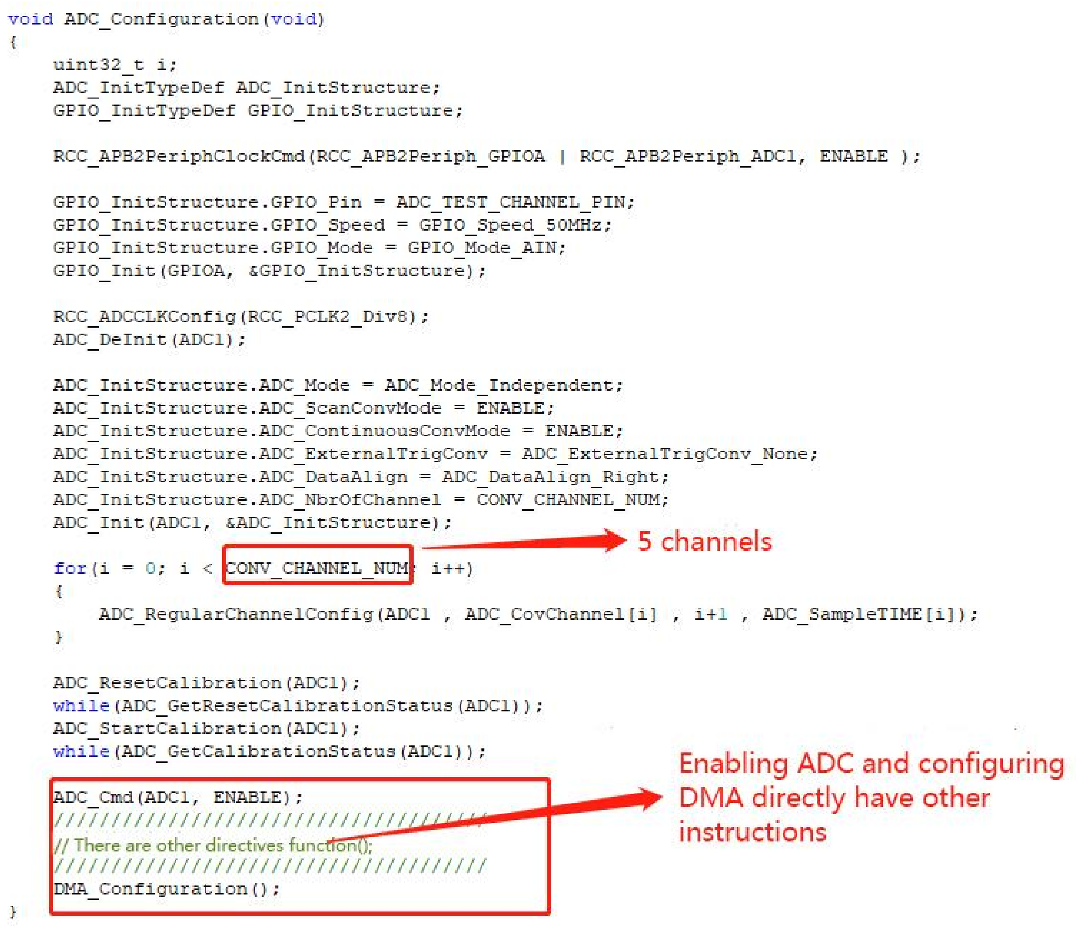
Problem description:

When an ADC is configured to continuously sample multiple channels and DMA is used to move data, errors (mismatches) occur in ADC quantization values. For example, convert five channels, configure them as rule groups, and scan continuously for conversion.

Expectations for {0V,1V,1.5V,2V,2.5V,   0V,1V,1.5V,2V,2.5V ,   0V,1V,1.5V,2V,2.5V …};

To obtain the value of the actual software {1.5V,2V,2,5V,0V,1V,   1.5V,2V,2,5V,0V,1V ,   1.5V,2V,2,5V,0V,1V …}

The following figure shows the configuration

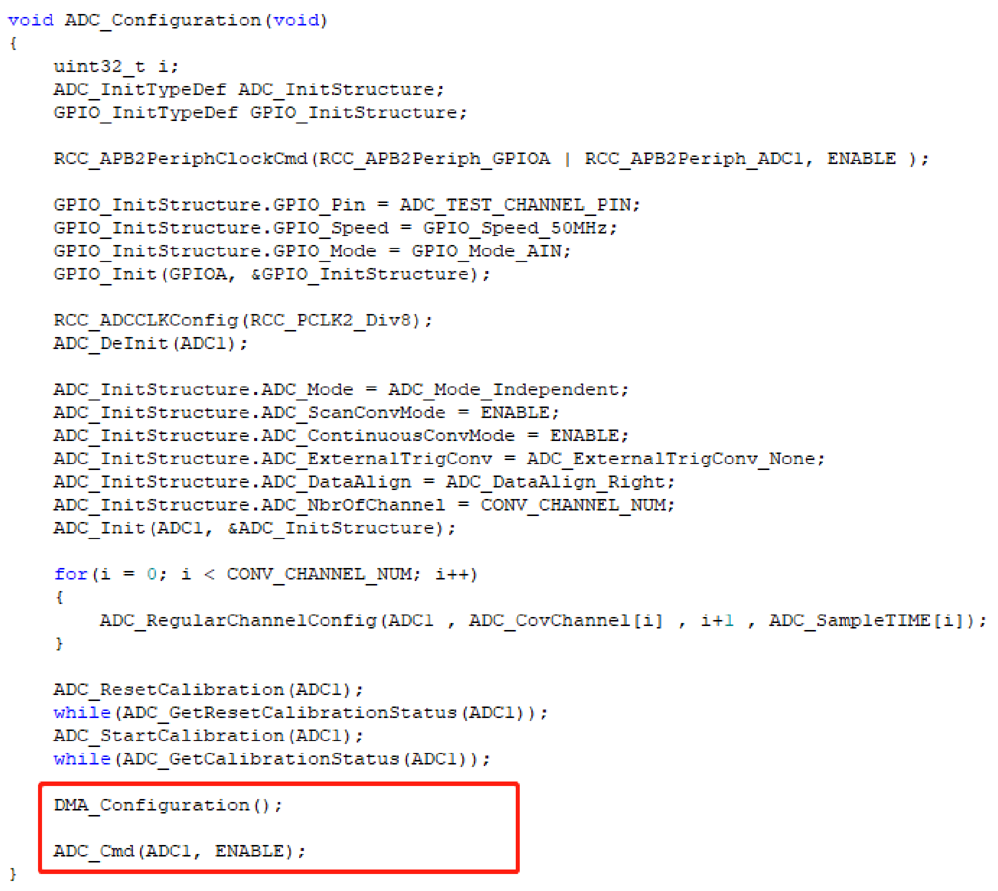

5 channels
Enabling ADC and configuring DMA directly have other instructions

```c
void ADC_Configuration(void)
{
    uint32_t i;
    ADC_InitTypeDef ADC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_ADC1, ENABLE );

    GPIO_InitStructure.GPIO_Pin = ADC_TEST_CHANNEL_PIN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    RCC_ADCCLKConfig(RCC_PCLK2_Div8);
    ADC_DeInit(ADC1);

    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = ENABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = CONV_CHANNEL_NUM;
    ADC_Init(ADC1, &ADC_InitStructure);

    for(i = 0; i < CONV_CHANNEL_NUM; i++)
    {
        ADC_RegularChannelConfig(ADC1 , ADC_CovChannel[i] , i+1 , ADC_SampleTIME[i]);
    }

    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));

    ADC_Cmd(ADC1, ENABLE);
    //////////////////////////////////////
    // There are other directives function();
    //////////////////////////////////////
    DMA_Configuration();
}
```

Solution:

If DMA is configured and then ADC is enabled, ADC quantization value error (misalignment) does not occur.

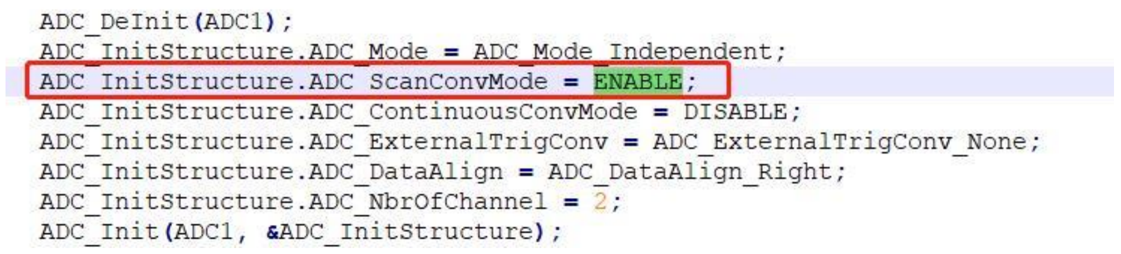
As shown in the following picture



```
void ADC_Configuration(void)
{
    uint32_t i;
    ADC_InitTypeDef ADC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_ADC1, ENABLE );

    GPIO_InitStructure.GPIO_Pin = ADC_TEST_CHANNEL_PIN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    RCC_ADCCLKConfig(RCC_PCLK2_Div8);
    ADC_DeInit(ADC1);

    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = ENABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = CONV_CHANNEL_NUM;
    ADC_Init(ADC1, &ADC_InitStructure);

    for(i = 0; i < CONV_CHANNEL_NUM; i++)
    {
        ADC_RegularChannelConfig(ADC1 , ADC_CovChannel[i] , i+1 , ADC_SampleTIME[i]);
    }

    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));

    DMA_Configuration();

    ADC_Cmd(ADC1, ENABLE);
}
```

3.2.6 ADC automatic injection mode use precautions

When using ADC auto-injection mode, scanning needs to be enabled, as follows:



```
ADC_DeInit(ADC1);
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 2;
ADC_Init(ADC1, &ADC_InitStructure);
```

3.3 TIM module

3.3.1 Precautions for using external brakes

Problem description:

When the time interval between two consecutive brakes is less than one TIM clock cycle, there will be a BIF(brake interrupt mark) set 1, unable to clear 0 phenomenon.

Solution:

Method 1: Software configuration brake signal is normal input IO interrupt , do not use BIF and brake interrupt.
Method 2: Set 1 in the BIF to soft-reset TIM.

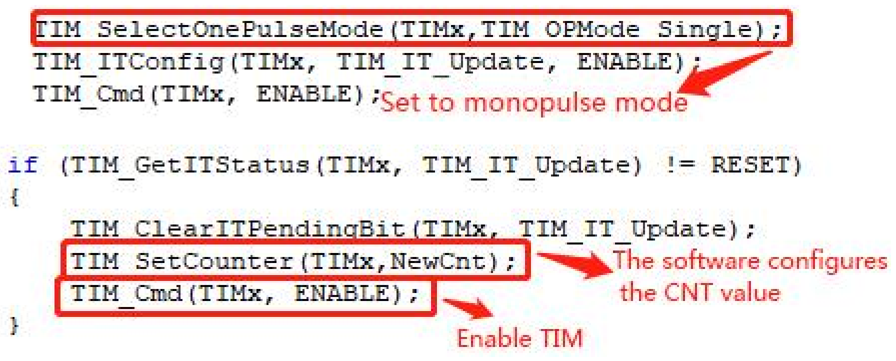
3.3.2 Precautions for software CNT value modification

Problem description:

In some cases, after the software changes the CNT value, TIM does not count the CNT reconfigured.

Case 1:

- TIM is configured in monopulse mode with update interrupt enabled.
- Modify the CNT value in interrupt and enable TIM.



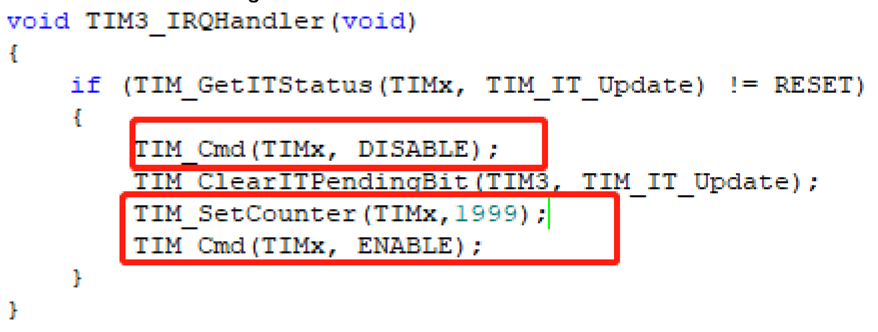
```
TIM_SelectOnePulseMode(TIMx,TIM_OPMode_Single);
TIM_ITConfig(TIMx, TIM_IT_Update, ENABLE);
TIM_Cmd(TIMx, ENABLE);

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

Attention:

In monopulse mode, the TIM counters of the SXX32F103 and MH2103C automatically stop when the next update event UEV is generated.

Case 2:

- TIM is not configured with monopulse mode.
- In interrupt, the software disables TIM, changes the CNT value, and enables TIM again.



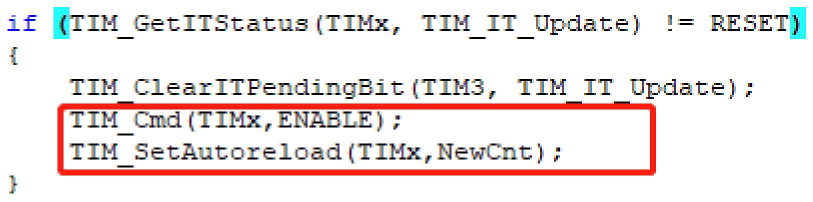
```
void TIM3_IRQHandler(void)
{
    if (TIM_GetITStatus(TIMx, TIM_IT_Update) != RESET)
    {
        TIM_Cmd(TIMx, DISABLE);
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
        TIM_SetCounter(TIMx,1999);
        TIM_Cmd(TIMx, ENABLE);
    }
}
```

Root cause:

The software modification CNT takes effect only when CEN is enabled.

Solution:

- If the monopulse mode is configured, after the update event is generated; To manually change the CNT value, enable TIM first.



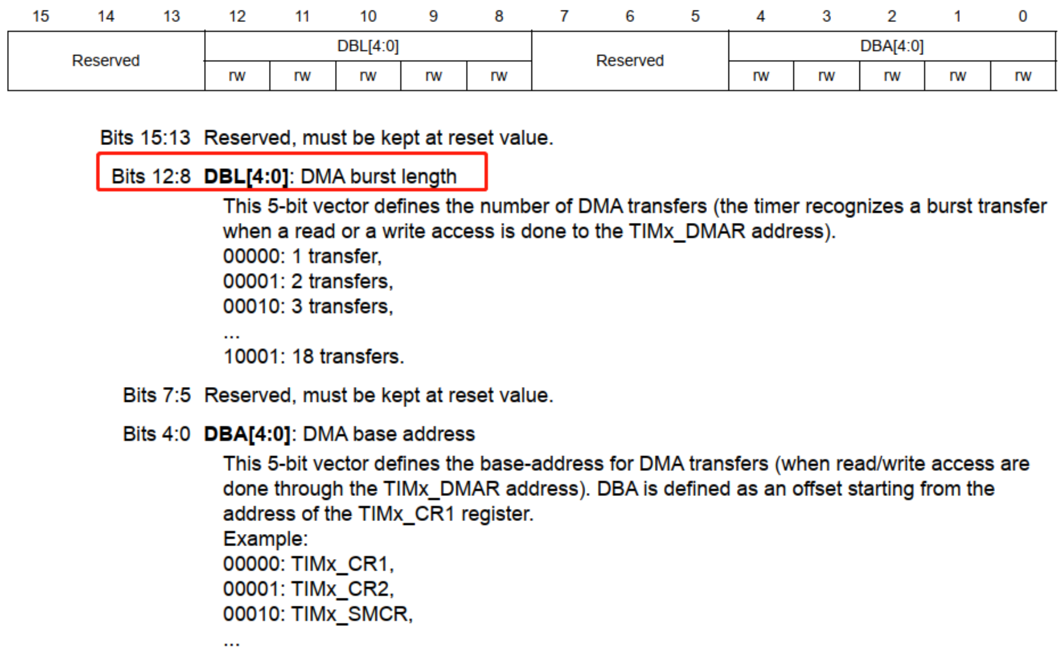
```
if (TIM_GetITStatus(TIMx, TIM_IT_Update) != RESET)
{
    TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
    TIM_Cmd(TIMx,ENABLE);
    TIM_SetAutoreload(TIMx,NewCnt);
}
```

- If the monopulse mode is not configured, you need to manually modify the CNT value and do not need to turn off TIM.

3.3.3 TIMDMABurst function use precautions

Problem description:

When the Advanced & General TIM configures DBL to be non-0, a DMA request is generated, and DMA carries data only once.



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			DBL[4:0]					Reserved			DBA[4:0]				
			rw	rw	rw	rw	rw				rw	rw	rw	rw	rw

Bits 15:13 Reserved, must be kept at reset value.

Bits 12:8 **DBL[4:0]**: DMA burst length
This 5-bit vector defines the number of DMA transfers (the timer recognizes a burst transfer when a read or a write access is done to the TIMx_DMAR address).
00000: 1 transfer,
00001: 2 transfers,
00010: 3 transfers,
...
10001: 18 transfers.

Bits 7:5 Reserved, must be kept at reset value.

Bits 4:0 **DBA[4:0]**: DMA base address
This 5-bit vector defines the base-address for DMA transfers (when read/write access are done through the TIMx_DMAR address). DBA is defined as an offset starting from the address of the TIMx_CR1 register.
Example:
00000: TIMx_CR1,
00001: TIMx_CR2,
00010: TIMx_SMCR,
...

Root cause:

The function mechanism of TIMDMABurst is different from SXX32F103.

For SXX32F103:

When the Advanced & General TIM configures DBL to non-0, a DMA request is generated and DMA carries (DBL+1) data

For MH2103C:

When the Advanced & General TIM configures DBL to non-0, a DMA request is generated and DMA carries (DBL+1) data

Solution:

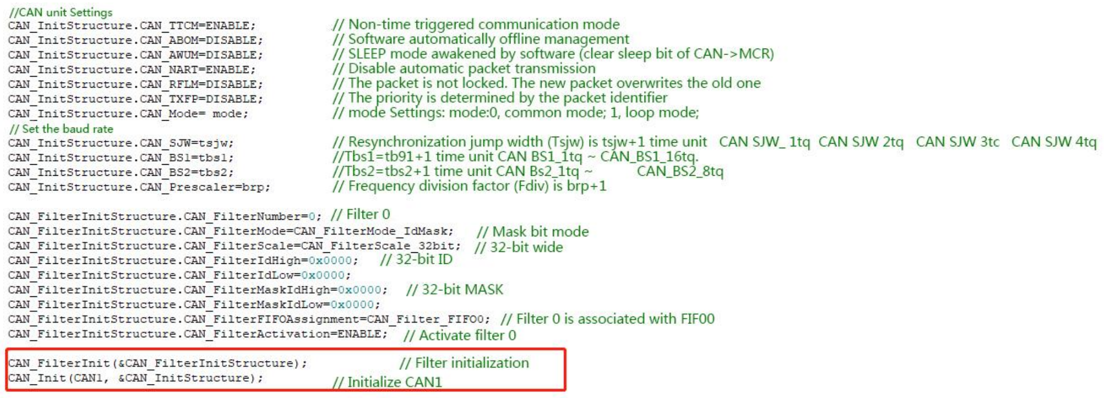
The software can configure TIM to send different DMA request signals to different DMA channels at the same time, and each DMA channel carries a data to achieve the TIMDMABurst effect of SXX32F103.

3.4 CAN module

3.4.1 Precautions for configuring a filter in hibernation mode

Problem description:

The CAN module first initializes the filter and then initializes the controller, resulting in the phenomenon that data can be sent but cannot be received. The code is as follows:



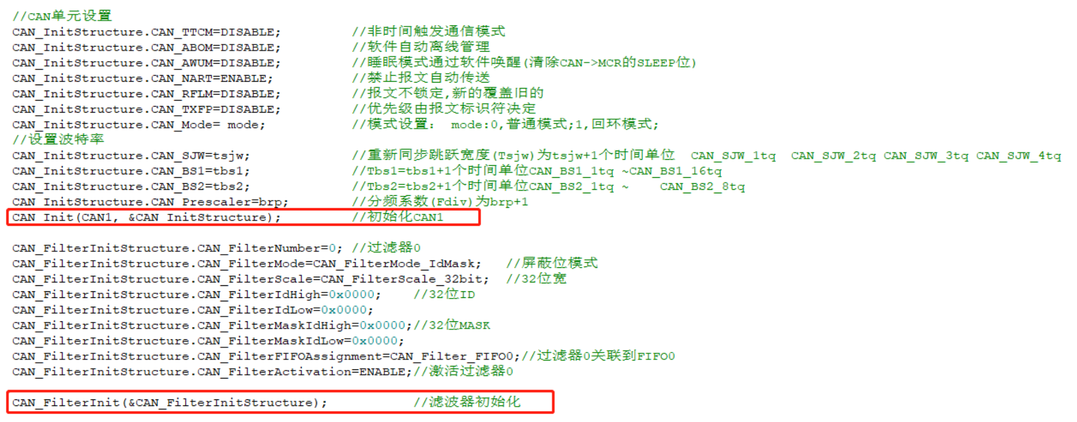
```
//CAN unit Settings
CAN_InitStructure.CAN_TTCM=ENABLE;              // Non-time triggered communication mode
CAN_InitStructure.CAN_ABOM=DISABLE;             // Software automatically offline management
CAN_InitStructure.CAN_AWUM=DISABLE;             // SLEEP mode awakened by software (clear sleep bit of CAN->MCR)
CAN_InitStructure.CAN_NART=ENABLE;              // Disable automatic packet transmission
CAN_InitStructure.CAN_RFLM=DISABLE;             // The packet is not locked. The new packet overwrites the old one
CAN_InitStructure.CAN_TXFP=DISABLE;             // The priority is determined by the packet identifier
CAN_InitStructure.CAN_Mode= mode;               // mode Settings: mode:0, common mode; 1, loop mode;
// Set the baud rate
CAN_InitStructure.CAN_SJW=tsjw;                 // Resynchronization jump width (Tsjw) is tsjw+1 time unit  CAN SJW_ 1tq  CAN SJW 2tq  CAN SJW 3tc  CAN SJW 4tq
CAN_InitStructure.CAN_BS1=tbs1;                 //Tbs1=tb91+1 time unit CAN BS1_1tq ~ CAN_BS1_16tq.
CAN_InitStructure.CAN_BS2=tbs2;                 //Tbs2=tbs2+1 time unit CAN Bs2_1tq ~       CAN_BS2_8tq
CAN_InitStructure.CAN_Prescaler=brp;            // Frequency division factor (Fdiv) is brp+1

CAN_FilterInitStructure.CAN_FilterNumber=0; // Filter 0
CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask;    // Mask bit mode
CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit;  // 32-bit wide
CAN_FilterInitStructure.CAN_FilterIdHigh=0x0000;    // 32-bit ID
CAN_FilterInitStructure.CAN_FilterIdLow=0x0000;
CAN_FilterInitStructure.CAN_FilterMaskIdHigh=0x0000;    // 32-bit MASK
CAN_FilterInitStructure.CAN_FilterMaskIdLow=0x0000;
CAN_FilterInitStructure.CAN_FilterFIFOAssignment=CAN_Filter_FIFO0; // Filter 0 is associated with FIF00
CAN_FilterInitStructure.CAN_FilterActivation=ENABLE;   // Activate filter 0

CAN_FilterInit(&CAN_FilterInitStructure);                 // Filter initialization
CAN_Init(CAN1, &CAN_InitStructure);             // Initialize CAN1
```

Root cause:

When configuring the CAN filter for the MH2103C, ensure that the CAN is in non-sleep mode.

Solution:

To solve this problem, initialize the CAN controller and then configure the filter.



```
//CAN单元设置
CAN_InitStructure.CAN_TTCM=DISABLE;          //非时间触发通信模式
CAN_InitStructure.CAN_ABOM=DISABLE;          //软件自动离线管理
CAN_InitStructure.CAN_AWUM=DISABLE;          //睡眠模式通过软件唤醒(清除CAN->MCR的SLEEP位)
CAN_InitStructure.CAN_NART=ENABLE;           //禁止报文自动传送
CAN_InitStructure.CAN_RFLM=DISABLE;          //报文不锁定,新的覆盖旧的
CAN_InitStructure.CAN_TXFP=DISABLE;          //优先级由报文标识符决定
CAN_InitStructure.CAN_Mode= mode;            //模式设置： mode:0,普通模式;1,回环模式;
//设置波特率
CAN_InitStructure.CAN_SJW=tsjw;              //重新同步跳跃宽度(Tsjw)为tsjw+1个时间单位  CAN_SJW_1tq  CAN_SJW_2tq CAN_SJW_3tq CAN_SJW_4tq
CAN_InitStructure.CAN_BS1=tbs1;              //Tbs1=tbs1+1个时间单位CAN_BS1_1tq ~CAN_BS1_16tq
CAN_InitStructure.CAN_BS2=tbs2;              //Tbs2=tbs2+1个时间单位CAN_BS2_1tq ~    CAN_BS2_8tq
CAN_InitStructure.CAN_Prescaler=brp;         //分频系数(Fdiv)为brp+1
CAN_Init(CAN1, &CAN_InitStructure);          //初始化CAN1

CAN_FilterInitStructure.CAN_FilterNumber=0; //过滤器0
CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask;   //屏蔽位模式
CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit;  //32位宽
CAN_FilterInitStructure.CAN_FilterIdHigh=0x0000;    //32位ID
CAN_FilterInitStructure.CAN_FilterIdLow=0x0000;
CAN_FilterInitStructure.CAN_FilterMaskIdHigh=0x0000;//32位MASK
CAN_FilterInitStructure.CAN_FilterMaskIdLow=0x0000;
CAN_FilterInitStructure.CAN_FilterFIFOAssignment=CAN_Filter_FIFO0;//过滤器0关联到FIFO0
CAN_FilterInitStructure.CAN_FilterActivation=ENABLE;//激活过滤器0

CAN_FilterInit(&CAN_FilterInitStructure);           //滤波器初始化
```

3.4.2 Send time stamp precautions

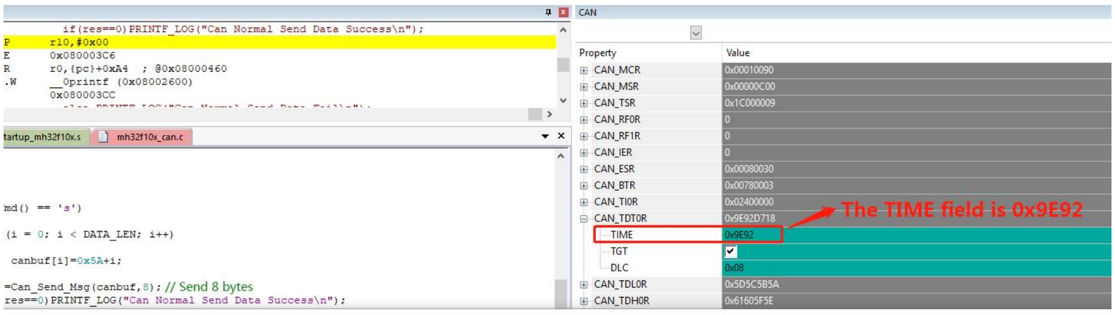
Problem description:

Due to the different design, MH2103C uses CAN to send the timestamp, and the TIME[15:0] field in the CAN_TDTxR register is different from that in SXX32F103 to fill the message location.

For SXX32F103：

TIME[7:0] as the seventh byte and TIME[15:8] as the eighth byte replace the data written to CAN_TDHxR[31:16] (DATA6[7:0] and DATA7[7:0]).

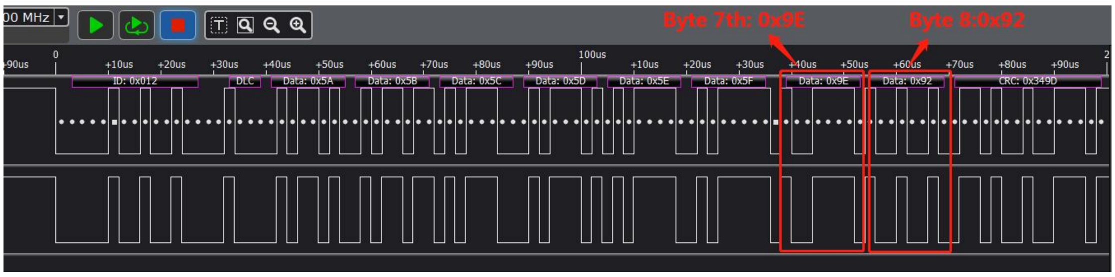
Bit 31:16	TIME[15:0]: Message time stamp This field contains the value of the 16-bit timer at the time the packet is sent.
Bit 15:9	Reserved bit
Bit 8	TGT: Transmit global time This bit is only valid if CAN is in time-triggered communication mode, i.e. the TTCM bit of the CAN__MCR register is '1'. 0: does not send timestamp TIME[15:0]; 1: Send timestamp TIME[15:0]. In a packet of length 8, the timestamp TIME[1 5:0] is the last two bytes sent: TIME[7:0] as the seventh byte and TIME[15:8] as the eighth byte replace the data written to CAN__TDHxR [31:16] (DATA6[7:0] and DATA7[7:0]). In order to send out 2 bytes of the timestamp, the DLC must be programmed to 8.
Bit 7:4	Reserve the bit
Bit 3:0	DLC[15:0]: Send Data length code This field specifies the data length of the data message or the data length of the remote frame request. A message contains 0 to 8 bytes of data, which is determined by the DLC.



```
        if(res==0)PRINTF_LOG("Can Normal Send Data Success\n");
P       r10,#0x00
E       0x080003C6
R       r0,{pc}+0xA4  ; @0x08000460
.W      __0printf (0x08002600)
        0x080003CC
```

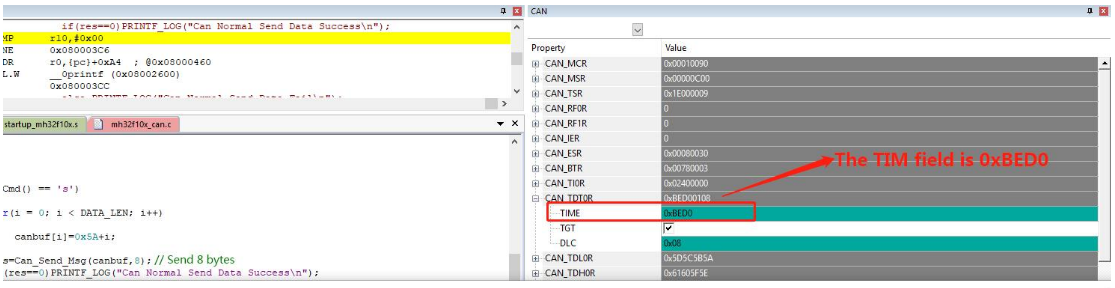
```
md() == 's')

(i = 0; i < DATA_LEN; i++)

 canbuf[i]=0x5A+i;

=Can_Send_Msg(canbuf,8); // Send 8 bytes
res==0)PRINTF_LOG("Can Normal Send Data Success\n");
```

Property	Value
CAN_MCR	0x00010090
CAN_MSR	0x00000C00
CAN_TSR	0x1C000009
CAN_RF0R	0
CAN_RF1R	0
CAN_IER	0
CAN_ESR	0x00080030
CAN_BTR	0x00780003
CAN_TI0R	0x02400000
CAN_TDT0R	0x9E92D718
TIME	0x9E92
TGT	✓
DLC	0x08
CAN_TDL0R	0x5D5C5B5A
CAN_TDH0R	0x61605F5E



For MH2103C:

TIME[7:0] as the eighth byte and TIME[15:8] as the seventh byte replace the data written to CAN_TDHxR[31:16] (DATA6[7:0] and DATA7[7:0]).



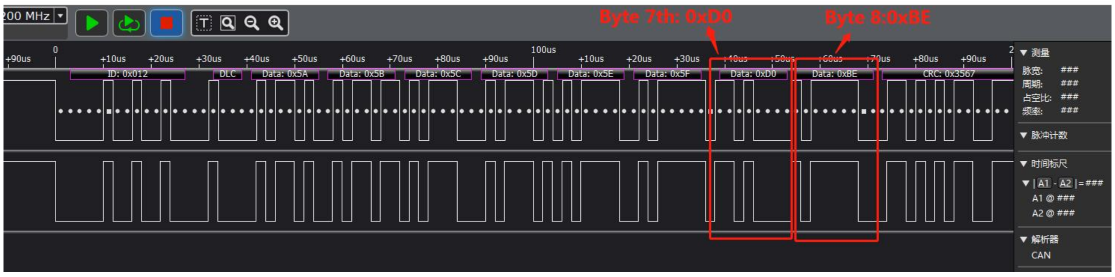
```
            if(res==0)PRINTF_LOG("Can Normal Send Data Success\n");
MP        r10,#0x00
NE        0x080003C6
DR        r0,{pc}+0xA4  ; @0x08000460
L.W       __0printf (0x08002600)
          0x080003CC
```

startup_mh32f10x.s | mh32f10x_can.c

```
Cmd() == 's')

r(i = 0; i < DATA_LEN; i++)

  canbuf[i]=0x5A+i;

s=Can_Send_Msg(canbuf,8); // Send 8 bytes
(res==0)PRINTF_LOG("Can Normal Send Data Success\n");
```

CAN

Property	Value
CAN_MCR	0x00010090
CAN_MSR	0x00000C00
CAN_TSR	0x1E000009
CAN_RF0R	0
CAN_RF1R	0
CAN_IER	0
CAN_ESR	0x00080030
CAN_BTR	0x00780003
CAN_TI0R	0x02400000
CAN_TDT0R	0xBED00108
TIME	0xBED0
TGT	✓
DLC	0x08
CAN_TDL0R	0x5D5C5B5A
CAN_TDH0R	0x61605F5E

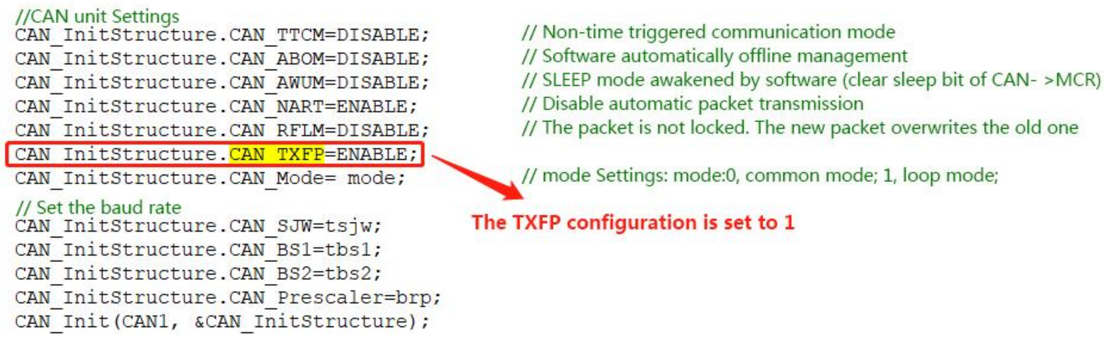


3.4.3 When TXFP is 1, the FIFO priority is different

Problem description:

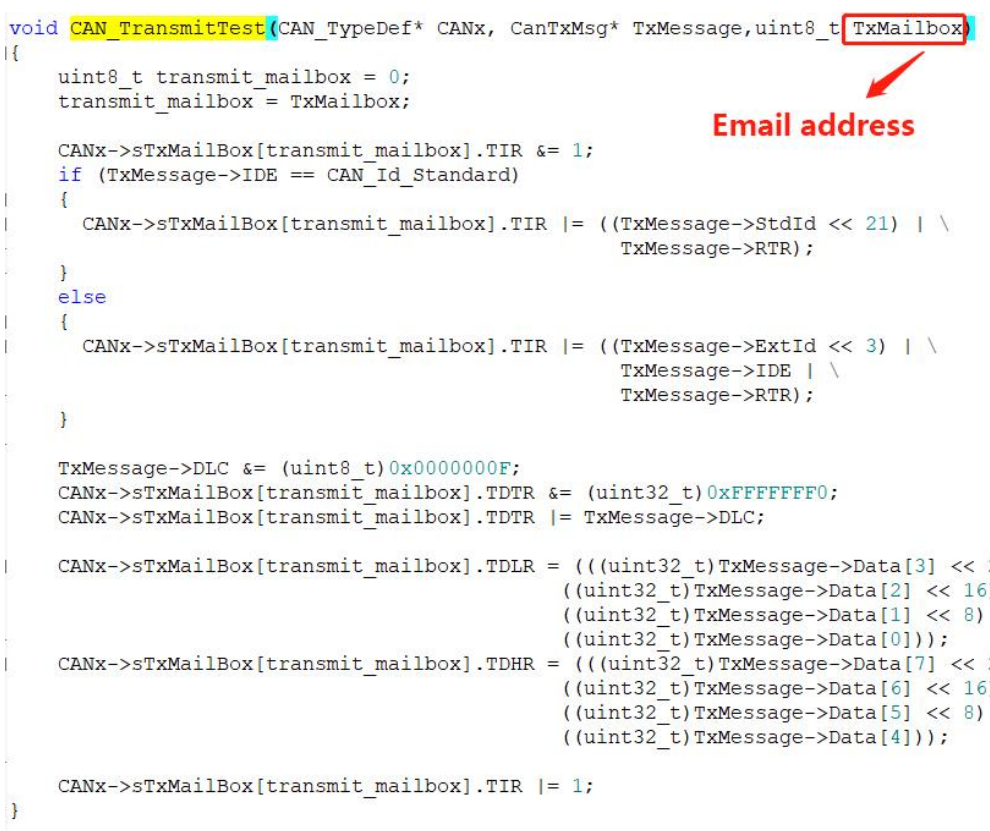
Due to different designs, the TXFP is set to 1 for the MH2103C. When multiple packets are waiting to be sent, the priority of sending packets is different from that of the SXX32F103.

The initial CAN configuration is as follows:



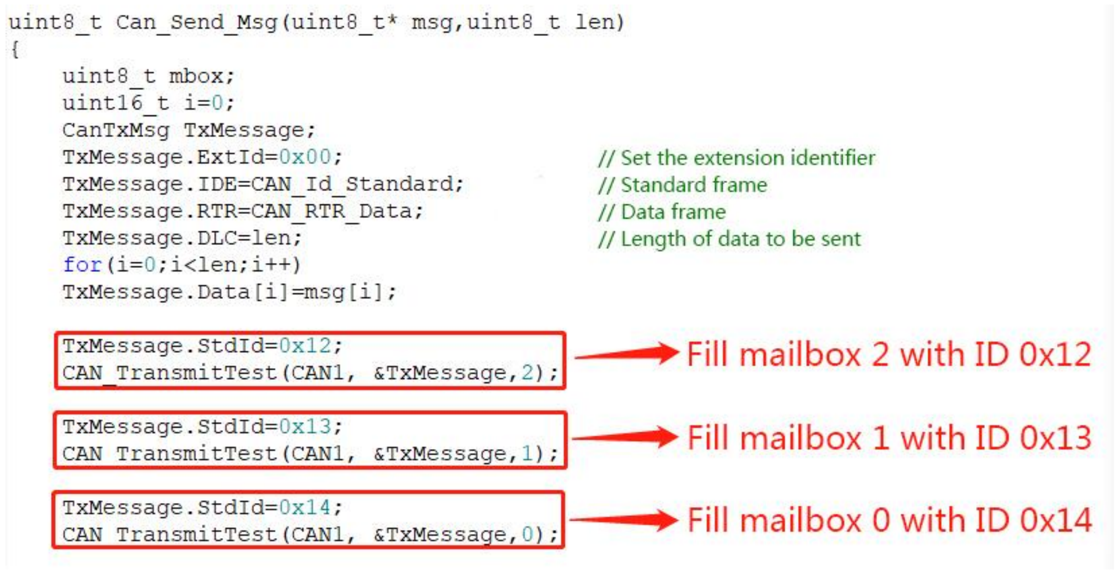
```
//CAN unit Settings
CAN_InitStructure.CAN_TTCM=DISABLE;         // Non-time triggered communication mode
CAN_InitStructure.CAN_ABOM=DISABLE;         // Software automatically offline management
CAN_InitStructure.CAN_AWUM=DISABLE;         // SLEEP mode awakened by software (clear sleep bit of CAN- >MCR)
CAN_InitStructure.CAN_NART=ENABLE;          // Disable automatic packet transmission
CAN_InitStructure.CAN_RFLM=DISABLE;         // The packet is not locked. The new packet overwrites the old one
CAN_InitStructure.CAN_TXFP=ENABLE;
CAN_InitStructure.CAN_Mode= mode;           // mode Settings: mode:0, common mode; 1, loop mode;
// Set the baud rate
CAN_InitStructure.CAN_SJW=tsjw;
CAN_InitStructure.CAN_BS1=tbs1;
CAN_InitStructure.CAN_BS2=tbs2;
CAN_InitStructure.CAN_Prescaler=brp;
CAN_Init(CAN1, &CAN_InitStructure);
```

The interface for sending packets can be defined as follows:



```c
void CAN_TransmitTest(CAN_TypeDef* CANx, CanTxMsg* TxMessage,uint8_t TxMailbox)
{
    uint8_t transmit_mailbox = 0;
    transmit_mailbox = TxMailbox;

    CANx->sTxMailBox[transmit_mailbox].TIR &= 1;
    if (TxMessage->IDE == CAN_Id_Standard)
    {
      CANx->sTxMailBox[transmit_mailbox].TIR |= ((TxMessage->StdId << 21) | \
                                                  TxMessage->RTR);
    }
    else
    {
      CANx->sTxMailBox[transmit_mailbox].TIR |= ((TxMessage->ExtId << 3) | \
                                                  TxMessage->IDE | \
                                                  TxMessage->RTR);
    }

    TxMessage->DLC &= (uint8_t)0x0000000F;
    CANx->sTxMailBox[transmit_mailbox].TDTR &= (uint32_t)0xFFFFFFF0;
    CANx->sTxMailBox[transmit_mailbox].TDTR |= TxMessage->DLC;

    CANx->sTxMailBox[transmit_mailbox].TDLR = (((uint32_t)TxMessage->Data[3] << 24) |
                                              ((uint32_t)TxMessage->Data[2] << 16) |
                                              ((uint32_t)TxMessage->Data[1] << 8) |
                                              ((uint32_t)TxMessage->Data[0]));
    CANx->sTxMailBox[transmit_mailbox].TDHR = (((uint32_t)TxMessage->Data[7] << 24) |
                                              ((uint32_t)TxMessage->Data[6] << 16) |
                                              ((uint32_t)TxMessage->Data[5] << 8) |
                                              ((uint32_t)TxMessage->Data[4]));

    CANx->sTxMailBox[transmit_mailbox].TIR |= 1;
}
```

The filling sequence of the sent packets is as follows:

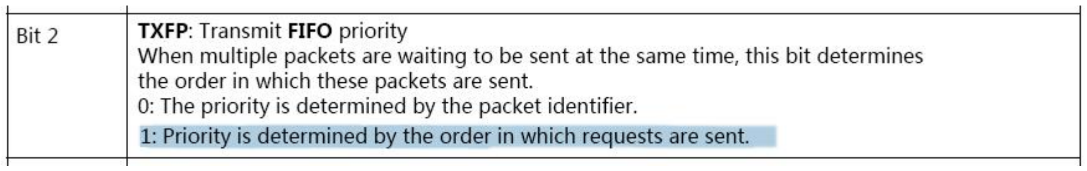


```c
uint8_t Can_Send_Msg(uint8_t* msg,uint8_t len)
{
    uint8_t mbox;
    uint16_t i=0;
    CanTxMsg TxMessage;
    TxMessage.ExtId=0x00;                  // Set the extension identifier
    TxMessage.IDE=CAN_Id_Standard;         // Standard frame
    TxMessage.RTR=CAN_RTR_Data;            // Data frame
    TxMessage.DLC=len;                     // Length of data to be sent
    for(i=0;i<len;i++)
    TxMessage.Data[i]=msg[i];

    TxMessage.StdId=0x12;
    CAN_TransmitTest(CAN1, &TxMessage,2);

    TxMessage.StdId=0x13;
    CAN_TransmitTest(CAN1, &TxMessage,1);

    TxMessage.StdId=0x14;
    CAN_TransmitTest(CAN1, &TxMessage,0);
```

Fill mailbox 2 with ID 0x12
Fill mailbox 1 with ID 0x13
Fill mailbox 0 with ID 0x14

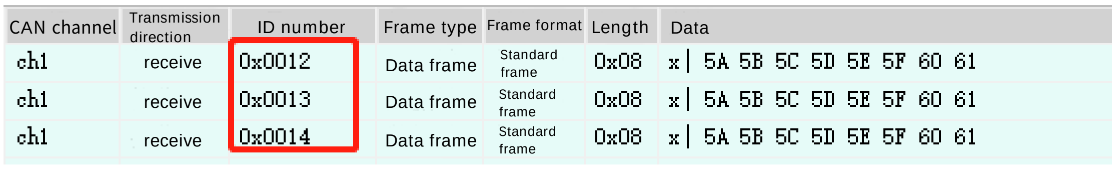
CAN packets are actually sent

For SXX32F103:
If the TXFP value is 1 and multiple packets are to be sent, the order of sending packets is determined by the order of requests.



Bit 2	**TXFP**: Transmit **FIFO** priority When multiple packets are waiting to be sent at the same time, this bit determines the order in which these packets are sent. 0: The priority is determined by the packet identifier. 1: Priority is determined by the order in which requests are sent.

The sequence of messages is as follows:

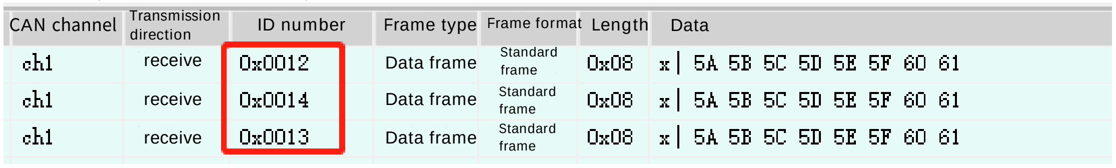


CAN channel	Transmission direction	ID number	Frame type	Frame format	Length	Data
ch1	receive	0x0012	Data frame	Standard frame	0x08	x| 5A 5B 5C 5D 5E 5F 60 61
ch1	receive	0x0013	Data frame	Standard frame	0x08	x| 5A 5B 5C 5D 5E 5F 60 61
ch1	receive	0x0014	Data frame	Standard frame	0x08	x| 5A 5B 5C 5D 5E 5F 60 61

For MH2103C:

When the TXFP value is 1 and multiple packets are to be sent, the order of sending packets is determined by the priority of the mailbox number. Email priority: Email 0> Email 1> Email 2.

The sequence of messages is as follows:



CAN channel	Transmission direction	ID number	Frame type	Frame format	Length	Data
ch1	receive	0x0012	Data frame	Standard frame	0x08	x| 5A 5B 5C 5D 5E 5F 60 61
ch1	receive	0x0014	Data frame	Standard frame	0x08	x| 5A 5B 5C 5D 5E 5F 60 61
ch1	receive	0x0013	Data frame	Standard frame	0x08	x| 5A 5B 5C 5D 5E 5F 60 61

3.5 FLASH module

3.5.1 Precautions for invoking the FLASH_EraseOptionBytes() interface

Problem description:

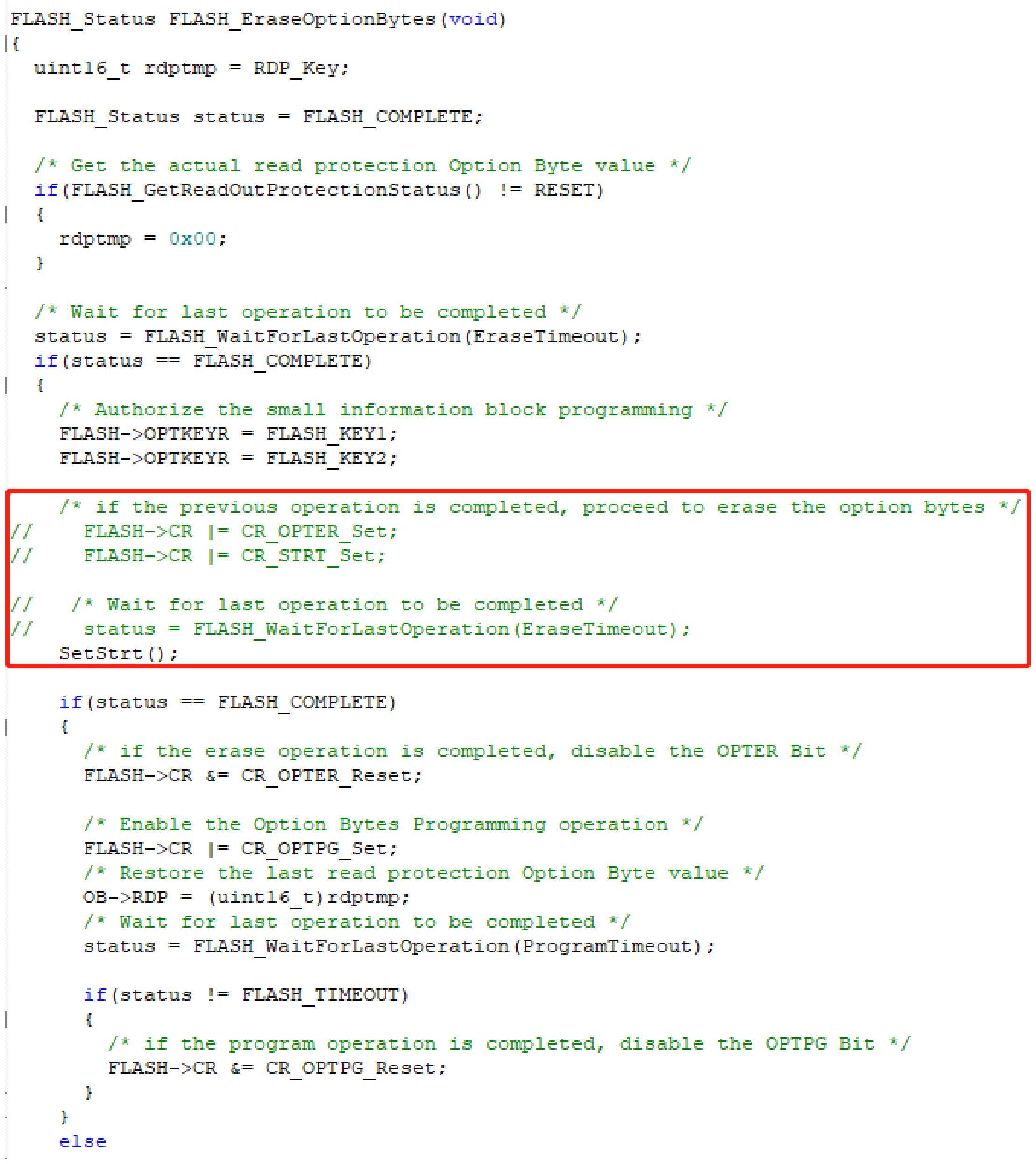
When the program uses the FLASH_EraseOptionBytes() interface to erase the option byte area, there is a probability that the HardFault_Handler exception will be interrupted.

Solution:

MH2103C Modifies the FLASH_EraseOptionBytes() interface as follows

Replace the following instructions with the SetStrt() interface

```
FLASH->CR |= CR_OPTER_Set;

FLASH->CR |= CR_STRT_Set;

FLASH_WaitForLastOperation(EraseTimeout);
```



```
FLASH_Status FLASH_EraseOptionBytes(void)
{
  uint16_t rdptmp = RDP_Key;

  FLASH_Status status = FLASH_COMPLETE;

  /* Get the actual read protection Option Byte value */
  if(FLASH_GetReadOutProtectionStatus() != RESET)
  {
    rdptmp = 0x00;
  }

  /* Wait for last operation to be completed */
  status = FLASH_WaitForLastOperation(EraseTimeout);
  if(status == FLASH_COMPLETE)
  {
    /* Authorize the small information block programming */
    FLASH->OPTKEYR = FLASH_KEY1;
    FLASH->OPTKEYR = FLASH_KEY2;

    /* if the previous operation is completed, proceed to erase the option bytes */
//    FLASH->CR |= CR_OPTER_Set;
//    FLASH->CR |= CR_STRT_Set;

//   /* Wait for last operation to be completed */
//    status = FLASH_WaitForLastOperation(EraseTimeout);
    SetStrt();

    if(status == FLASH_COMPLETE)
    {
      /* if the erase operation is completed, disable the OPTER Bit */
      FLASH->CR &= CR_OPTER_Reset;

      /* Enable the Option Bytes Programming operation */
      FLASH->CR |= CR_OPTPG_Set;
      /* Restore the last read protection Option Byte value */
      OB->RDP = (uint16_t)rdptmp;
      /* Wait for last operation to be completed */
      status = FLASH_WaitForLastOperation(ProgramTimeout);

      if(status != FLASH_TIMEOUT)
      {
        /* if the program operation is completed, disable the OPTPG Bit */
        FLASH->CR &= CR_OPTPG_Reset;
      }
    }
    else
```

The SetStrt() interface is shown in the following figure, which is implemented in mh32f10x_flash.c.

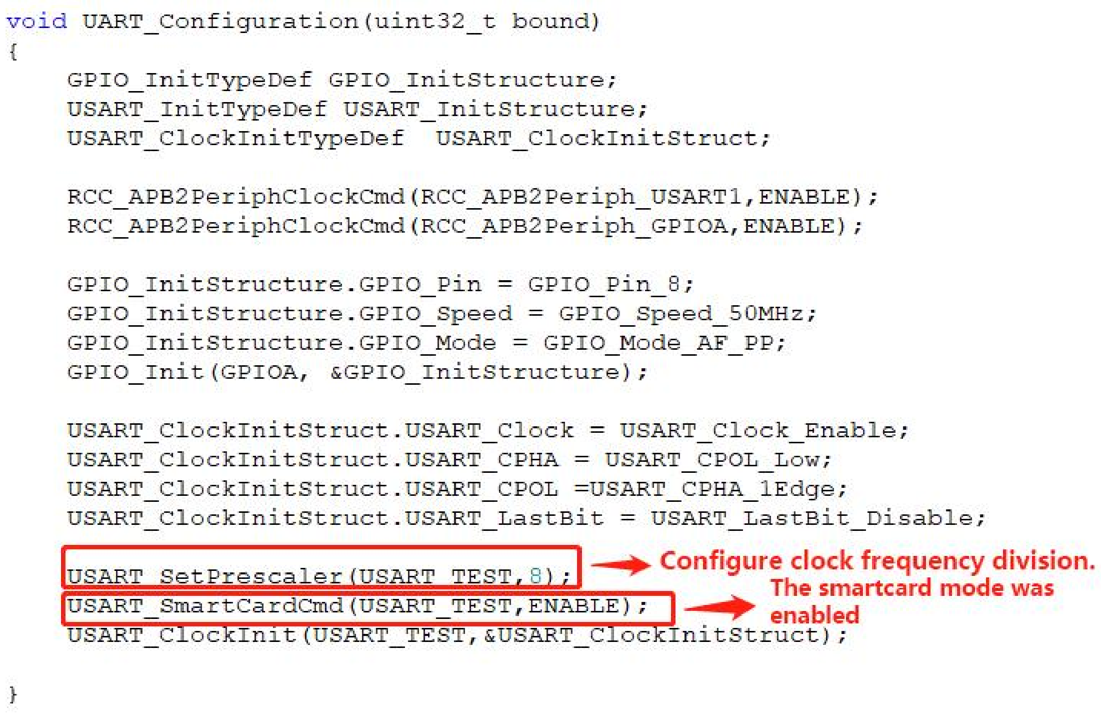
```
#if defined(__CC_ARM)
__ASM void SetStrt(void)
{
    MOV     R0, PC
    LDR     R1, [R0,#16]
    LDR     R1, [R0,#32]
    LDR     R0, =0x40022010
    LDR     R1, =0x60
    STR     R1,[R0]
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
FLAGLABLE
    LDR     R1, =0x4002200C
    LDR     R2, [R1]
    AND     R2, #0x01
    CMP     R2, #0x00
    BNE     FLAGLABLE
    BX      lr
}
#elif defined(__ICCARM__)
void SetStrt(void)
{
    __ASM("MOV      R0, PC\n"
          "LDR      R1, [R0,#16]\n"
          "LDR      R1, [R0,#32]\n"
          "LDR      R0, =0x40022010\n"
          "LDR      R1, =0x60\n"
          "STR      R1,[R0]\n"
          "NOP\n"
          "NOP\n"
          "NOP\n"
          "NOP\n"
          "NOP\n"
          "NOP\n"
          "FLAGLABLE:\n"
          "LDR      R1, =0x4002200C\n"
          "LDR      R2, [R1]\n"
          "AND      R2, R2, #0x01\n"
          "CMP      R2, #0x00\n"
          "BNE      FLAGLABLE\n"
          "BX       lr");
}
#elif defined(__GNUC__)
void SetStrt(void)
{
    asm("MOV        R0, PC");
    asm("LDR        R1, [R0,#16]");
    asm("LDR        R1, [R0,#32]");
```

3.6 USART module

3.6.1 Precautions for clock output in smart card mode

Problem description:

When only CLK is configured in smart card mode, there is no clock output. The configuration is as follows:



```
void UART_Configuration(uint32_t bound)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    USART_ClockInitTypeDef  USART_ClockInitStruct;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    USART_ClockInitStruct.USART_Clock = USART_Clock_Enable;
    USART_ClockInitStruct.USART_CPHA = USART_CPOL_Low;
    USART_ClockInitStruct.USART_CPOL =USART_CPHA_1Edge;
    USART_ClockInitStruct.USART_LastBit = USART_LastBit_Disable;

    USART_SetPrescaler(USART_TEST,8);
    USART_SmartCardCmd(USART_TEST,ENABLE);
    USART_ClockInit(USART_TEST,&USART_ClockInitStruct);

}
```

Configure clock frequency division.
The smartcard mode was enabled

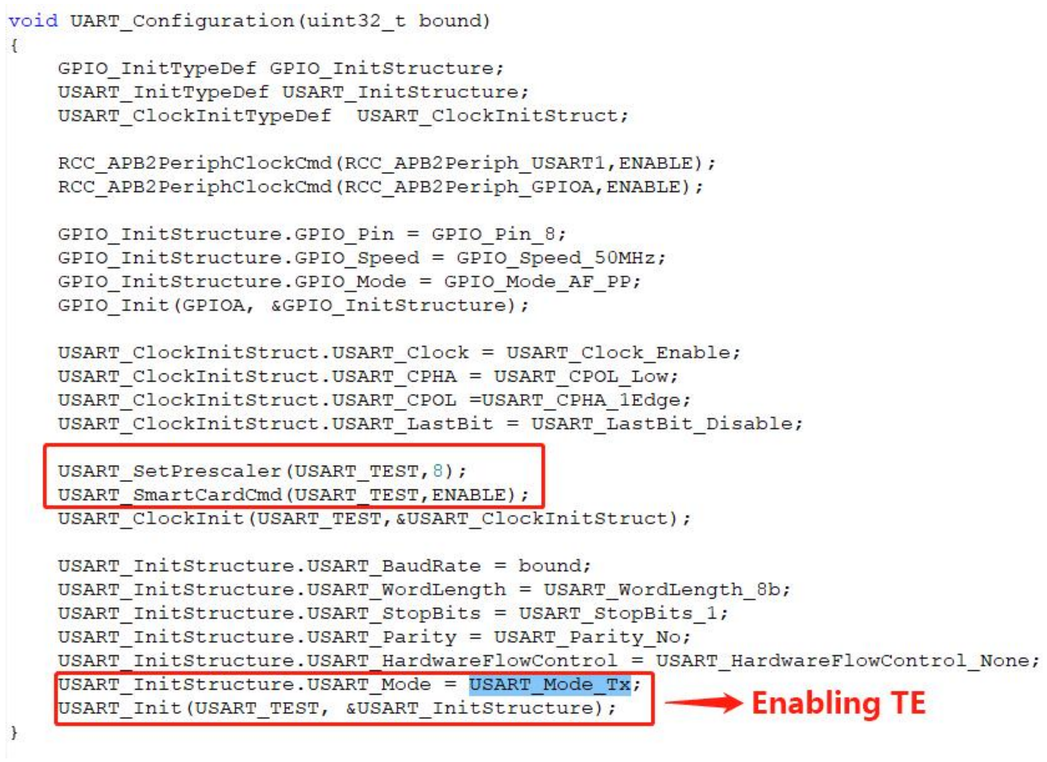
Root cause:

The clock output condition of the MH2103CUSART smart card mode is different from that of the SXX32F103, and is also controlled by the send enable TE.

Solution:

After the clock frequency division is configured and the smart card mode is enabled, set the transmit TE function to 1.

As shown in the picture below:



```
void UART_Configuration(uint32_t bound)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    USART_ClockInitTypeDef  USART_ClockInitStruct;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    USART_ClockInitStruct.USART_Clock = USART_Clock_Enable;
    USART_ClockInitStruct.USART_CPHA = USART_CPOL_Low;
    USART_ClockInitStruct.USART_CPOL =USART_CPHA_1Edge;
    USART_ClockInitStruct.USART_LastBit = USART_LastBit_Disable;

    USART_SetPrescaler(USART_TEST,8);
    USART_SmartCardCmd(USART_TEST,ENABLE);
    USART_ClockInit(USART_TEST,&USART_ClockInitStruct);

    USART_InitStructure.USART_BaudRate = bound;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Tx;
    USART_Init(USART_TEST, &USART_InitStructure);
}
```

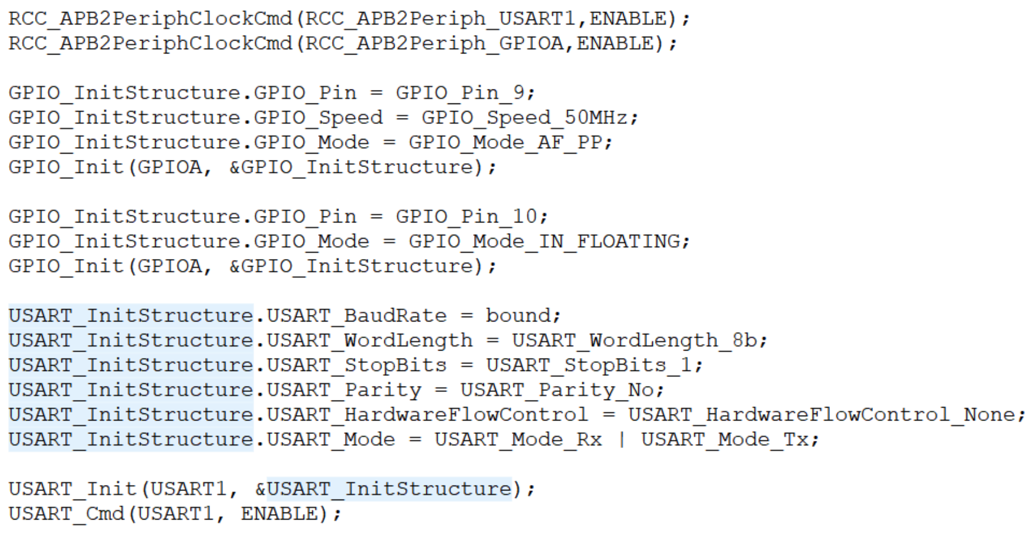
3.6.2 When DMA is used to send data, USARTTC flags are used to determine whether all data is sent. Precautions

Problem description:

When DMA is used to send data larger than 2Byte, the USARTTC flag is used to confirm whether the data is sent. You Need to proceed,

for example:

- ➢ USART1 is configured normally, and RE and TE are enabled. As shown in the following picture



```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);

USART_InitStructure.USART_BaudRate = bound;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

USART_Init(USART1, &USART_InitStructure);
USART_Cmd(USART1, ENABLE);
```

- Configure and enable the DMA channel to carry data. As shown in the following picture

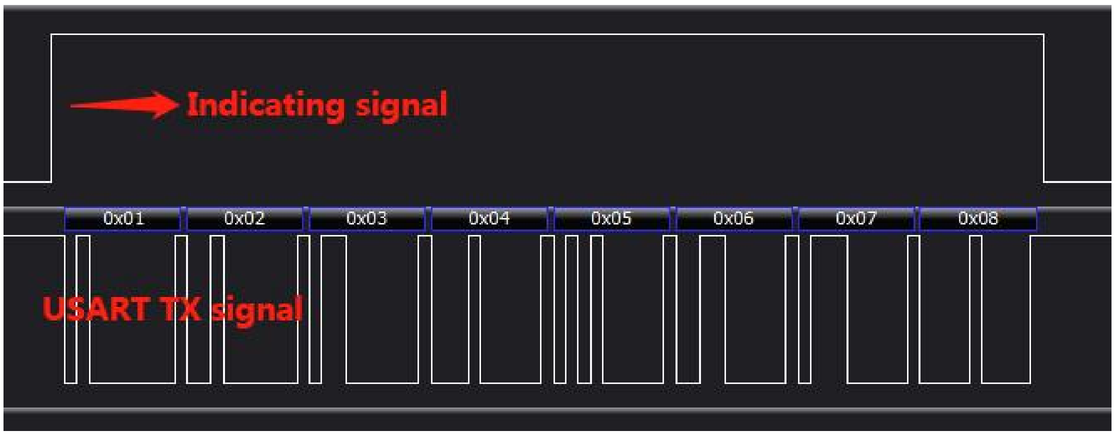
```
DMA_DeInit(DMA1_Channel4);
DMA_InitStructure.DMA_PeripheralBaseAddr = (u32)&USART1->DR;
DMA_InitStructure.DMA_MemoryBaseAddr     = (u32)pdat;
DMA_InitStructure.DMA_DIR                = DMA_DIR_PeripheralDST;
DMA_InitStructure.DMA_BufferSize         = length;
DMA_InitStructure.DMA_PeripheralInc      = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc          = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
DMA_InitStructure.DMA_MemoryDataSize     = DMA_PeripheralDataSize_Byte;
DMA_InitStructure.DMA_Mode               = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority           = DMA_Priority_Low;
DMA_InitStructure.DMA_M2M                = DMA_M2M_Disable;
DMA_Init(DMA1_Channel4, &DMA_InitStructure);

USART_DMACmd(USART1, USART_DMAReq_Tx, ENABLE);
DMA_Cmd(DMA1_Channel4, ENABLE);
```

- After DMATC is set, determine the USARTTC flag. As shown in the following picture

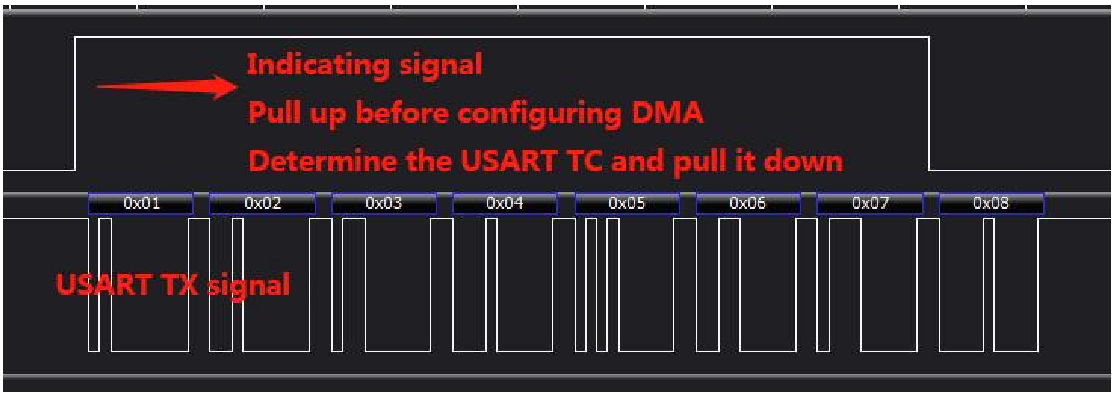
```
while(DMA_GetFlagStatus(DMA1_FLAG_TC4) != SET);
while(USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET);
```

- Detect indicating signals and data sent by USARTTX pins. As shown in the following picture

For SXX32F103:

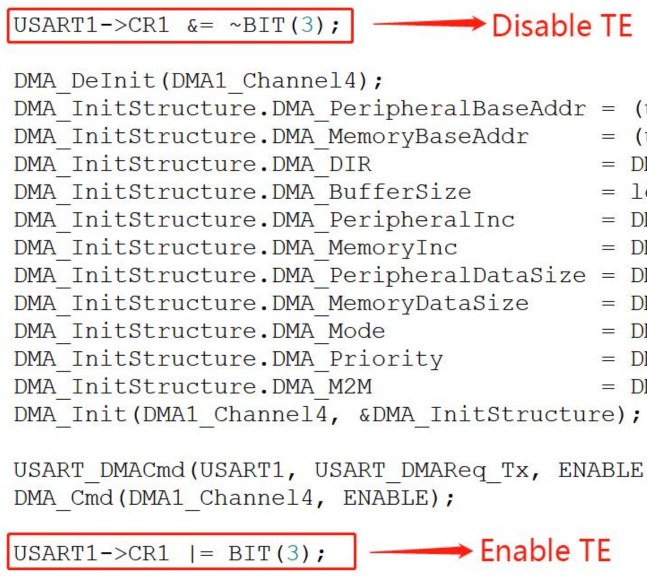


For MH2103C:


Indicating signal
Pull up before configuring DMA
Determine the USART TC and pull it down
0x01
0x02
0x03
0x04
0x05
0x06
0x07
0x08
USART TX signal

Solution:

Before configuring DMA, disable TE. After DMA is configured, TE is enabled to send data. As shown in the following picture



```
USART1->CR1  &= ~BIT(3);

DMA_DeInit(DMA1_Channel4);
DMA_InitStructure.DMA_PeripheralBaseAddr = (u32)&USART1->DR;
DMA_InitStructure.DMA_MemoryBaseAddr     = (u32)pdat;
DMA_InitStructure.DMA_DIR                = DMA_DIR_PeripheralDST;
DMA_InitStructure.DMA_BufferSize         = length;
DMA_InitStructure.DMA_PeripheralInc      = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc          = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
DMA_InitStructure.DMA_MemoryDataSize     = DMA_PeripheralDataSize_Byte;
DMA_InitStructure.DMA_Mode               = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority           = DMA_Priority_Low;
DMA_InitStructure.DMA_M2M                = DMA_M2M_Disable;
DMA_Init(DMA1_Channel4, &DMA_InitStructure);

USART_DMACmd(USART1, USART_DMAReq_Tx, ENABLE);
DMA_Cmd(DMA1_Channel4, ENABLE);

USART1->CR1  |= BIT(3);
```

3.6.3 Smart card & Single-wire half-duplex mode TX pin configuration precautions

Problem description:

In smart card & single-wire half-duplex mode, when the TX pin is configured to multiplexed open-leak mode, the TX output is always low. The I/O mode is configured as follows:

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;
```

Root cause:

Due to the different design, the smart card & single-wire half-duplex mode of the MH2103CUSART requires the TX pin to be configured as a multiplexed push-pull mode.

Solution:

In smart card & single-wire half-duplex mode, the TX pin is configured as the push-pull mode, which can be used normally.
The I/O mode is configured as follows:

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
```

3.6.4 Precautions for DR Overflow when accepting data

Problem description:

When RXNE is set to 1, it is not reset (DR Is not read), and one or more

characters are received, RXNE is abnormal.

Root cause:

The design is different.

Solution:

Software is designed to avoid DR Overflow. You can use the following methods:

- ➢ Use interrupt /DAM mode to obtain data to ensure that the DR Can be read in time after RXNE is set.
- ➢ When the RXNE reads data, ensure that the CPU detects the RXNE status in real time when the data is received.

3.7 SPI module

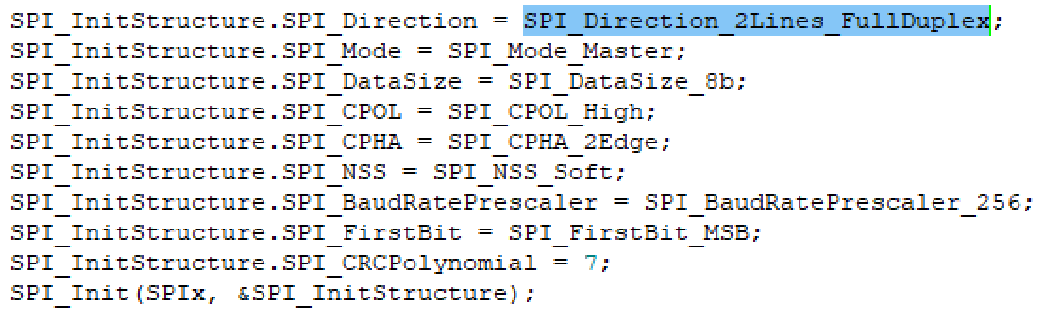
3.7.1 As the Master, this section describes the precautions for sending and receiving data in full-duplex mode

Problem description:

When communicating with SPI as Master, the received data is abnormal.

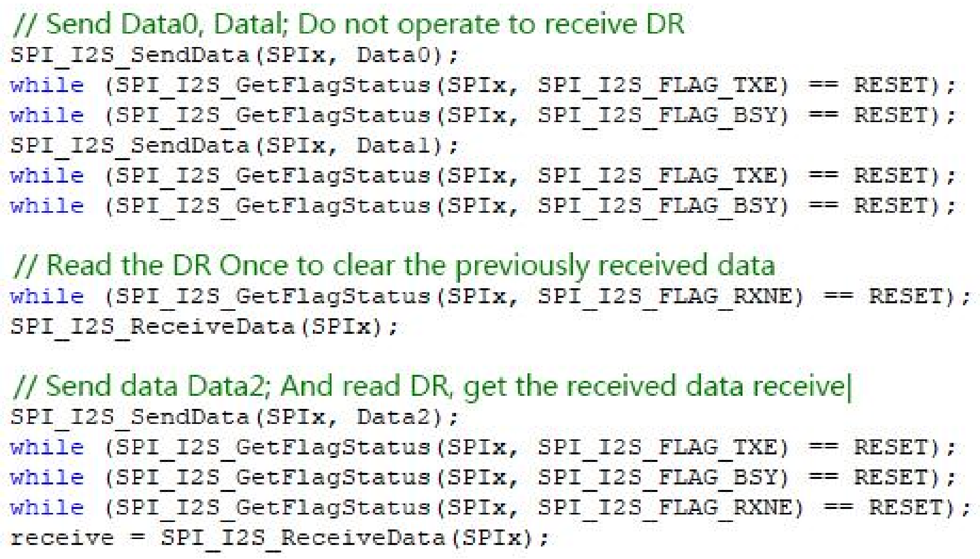
For example:

- ➢ The SPI is configured in Master two-wire full-duplex mode. As shown in the following picture



```
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);
```

- ➢ Send and receive data according to the following figure.



```
// Send Data0, Datal; Do not operate to receive 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);

// Read the DR Once to clear the previously received data
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_RXNE) == RESET);
SPI_I2S_ReceiveData(SPIx);

// Send data Data2; And read DR, get the received data 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);
```

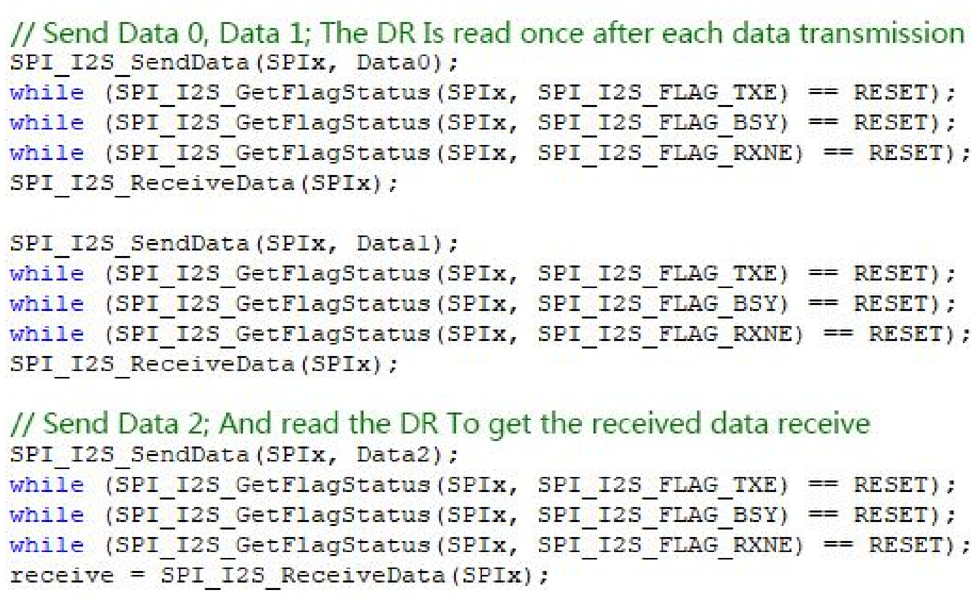
MH2103C sends Data2 The received data is abnormal data.

Root cause:

The MH2103CSPI has a 16Byte receiving FIFO. The data received by Data1 is cached in the receiving FIFO, and the operation of reading the DR Once does not clear all the useless data in the FIFO, resulting in the data received by the subsequent sending Data2 is not the expected data.

Solution:

As a Master, the DR Is read every time the data is sent, ensuring that no residual data in the FIFO can be resolved.
As shown in the picture below:

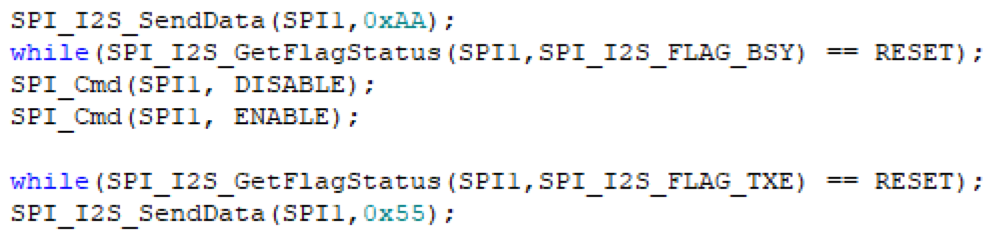


```
// Send Data 0, Data 1; The DR Is read once after each data transmission
SPI_I2S_SendData(SPIx, Data0);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_TXE) == RESET);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_BSY) == RESET);
while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_RXNE) == RESET);
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);

// Send Data 2; And read the DR To get the received data 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);
```

3.7.2 Turn off the SPE in BUSY state

Problem description:

When the SPI is operated, if the software does not determine the non-busy status of the SPE when it is turned off, data may fail to be sent and received. Operation as shown below:
The MH2103C can normally send 0xAA, but cannot send 0x55.



```
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);
```

Root cause:

MH2103CSPI When the SPE device is in BUSY state, disabling the SPE device takes effect after data sending is complete (in non-busy state). Ignoring the enabling action does not take effect.

Solution:

Before shutting down the SPE, check whether the SPE is in BUSY state.

You can shut down the SPE only when the SPE is not BUSY.

3.7.3 Precautions for Data Transmission Using DMA

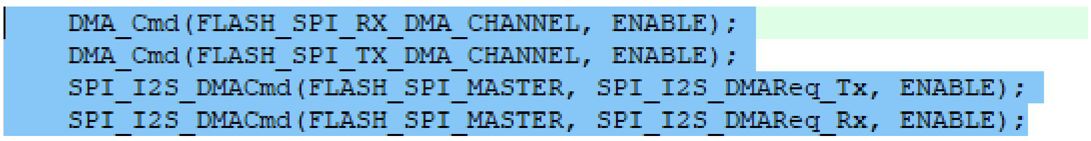
Problem description:

When the SPI uses DMA to transmit data, the data received is incorrect.

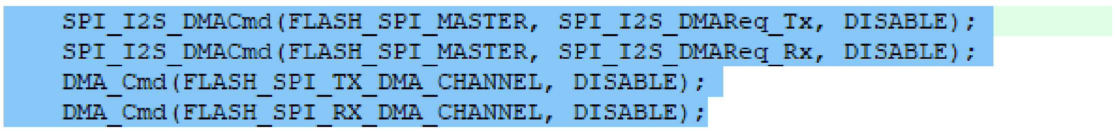
Solution:

When SPI uses DMA to transmit data, it is recommended that peripheral DAM

and DMA channel be enabled/disabled at the same time.
As shown in the picture below:



```
    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);
```

3.8 USB module

3.8.1 USB use precautions

Problem description:

When USB is configured in Slave mode, the Host sends requests from different endpoints or from the same endpoint in different directions in a short period of time, and there is a probability of crash.

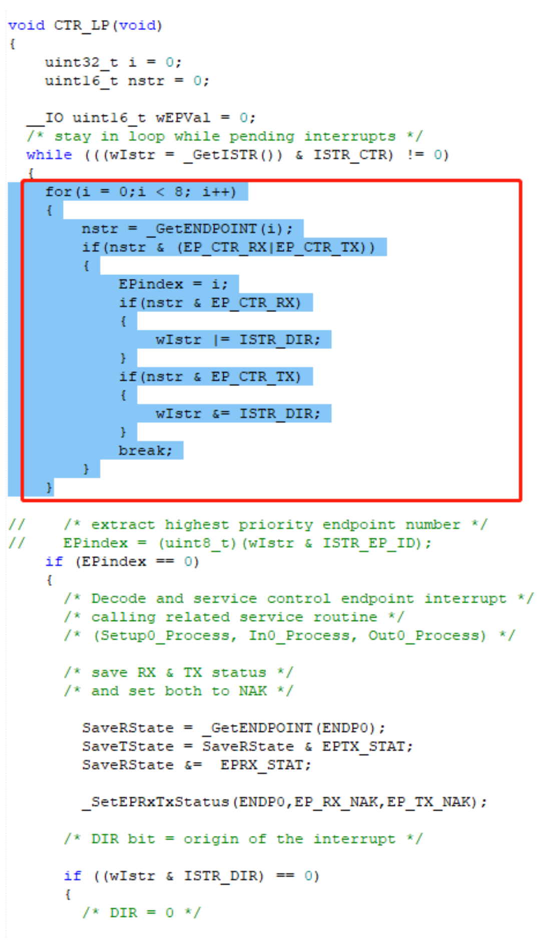
Root cause:

The EP_ID and DIR update mechanism in the USB_ISTR(Interrupt status

register) of the MH2103C USB is different from that of the SXX32F103.

Solution:

Modify the CTR_LP() interface, as shown in the following figure. After modification, it is compatible with SXX32F103.



```
void CTR_LP(void)
{
    uint32_t i = 0;
    uint16_t nstr = 0;

  __IO uint16_t wEPVal = 0;
  /* stay in loop while pending interrupts */
  while (((wIstr = _GetISTR()) & ISTR_CTR) != 0)
  {
    for(i = 0;i < 8; i++)
    {
        nstr = _GetENDPOINT(i);
        if(nstr & (EP_CTR_RX|EP_CTR_TX))
        {
            EPindex = i;
            if(nstr & EP_CTR_RX)
            {
                wIstr |= ISTR_DIR;
            }
            if(nstr & EP_CTR_TX)
            {
                wIstr &= ISTR_DIR;
            }
            break;
        }
    }
//    /* extract highest priority endpoint number */
//    EPindex = (uint8_t)(wIstr & ISTR_EP_ID);
    if (EPindex == 0)
    {
      /* Decode and service control endpoint interrupt */
      /* calling related service routine */
      /* (Setup0_Process, In0_Process, Out0_Process) */

      /* save RX & TX status */
      /* and set both to NAK */

        SaveRState = _GetENDPOINT(ENDP0);
        SaveTState = SaveRState & EPTX_STAT;
        SaveRState &=  EPRX_STAT;

        _SetEPRxTxStatus(ENDP0,EP_RX_NAK,EP_TX_NAK);

      /* DIR bit = origin of the interrupt */

      if ((wIstr & ISTR_DIR) == 0)
      {
        /* DIR = 0 */
```

3.9 DMA module

3.9.1 DMA usage precautions

Problem description:

When data is being transferred by a Channel in the DMA module, modifying the CNT value (data transfer quantity) of another Channel may not take effect.

Root cause:

The CNT(Data Transfer quantity) modification of MH2103CDMAChannel takes effect under different conditions from SXX32F103.

Solution:

The software prevents the DMA Channel from modifying the CNT value (the amount of data transferred) of another Channel while it is moving data.

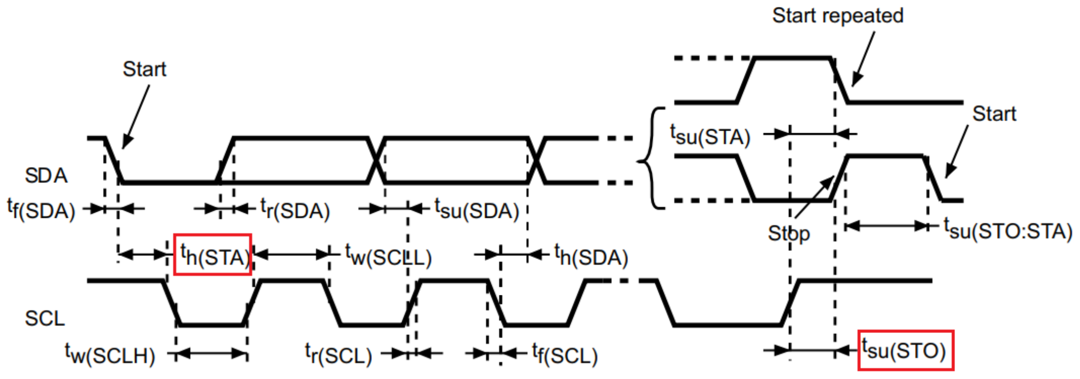
3.10 I I C module

3.10.1 Send start bit & Stop bit as Master Note

Problem description:

As shown in the figure below MH2103C start bit Hold time ($t_{h(STA)}$)&stop bit Hold time($t_{SU(STO)}$)and SXX32F103 is different.

When the peer end is an I/O analog Slave, you need to confirm whether the peer end has requirements for $t_{h(STA)}$ and $t_{SU(STO)}$.



For SXX32F103:

$t_{h(STA)}$ and $t_{SU(STO)}$equal the high level time of the SCL.

For MH2103C:

In standard mode, $t_{h(STA)}$&$t_{SU(STO)}$ equals (FREQ*4+8)*T_{APB1}。 In fast mode, $t_{h(STA)}$&$t_{SU(STO)}$ equals (FREQ+8)*T_{APB1}。

Root cause:

Different design.

Solution:

If you have requirements for $t_{h(STA)}$ and $t_{SU(STO)}$ time, you can adjust the time by modifying FREQ or modifying APB1CLK.

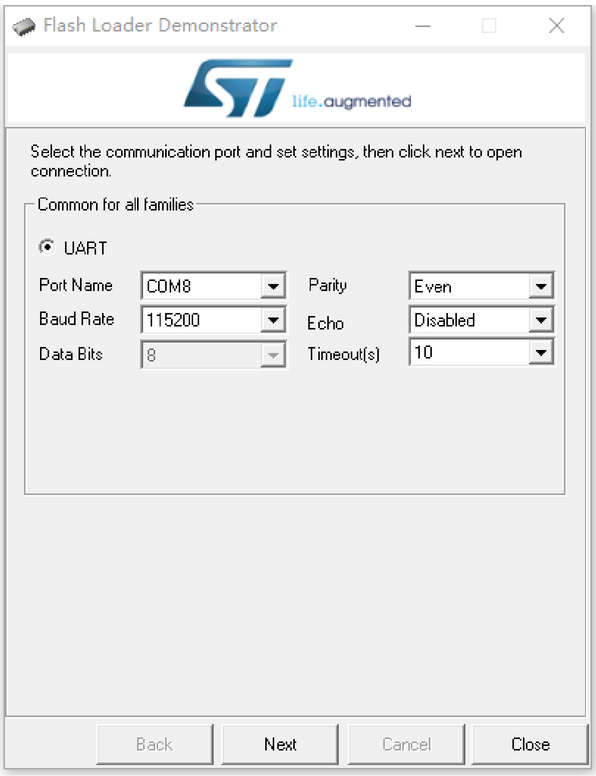
4. ISP, emulator, offline burner use precautions

4.1 ISP tool usage Precautions

MH2103C supports the ISP tools commonly used in the market.

The programmer includes STMFlashLoader, FlyMcu, and STM32CubeProgrammer.

STMFlashLoader：


Flash Loader Demonstrator

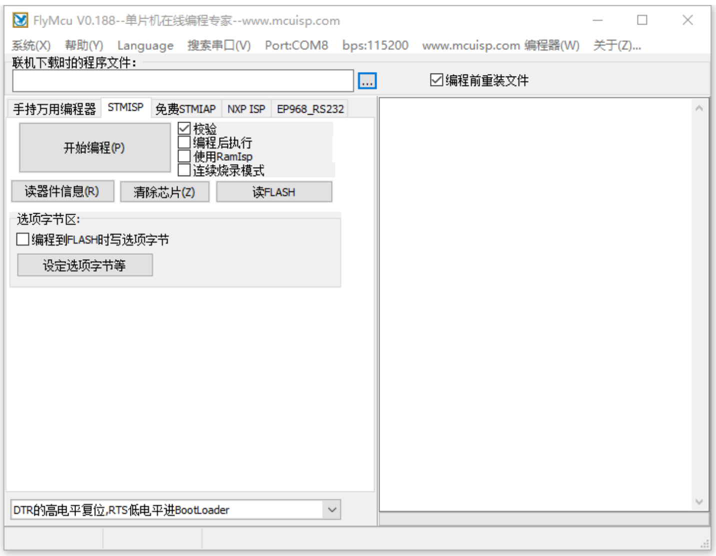
Select the communication port and set settings, then click next to open connection.

Common for all families

UART

Port Name	COM8	Parity	Even
Baud Rate	115200	Echo	Disabled
Data Bits	8	Timeout(s)	10

FlyMcu：


FlyMcu V0.188--单片机在线编程专家--www.mcuisp.com

系统(X) 帮助(Y) Language 搜索串口(V) Port:COM8 bps:115200 www.mcuisp.com 编程器(W) 关于(Z)...

联机下载时的程序文件：

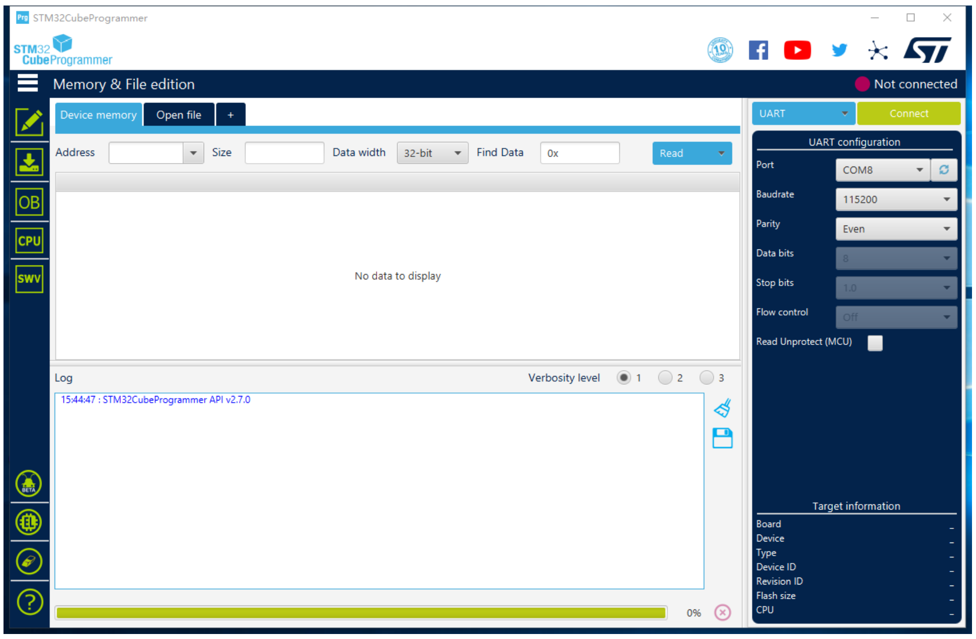
☑编程前重装文件

☑校验
☐编程后执行
☐使用RamIsp
☐连续烧录模式

选项字节区：
☐编程到FLASH时写选项字节

DTR的高电平复位,RTS低电平进BootLoader

STM32CubeProgrammer：



UART configuration

Port COM8

Baudrate 115200

Parity Even

Data bits 8

Stop bits 1.0

Flow control Off

Read Unprotect (MCU)

Data width 32-bit

Find Data 0x

No data to display

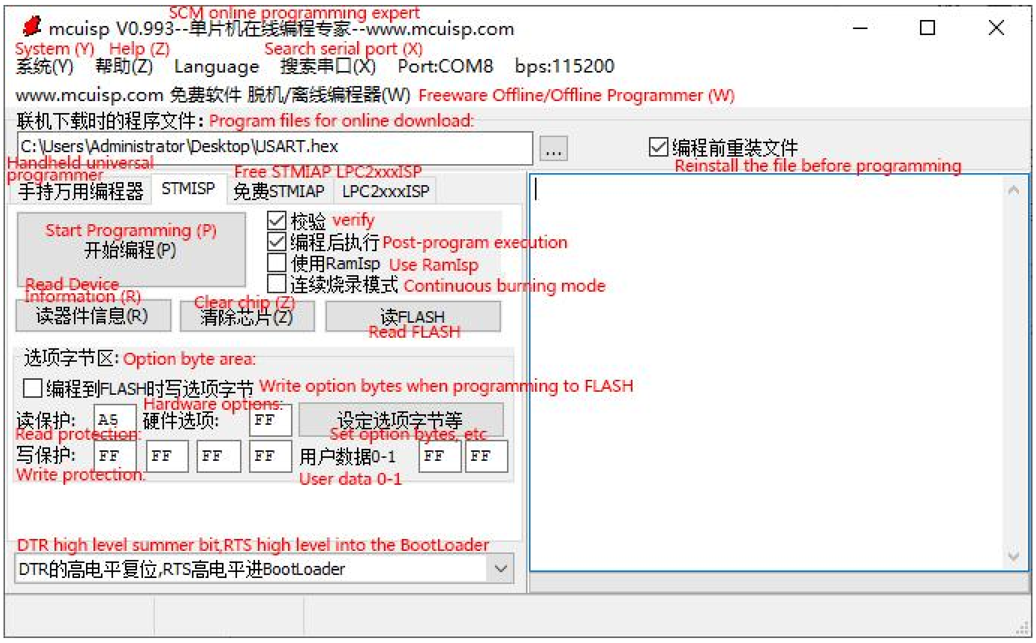
Log

Verbosity level ● 1 ○ 2 ○ 3

15:44:47 : STM32CubeProgrammer API v2.7.0

0%

MH2103C does not support mcuisp V0.993.



Port:COM8 bps:115200

联机下载时的程序文件: Program files for online download:

C:\Users\Administrator\Desktop\USART.hex

☑ 编程前重装文件 Reinstall the file before programming

☑ 校验 verify

☑ 编程后执行 Post-program execution

☐ 使用RamIsp Use RamIsp

☐ 连续烧录模式 Continuous burning mode

选项字节区: Option byte area:

☐ 编程到FLASH时写选项字节 Write option bytes when programming to FLASH

读保护: A5 Read protection

Hardware options: 硬件选项: FF

写保护: FF FF FF FF Write protection

用户数据0-1 FF FF User data 0-1

DTR high level summer bit,RTS high level into the BootLoader

DTR的高电平复位,RTS高电平进BootLoader

4.2 Precautions for using emulator

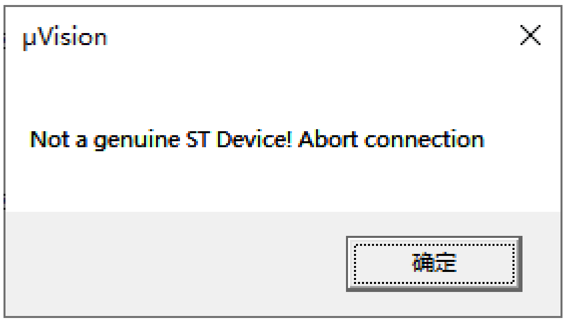
Emulators supported by MH2103C include ST-LINKV2, JLINK, ARM Emulator, etc.

The MH2103C does not support ST-LINKV3.

4.2.1 Precautions for using the Keil.SXX32F1xx_DFP.2.3.0 or later pack

Problem description:

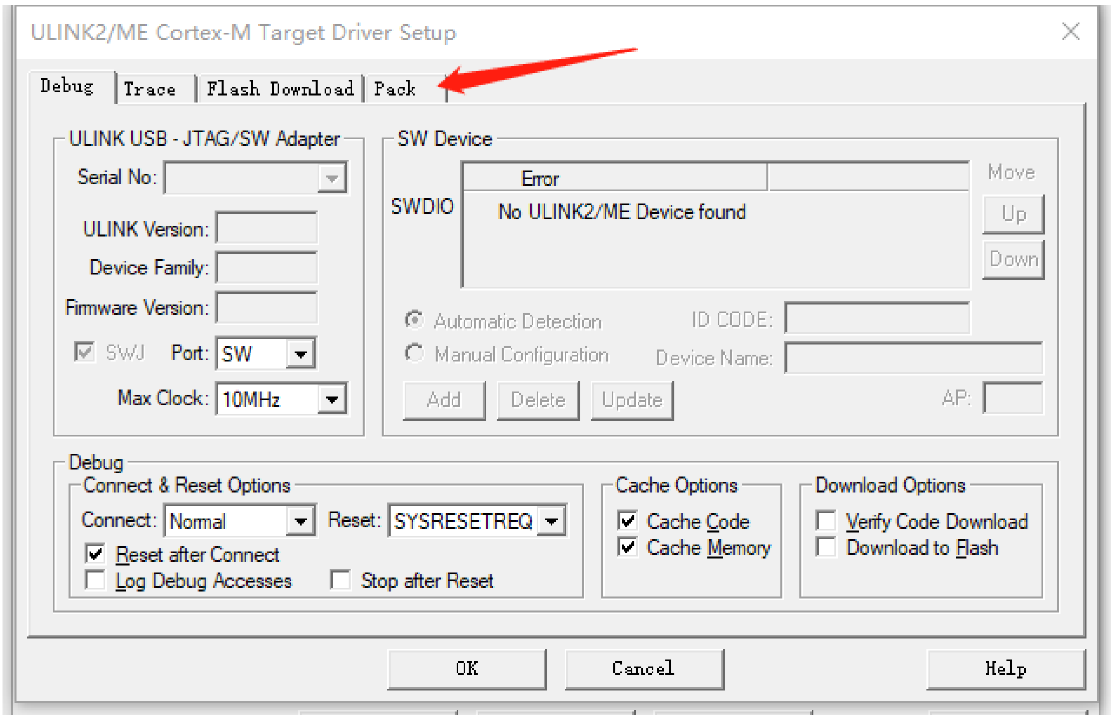
When using ULINK2 or CMSIS-DAP emulator and using pack package Keil. SXX32F1xx_DFP.2.3.0 or later (including version 2.3.0), the program cannot be downloaded. The following information is displayed during the download.



Not a genuine ST Device! Abort connection

Solution:

Deselect the enable option in pack, as shown in the following figure.



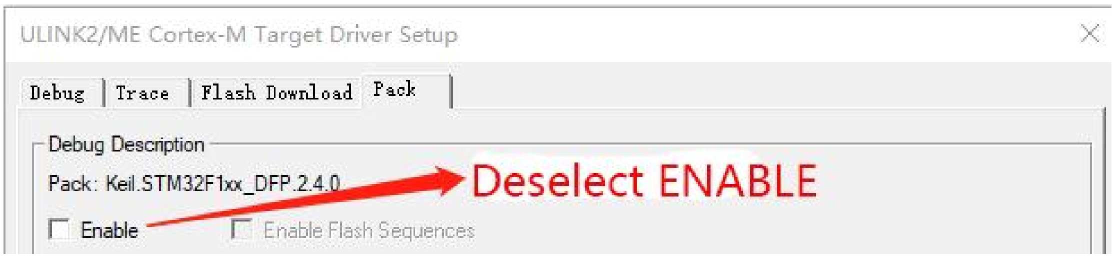
ULINK USB - JTAG/SW Adapter
Serial No:
ULINK Version:
Device Family:
Firmware Version:
SWJ Port: SW
Max Clock: 10MHz

SW Device
SWDIO Error: No ULINK2/ME Device found
Automatic Detection ID CODE:
Manual Configuration Device Name:
AP:

Debug
Connect & Reset Options
Connect: Normal Reset: SYSRESETREQ
☑ Reset after Connect
☐ Log Debug Accesses ☐ Stop after Reset

Cache Options
☑ Cache Code
☑ Cache Memory

Download Options
☐ Verify Code Download
☐ Download to Flash



Debug Description
Pack: Keil.STM32F1xx_DFP.2.4.0
☐ Enable ☐ Enable Flash Sequences

4.3 Notes on the use of downloader

At present, there are many downloader manufacturers on the market, and MH2103C fully supports the downloader: Jindiao offline downloader

GOLDEN EAGLE
产品指示图
深耕行业多年 致力于成为中国编程器领导者
MCU型号 MCU MODEL
序列号
基本配置
进度显示栏
状态栏
一键下载
中文镜像名称
下载次数统计
电压监控
镜像管理
LED指示 BOOT KEY
GOLDEN
STM32F103ZET6
总共次数: 1000 剩余: 996
芯片加密
VOUT
芯片擦除
复位运行
V-5V
结束断电
MCU连接成功..
解除读保护..
擦除FLASH..
编写FLASH..
FLASH校验..
序列号编写成功..
读保护设置..
MCU复位运行..
3.30V
镜像: 03

The MH2103C does not support SmartPRO T9000-PLUS and XELTEKSUPERPRO 6100N.

4.3.1 When the chip is in the read/write protection state, burn precautions

Problem description:

Using positive atom MINI, positive atom P100 and other off-line burner, the chip with read and write protection was burned, and the burning failure occurred.

Solution:

After the failure, power on or off the MCU again or Reset the external reset. Burn again to burn successfully.

Note: If the chip is empty, using the above burner, there will be no burning failure problem.

4.3.2 Precautions for using WizPro200ST8 programmer

Problem description:

The burning failure occurred with the WizPro200ST8 programmer.

Solution:

Disconnect the Reset pin and chip of WizPro200ST8 programmer; Only connected VDD, SWCLK, SWDIO can be burned normally.

Historical version

Edition	Alter	Time
1.00	Initial version	
1.01	Added notes related to system functions, FLASH, USART, SPI/IIS	
1.02	Added SPI related precautions	
1.03	Added ADC and TIM related notes	
1.04	Added precautions for non-32bit alignment access to the APB bus	
1.05	Added Precautions for interrupt controller	
1.06	Added notes for different modes of dual ADCs	
1.07	Added notes for ISP, emulator, and downloader	
1.08	Added the precautions for hardware migration	
1.09	Add precautions for the use of system modules, CAN and DMA	
1.10	Added USART smart card & Single wire Half duplex mode use precautions	
1.11	Added TIM, IIC use precautions	