XHSC 小华半导体
XIAOHUA SEMICONDUCTOR

32 位微控制器

HC32 系列 I2S 从模式的帧同步方法

应用笔记

Rev1.00　2026 年 08 月

适用对象

产品系列	产品型号	产品系列	产品型号
HC32F451	ALL	**HC32F452**	ALL
HC32F460	ALL	**HC32A460**	ALL
HC32F467	ALL	**HC32F4A0**	ALL
HC32F4A2	ALL	**HC32A4A0**	ALL

声　明

★ 小华半导体有限公司（以下简称：“XHSC”）保留随时更改、更正、增强、修改小华半导体产品和/或本文档的权利，恕不另行通知。用户可在下单前获取最新相关信息。XHSC 产品依据购销基本合同中载明的销售条款和条件进行销售。

★ 客户应针对您的应用选择合适的 XHSC 产品，并设计、验证和测试您的应用，以确保您的应用满足相应标准以及任何安全、安保或其它要求。客户应对此独自承担全部责任。

★ XHSC 在此确认未以明示或暗示方式授予任何知识产权许可。

★ XHSC 产品的转售，若其条款与此处规定不同，XHSC 对此类产品的任何保修承诺无效。

★ 任何带有“®”或“™”标识的图形或字样是 XHSC 的商标。所有其他在 XHSC 产品上显示的产品或服务名称均为其各自所有者的财产。

★ 本通知中的信息取代并替换先前版本中的信息。

目　录

1 概述

本文档主要介绍小华半导体 HC32 系列芯片如何实现 I2S 从模式的帧同步，即对 WS 信号的同步。

2 I2S 功能介绍

I2S（Inter_IC Sound Bus），集成电路内置音频总线，该总线专责于音频设备之间的数据传输。

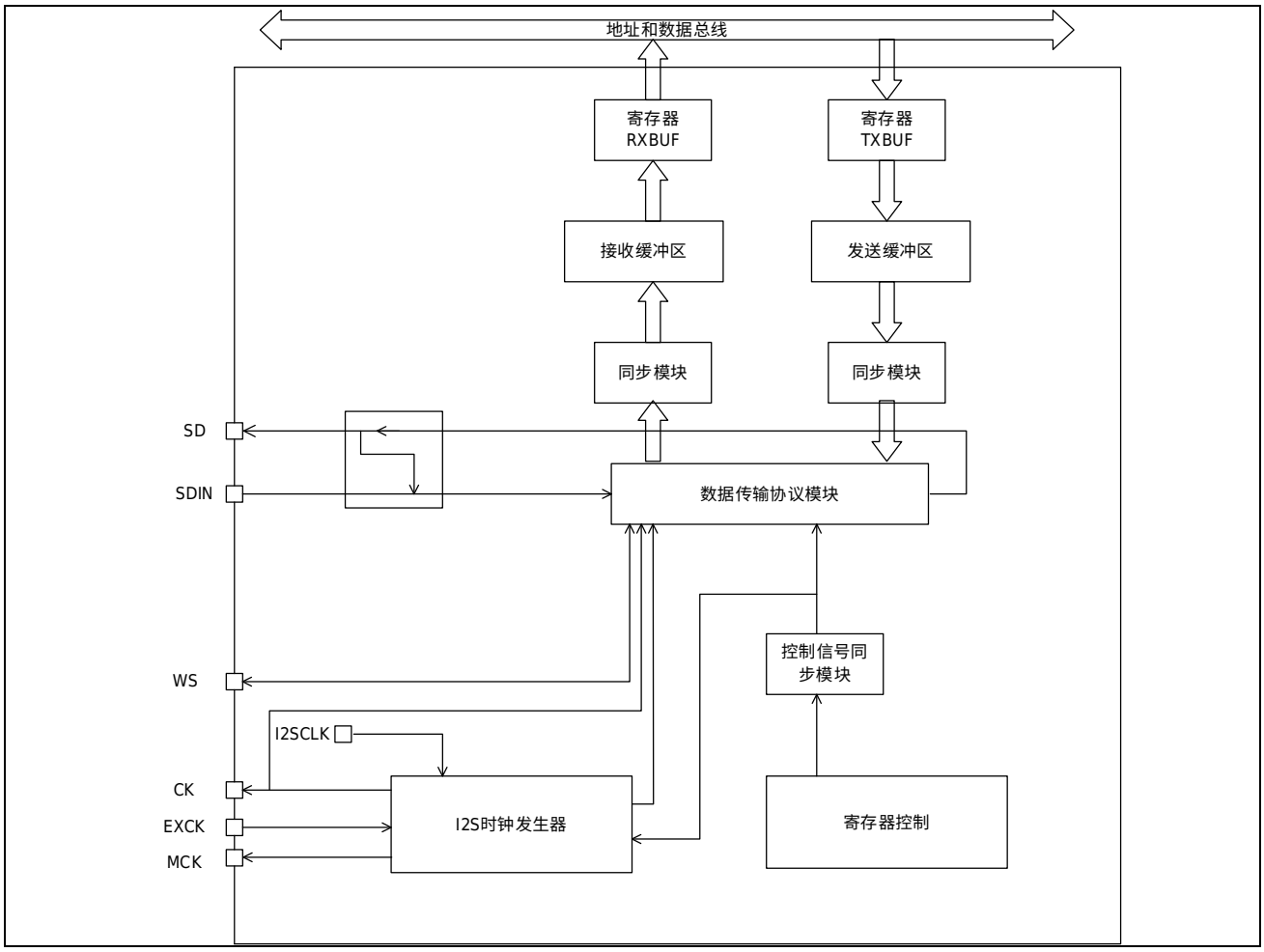


图 2-1 I2S 系统框图

2.1 I2S 主要特性

- 可使用内部 I2SCLK 时钟（PCLK、PLLx），也可使用 I2S_EXCK 引脚上的外部时钟
- 可输出驱动时钟以驱动外部音频元件，比特率固定为 256*Fs（Fs 为音频采样频率）
- 支持采样频率：192k、96k、48k、44.1k、32k、22.05k、16k、8k
- 支持 I2S 协议：I2S Philips 标准、MSB 对齐标准、LSB 对齐标准、PCM 标准

2.2 数据和帧格式

有四种数据和帧格式组合，可采用下列格式发送数据：

- 将 16 位数据封装在 16 位帧中
- 将 16 位数据封装在 32 位帧中
- 将 24 位数据封装在 32 位帧中

- 将 32 位数据封装在 32 位帧中

对于所有数据格式和通信标准而言，始终会先发送最高有效位（MSB 优先）。

2.3 I2S 主模式和从模式

I2S 通信中主模式器件是 CK 信号和 WS 信号的发送方，I2S 外设可以配置为主模式或者从模式。

管脚名	主模式		从模式	
	全双工	半双工	全双工	半双工
通讯时钟 I2S_CK	输出	输出	输入	输入
字选择 I2S_WS	输出	输出	输入	输入
串行数据 I2S_SD	输出	输入/输出	输出	输入/输出
全双工音频数据输入 I2S_SDIN	输入	-	输入	-

3 帧同步问题

以 I2S Philips 标准为例，主机持续输出 CK 与 WS。若从机在数据帧中的任意时刻使能 I2S，则将导致其与 WS 信号失步，产生帧错位数据。如图 3-1 所示，I2S 从机在时钟 CK 两个有效上升沿后再启动 I2S 功能，会导致从机收到或发出的数据有两个位的偏移。

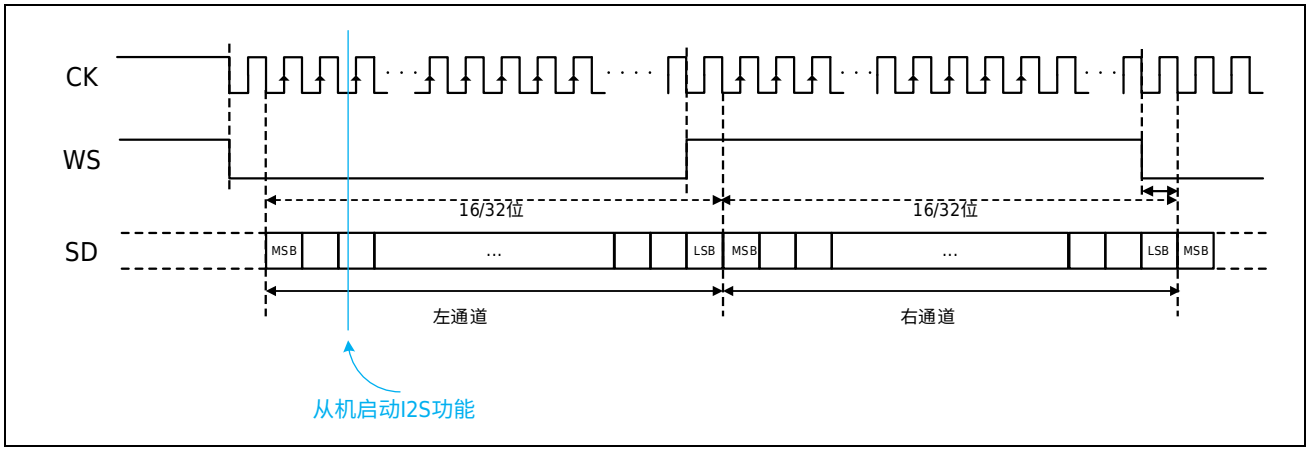


图 3-1 I2S Philips 协议波形（16/32 位全精度）

4 同步方案

本方案以 WS 信号的边沿作为触发条件来使能 I2S 从机。为确保启动动作的及时性与可靠性，利用 WS 引脚的外部中断功能，以硬件方式完成 I2S 控制寄存器的配置操作。

此方案中会使用到 GPIO 外部中断功能、AOS 功能和 DMA 数据搬运，这三个硬件模块协同工作实现无 CPU 干预的 I2S 自动同步使能。

4.1 GPIO 外部中断

HC32 系列 MCU 的任意 GPIO 引脚均可配置为外部中断源，支持上升沿、下降沿及双边沿触发模式，并且该引脚配置的周边功能可同时使用。

此方案中当 WS 边沿（上升沿或下降沿，取决于帧起始定义）到来时，GPIO 外设硬件自动触发中断请求或产生事件信号，无需软件轮询检测。

4.2 自动运行系统（AOS）

自动运行系统（Automatic Operation System）用于在不借助 CPU 的情况下实现外设硬件电路之间的联动。利用外设电路产生的事件作为 AOS 源（AOS Source），如引脚外部中断、定时器比较匹配、ADC 转换结束等，来触发其他外设电路动作。被触发的外设电路动作称为 AOS 目标（AOS Target）。

在本方案中，配置 AOS 将 GPIO 外部中断事件（AOS 源）路由至 DMA 控制器（AOS 目标），使 WS 边沿触发的事件直接启动 DMA 传输，无须进入 CPU 中断服务程序。

4.3 DMA

DMA 外设能够在 CPU 不参与的情况下实现存储器之间、存储器和外围功能模块之间以及外围功能模块之间的数据交换。

本方案中配置 DMA 目标地址为 I2S 控制寄存器 I2S_CTRL，位宽为 32 位。DMA 源地址为一个 32 位的变量地址，变量保存了计划写入控制寄存器的值。配置触发 DMA 搬运的条件为 WS 的上升沿或者下降沿，且 DMA 工作的传输次数为 1，当 WS 产生外部中断事件后，DMA 将控制寄存器的值搬运到 I2S_CTRL 寄存器，实现 I2S 的硬件启动。

4.4 同步方案框图

本方案的功能框图如图 4-1 所示。

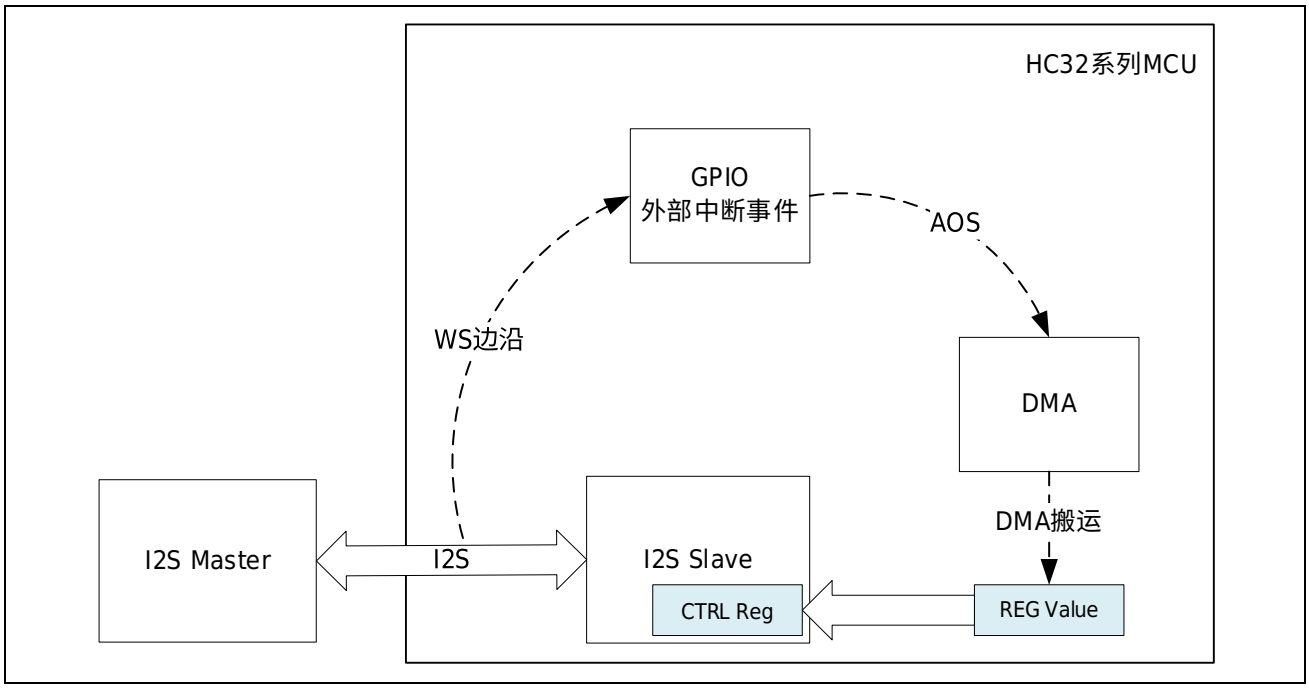


图 4-1 I2S Slave 同步方案框图

5 参考样例及驱动

本章节以 HC32F460 芯片为例，在 HC32F460_DDL_Rev3.3.0 的 i2s_master_slave_int 样例代码上加入 I2S slave 同步功能，提供参考代码。

5.1 将样例定义为从模式

```
/* Master or Slave mode */
#define I2S_UNIT_MASTER_SLAVE                (I2S_MD_SLAVE)
```

5.2 I2S 功能初始化

将样例现有函数进行修改，I2S_Config(void)函数内不使能 I2S 功能，删除以下代码。

```
    /* Enable i2s transfer and receive */
    I2S_FuncCmd(I2S_UNIT, (I2S_FUNC_TX | I2S_FUNC_RX), ENABLE);
```

5.3 宏及变量定义

按实际使用的硬件资源配置宏

```
/* For i2s sync */
#define I2S_EXTINT_CH                        (EXTINT_CH00)
#define I2S_EXINT_SRC                        (INT_SRC_PORT_EIRQ0)

#define DMA_KICK_UNIT                        (CM_DMA1)
#define DMA_KICK_CH                          (DMA_CH0)
#define DMA_KICK_TRIG_CH                     (AOS_DMA1_0)
#define DMA_KICK_SRC                         (EVT_SRC_PORT_EIRQ0)

#define I2sSlaveDmaPeriphEnable()            FCG_Fcg0PeriphClockCmd(FCG0_PERIPH_DMA1 | FCG0_PERIPH_AOS, ENABLE);

#define REG_I2S_CTRL_ADR                     ((uint32_t)&I2S_UNIT->CTRL)

static uint32_t I2S_CTRL_Value = 0;
```

5.4 WS 引脚外部中断初始化

```
static void I2S_ExintInit(void)
{
    stc_extint_init_t stcExtIntInit;
    stc_gpio_init_t stcGpioInit;

    /* GPIO config */
    (void)GPIO_StructInit(&stcGpioInit);
```

```
    stcGpioInit.u16ExtInt = PIN_EXTINT_OFF;
    stcGpioInit.u16PullUp = PIN_PU_ON;
    (void)GPIO_Init(I2S_WS_PORT, I2S_WS_PIN, &stcGpioInit);

    /* Extint config */
    (void)EXTINT_StructInit(&stcExtIntInit);
    stcExtIntInit.u32Filter       = EXTINT_FILTER_OFF;
    stcExtIntInit.u32FilterClock = EXTINT_FCLK_DIV1;
    stcExtIntInit.u32Edge = EXTINT_TRIG_FALLING;
    (void)EXTINT_Init(I2S_EXTINT_CH, &stcExtIntInit);
}
```

5.5 DMA 初始化

```
void KickI2sDmaInit(void)
{
    stc_dma_init_t stcDmaInit;

    I2sSlaveDmaPeriphEnable();
    AOS_SetTriggerEventSrc(DMA_KICK_TRIG_CH, DMA_KICK_SRC);

    (void)DMA_StructInit(&stcDmaInit);

    stcDmaInit.u32IntEn       = DMA_INT_DISABLE;
    stcDmaInit.u32BlockSize   = 1U;
    stcDmaInit.u32TransCount = 1U;
    stcDmaInit.u32DataWidth   = DMA_DATAWIDTH_32BIT;
    stcDmaInit.u32DestAddr    = REG_I2S_CTRL_ADR;
    stcDmaInit.u32SrcAddr     = (uint32_t)(&I2S_CTRL_Value);
    stcDmaInit.u32SrcAddrInc      = DMA_SRC_ADDR_FIX;
    stcDmaInit.u32DestAddrInc     = DMA_DEST_ADDR_FIX;
    (void)DMA_Init(DMA_KICK_UNIT, DMA_KICK_CH, &stcDmaInit);

    I2S_CTRL_Value = I2S_UNIT->CTRL | I2S_FUNC_TX | I2S_FUNC_RX;

    /* DMA module enable */
    DMA_Cmd(DMA_KICK_UNIT, ENABLE);
}
```

5.6 I2S 功能使能

```
void StartI2S(void)
{
    /* DMA software trigger channel enable */
    (void)DMA_ChCmd(DMA_KICK_UNIT, DMA_KICK_CH, ENABLE);
```

```
    GPIO_ExtIntCmd(I2S_WS_PORT, I2S_WS_PIN, ENABLE);
}
```

5.7 I2S 功能关闭

```
void StopI2S(void)
{
    GPIO_ExtIntCmd(I2S_WS_PORT, I2S_WS_PIN, DISABLE);
    I2S_FuncCmd(I2S_UNIT, (I2S_FUNC_TX | I2S_FUNC_RX), DISABLE);
    I2S_ReadData(I2S_UNIT);
}
```

5.8 测试代码 — 检验接收数据

此函数检测从机接收到的数据是否与主机发送的数据循环匹配。

```
en_flag_status_t I2S_SlaveCheckData(const uint16_t *rx_buf, const uint16_t *ref_buf, uint32_t len)
{
    if ((rx_buf == NULL) || (ref_buf == NULL) || (len == 0U)) {
        return RESET;
    }
    for (uint32_t k = 0U; k < len; ++k) {
        uint32_t matched = 1U;
        for (uint32_t i = 0U; i < len; ++i) {
            if (rx_buf[i] != ref_buf[(k + i) % len]) {
                matched = 0U;
                break;
            }
        }
        if (matched) {
            return SET;
        }
    }
    return RESET;
}
```

5.9 测试代码 — 同步测试流程

修改 main(void)测试 I2S 从模式数据同步功能。

```
/* I2S slave sync initial */
KickI2sDmaInit();
I2S_ExintInit();

/* Peripheral registers write protected */
LL_PERIPH_WP(EXAMPLE_PERIPH_WP);
```

```
for (;;) {
    /* Start, wait WS edge */
    StartI2S();

    /* Wait receive finish */
    while (0U == u8RxCompleteFlag);
    /* Stop I2S immediately */
    StopI2S();
    u8RxCompleteFlag = 0U;

    /* Data check */
    if (SET == I2S_SlaveCheckData(u16ReadBuffer, u16WriteBuffer, I2S_DATA_LEN)) {
        printf("Data check success");
    } else {
        printf("r\nData check failed\r\n");
    }
    ClearI2sData();
}
```

6 样例演示

6.1 环境准备

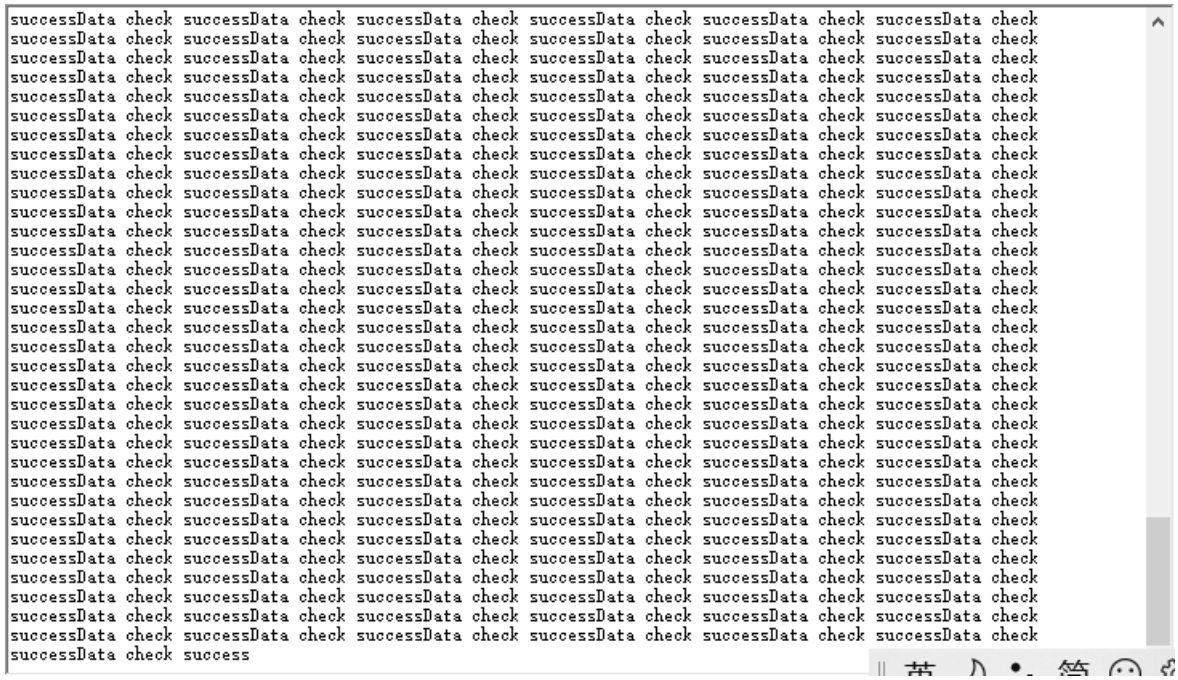
- 确认安装正确的 IAR EWARM v7.7 工具（请从 IAR 官方网站下载相应的安装包，并参考用户手册进行安装）；
- 主机使用样例 i2s_master_slave_int，并且代码中将宏定义修改为
  #define I2S_UNIT_MASTER_SLAVE (I2S_MD_MASTER)
- 硬件连接：

  测试板 1 运行主机样例，测试板 2 通过 OTG 线连接测试板 1

  测试板 1（主机） 测试板 2（从机）
  CK PA7 <----------> PA7
  WS PB0 <----------> PB0
  SD PC4 <----------> PC5
  SDIN PC5 <----------> PC4
  GND <----------> GND

6.2 运行样例

- 运行主机代码，短按按键 K2，启动主机 I2S 数据收发；
- 通过 DAP 下载从机代码，全速运行，通过串口调试助手查看从机数据接收情况；
- 可以看到从机不断地启动和停止 I2S 数据收发，并且校验接收到的数据，显示“Data check success”。



7 总结

本文档主要介绍小华半导体 HC32 系列芯片实现 I2S 从模式同步的方法，即通过 GPIO 外部中断捕获 WS 边沿，经 AOS 触发 DMA 自动写入 I2S_CTRL 寄存器以实现硬件级帧同步，以解决 I2S 从模式应用中可能会遇到的声道不同步和数据位不同步问题，给客户实际使用提供参考。

版本修订记录

版本号	修订日期	修订内容
Rev1.00	2026/08/05	初版发布。