

08x-05x 芯片 SPI 相互通讯说明

1、概述

08x 或 05x 系列两颗芯片相互 SPI 通讯会出现主机或从机接收数据发送移位现象，该移位现象会根据 SPI 时钟信号在默认情况下的电平状态和数据的发送/接收时刻配置不同，导致主机或从机接收数据出现移位（表 1 SPI 主/从接收数据移位统计表）；移位方向会根据传输顺序不同而不同，LSB 低位先传输则接收数据右移位，反之 MSB 高位先传输接收数据左移。

本实验以配置主机相位移位 Phase 为 1，极性选择 Polarity 为 0；从机相位移位 Phase 为 0，极性选择 Polarity 为 0，对应表 1 第 9 组现象，数据低位先发送（LSB）为例，进行软件处理解决主或从机接收数据错位和发送第一个 bit 丢失问题。Phase 和 Polarity 具体配置对应数据处理关系如图 1 和图 2：

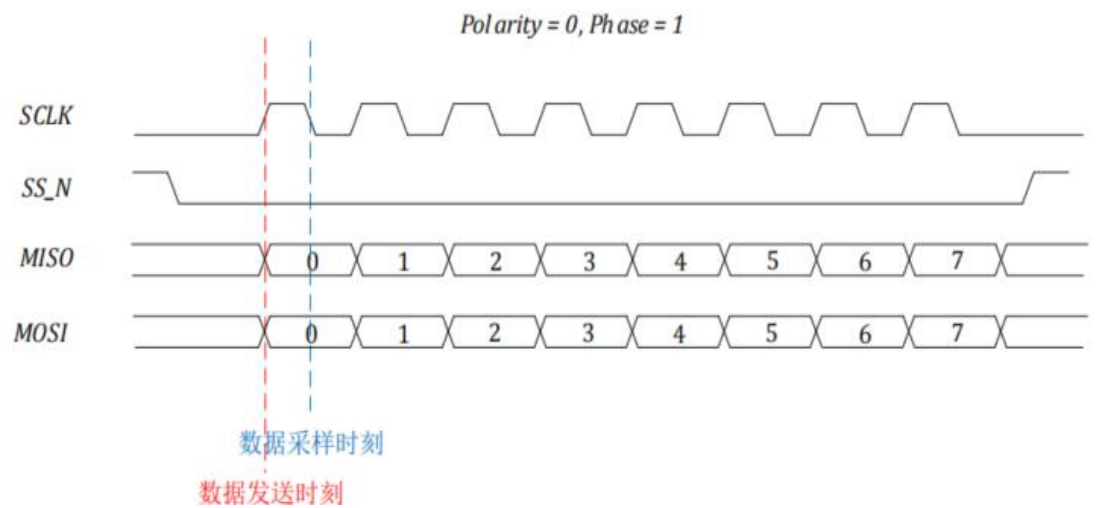


图 1

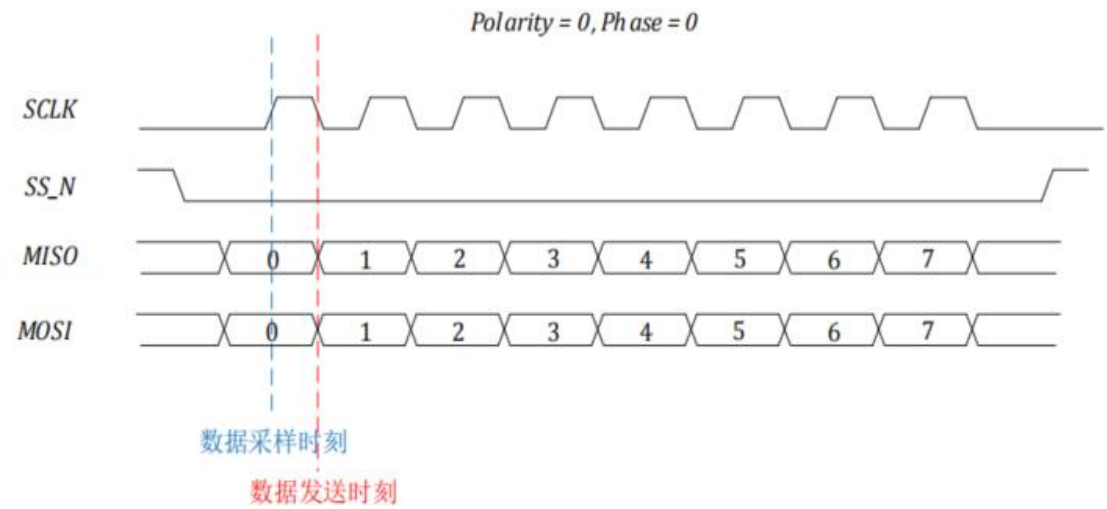


图 2

按照上述配置数据错位和发送的第一个 bit 丢失现象说明: 正常主机接收数据会出现整体右移 1 位, 然后第一字节的 bit0 会被丢失。如图 3 为从机发送的正常两字节数据 0xAA 和 0x55, 图 4 为主机接收的异常数据, 整体数据向右移 1 位, 第一字节的 bit0 丢失, 最终接收两字节数据 0xD5 和 0x2A, 如图所示:

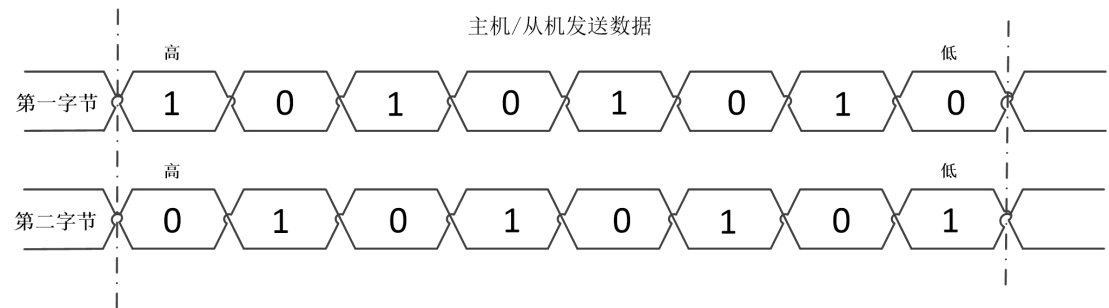


图 3

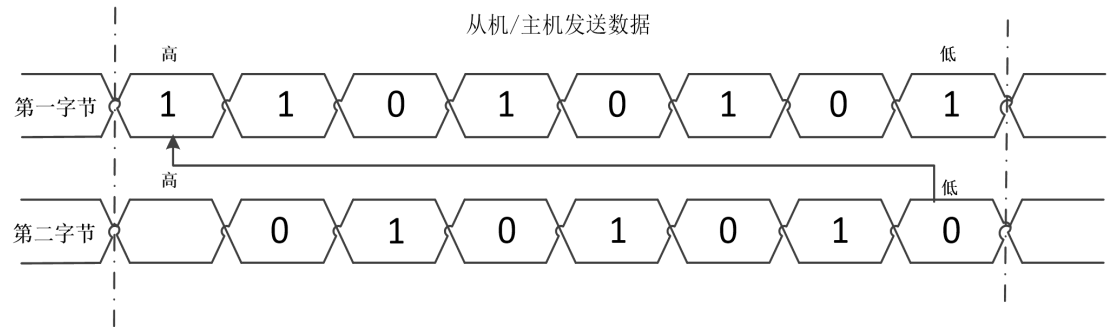


图 4

该现象是由于芯片内部 SPI 设计原因, 导致 LKS 两颗芯片相互通讯会发生第一个 bit 丢失。数据实际接收从发送的第二个 bit 进行采集数据, 这种现象只会出现在 LKS 两颗芯片 SPI 通讯问题上, 与其它设备和 MCU 通讯均正常。

2、SPI 数据处理方法描述

为了解决 LKS 两颗芯片 SPI 正常相互通讯，我们只能做软件补偿。

补偿原理：我们以发送 n 个字节举例，因为主模式接收的数据是实际数据右移后的数据，所以我们将错就错，在从模式发送数据之前，将发送的数据统一左移一位，使用移位后的数据进行发送，这样主机接收的数据又自动右移，这样得到的结果与我们从模式期望主机接收的数据就一致啦。

那么我们从机将原始数据统一右移以后得到了 n+1 个字节，此时第 1 字节的最低位为 0，第 n+1 字节的 bit1-bit7 为 0，bit0 为实际数据的第 n 字节最高位，之后我们将实际数据移位之后的数组进行 SPI 发送，SPI 收发 n+1 个字节即可。

注意此时也需要主从模式收发字节个数配置 n+1 个字节（因为是全双工模式，主从收发数据字节数必须保持一致），这样主模式接收 n 字节的数据，就是我们期望的数据（正确数据），第 n+1 字节全为零对我们没有什么影响可以软件舍弃，从机也接收 n+1 个字节，第 n+1 字节舍弃。

下表为 08x/05x 配置 SPI 以 LSB 顺序发送数据的基础上，配置 Polarity(SPI 极性选择)，Phase（SPI 相位选择）与主从数据收发数据之间的关系表格。

表 1 SPI 主/从接收数据移位统计表

标号	主机	从机	现象
数据传输方向 LSB，低位先传输			
1	Polarity=0 Phase=0	Polarity=0, Phase=0	主模式接收数据右移 1 位
2		Polarity=1, Phase=0	从模式接收数据右移 1 位
3		Polarity=0, Phase=1	从模式接收数据右移 1 位
4		Polarity=1, Phase=1	主从接收发送数据错乱
5	Polarity=1 Phase=0	Polarity=0, Phase=0	从模式接收数据右移 1 位
6		Polarity=1, Phase=0	主模式接收数据右移 1 位
7		Polarity=0, Phase=1	主从接收发送数据错乱
8		Polarity=1, Phase=1	从模式接收数据右移 1 位
9	Polarity=0 Phase=1	Polarity=0, Phase=0	主模式接收数据右移 1 位
10		Polarity=1, Phase=0	主模式接收数据右移 1 位
11		Polarity=0, Phase=1	主模式接收数据右移 1 位
12		Polarity=1, Phase=1	主从接收发送数据错乱
13	Polarity=1 Phase=1	Polarity=0, Phase=0	主模式接收数据右移 1 位
14		Polarity=1, Phase=0	主模式接收数据右移 1 位
15		Polarity=0, Phase=1	主从接收发送数据错乱
16		Polarity=1, Phase=1	主模式接收数据右移 1 位

3、程序说明

3.1 SPI 数据补偿临时变量申请

```
typedef struct
{
    u8 SPI_TXRX_SIZE;      /* SPI 数据收发字节个数*/
    u8 SPI_Temp_Data[512]; /* 发送数据移位后，数据存放缓存区*/
    u8 *SPI_TempTX_Data;   /* 发送数组指针*/
    u8 *SPI_TempRX_Data;   /* 接收数组指针*/
} SPI_BuffData;
extern SPI_BuffData SPI_DATA;
```

3.2 SPI 从机发送数据处理

3.2.1 SPI 数据补偿从机发送函数处理

第一步:将 SPI 发送数据移位后的临时存储数组进行清零，数组长度为 spi_len+1（左移位后数据个数与原始数据相比增加 1 位）。

```
void Spi_master_DMA(UINT8 *spi_txdma_data, UINT8 *spi_rxdma_data, u8 spi_len)
{
    u8 i;
    for (i = 0; i < spi_len + 1; i++)
    {
        SPI_DATA.SPI_Temp_Data[i] = 0; //发送数组缓冲区清零
    }
}
```

第二步:将原始数据左移后存储到 SPI_DATA.SPI_Temp_Data 临时缓冲区，然后第 spi_len 字节的最高位存储到临时缓冲区的 spi_len+1 字节区，此时从机发送的数据处理完毕。

```
/******数据整体右移******/
for (i = 0; i < spi_len; i++)
{
    SPI_DATA.SPI_Temp_Data[i] = spi_txdma_data[i] << 1;
    if (i != 0)
    {
        SPI_DATA.SPI_Temp_Data[i] |= (spi_txdma_data[i - 1] & 0x80) >> 7;
    }
}
SPI_DATA.SPI_Temp_Data[i] = ((spi_txdma_data[i - 1] & 0x80) >> 7);
```

第三步：将临时发送接收数组指针指向发送和接收数组，把发送与接收字节数信息赋值给临时发送与接收数据结构体变量。该操作是为了在 SPI 从模式数据传输异常和数据传输溢出时间发生时进行再次数据发送使用。

```
/*临时变量进行发送/接收信息获取，SPI 发送中断对发送失败事件进行再次发送时使用*/
SPI_DATA.SPI_TXRX_SIZE = spi_len;
SPI_DATA.SPI_TempTX_Data = spi_txdma_data;
SPI_DATA.SPI_TempRX_Data = spi_rxdma_data;
```

第四步：因为最终移位后数组比原始数组对一个字节，所以发送字节数需要加 1。

```
spi_len = spi_len + 1;          //实际发送数据为 spi_len + 1
```

第五步：将原始数据偏移后存储的临时数组 SPI_DATA.SPI_Temp_Data 地址给 DMA，然后进行正常的 SPI 数据发送即可。

```
/******SPI 数据发送******/
SPI_SIZE = spi_len;          //SPI 发送 spi_len 字节
//发送 1 轮，每轮 1 个字节
DMA_CTMS2 = (((spi_len) << 16) & 0x00ff0000) | 0x00000001;
DMA_CPAR2 = (u32)&SPI_TX_DATA;          //SPI_TX_DATA
DMA_CMAR2 = (u32)SPI_DATA.SPI_Temp_Data; //内存地址
DMA_CCR2 = 0x2091; //BIT0:通道 2 使能,BIT4: 传输方向外设至内存,BIT7: 内存第二轮地址在第一轮上递增,BIT8~BIT9 = 0b00:外设访问位宽 byte,BIT10~BIT11 = 0b00:内存访问位宽 byte, BIT12~BIT14 = 0b010:选择 SPI_TX 作为 DMA_CH2 触发信号
//接收 1 轮，每轮一个字节
DMA_CTMS1 = (((spi_len) << 16) & 0x00ff0000) | 0x00000001;
DMA_CPAR1 = (u32)&SPI_RX_DATA;          //SPI_RX_DATA
DMA_CMAR1 = (u32)spi_rxdma_data;        //内存地址
DMA_CCR1 = 0x1081; //BIT0:通道 2 使能,BIT1:传输完成中断使能,BIT7: 内存第二轮地址在第一轮上递增, BIT8~BIT9 = 0b00:外设访问位宽 byte, BIT10~BIT11 = 0b00:内存访问位宽 byte, BIT12~BIT14 = 0b001:选择 SPI_RX 作为 DMA_CH1 触发信号
DMA_CTRL = 0x0001; //DMA 使能
//触发 SPI 传输完成标志位置位，使 SPI_DMA 搬移，
SPI_TX_DATA = 0x00000000;
}
```

3.2.2 SPI 主函数与收发数组定义

首先我们申请发送数组，数组发送为 8 字节，那么我们接收数据申请大小必

须大于 8 个字节,因为从模式发送数据由于左移位后数组字节个数增加移位,SPI 实际发送数据个数为 9 个字节,而我们 SPI 配置为全双工模式,所以数据接收个数也是 9。如果我们申请接收数组大小也是 8,那么接收的第 9 个字节就会导致数组溢出。(主模式接收与发送缓冲区大小也需要申请 9 个字节)

```
volatile u8 Tx_data_tbl[8] = {0xAA, 0x55, 0x34, 0x56, 0xAA, 0x9a, 0x55, 0xd1}; //数据发送缓冲区
volatile u8 Rx_data_tbl[sizeof(Tx_data_tbl) + 1] = {0};
int main(void)
{
    Hardware_init();
    Spi_master_DMA((u8*)Tx_data_tbl, (u8*)Rx_data_tbl, sizeof(Tx_data_tbl));

    while(1)
    {
        Delay(0xffff);
    }
}
```

3. 2. 3 SPI 中断初始函数

演示例程,从模式初始化后发送一次,发送完成后进入发送完成中断,在发送完成中断继续进行数据发送,如果不需要继续发送可以去掉发送完成中断的发送函数。

如果 SPI 发生出现数据传输异常或传输溢出事件时,在相应中断内对 SPI 进行初始化后重新进行数据发送。

```
extern void SPI_init(void);
void SPI0_IRQHandler(void)
{
    if (SPI_IE & BIT2) //SPI 传输完成事件中断
    {
        SPI_IE |= BIT2;
        Spi_master_DMA(SPI_DATA.SPI_TempTX_Data,
                        SPI_DATA.SPI_TempRX_Data, SPI_DATA.SPI_TXRX_SIZE);
    }
    if (SPI_IE & BIT1) //error SPI 传输异常事件中断
    {
        SPI_IE |= BIT1;
        SPI_init();
        Spi_master_DMA(SPI_DATA.SPI_TempTX_Data,
                        SPI_DATA.SPI_TempRX_Data, SPI_DATA.SPI_TXRX_SIZE);
    }
}
```

```
if (SPI_IE & BIT0) //ovfl SPI 传输溢出事件中断
{
    SPI_IE |= BIT0;
    SPI_init();
    Spi_master_DMA(SPI_DATA.SPI_TempTX_Data,
        SPI_DATA.SPI_TempRX_Data, SPI_DATA.SPI_TXRX_SIZE);
}
}
```

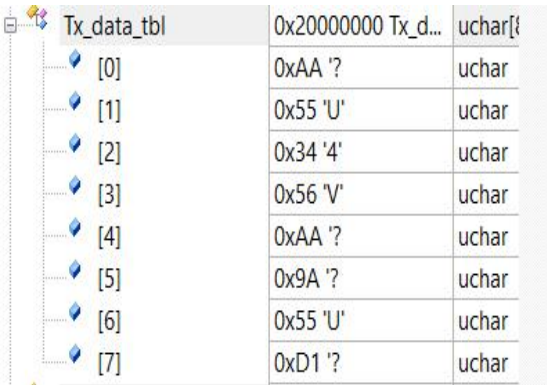
3.2.4 SPI 主模式申请收发数组大小注意事项

因为该例程为 LKS 双芯片 SPI 相互通讯，所以传输模式为全双工模式。上述表述了 SPI 从机发送数据为 9 字节，那么主机申请收发数组字节个数也必须大于或等于从机发送字节个数，防止数据溢出问题。

```
#define Tx_Rx_Len 8 /*全双工模式发送 8 个字节*/
u8 tx_table[Tx_Rx_Len+1] = {0x05, 0x12, 0x34, 0x56, 0x78, 0x9a, 0xbc,
    0xd1};
//接收数据，接收数组申请大小与发送数组大小保持一致
u8 rx_table[sizeof(tx_table)];
```

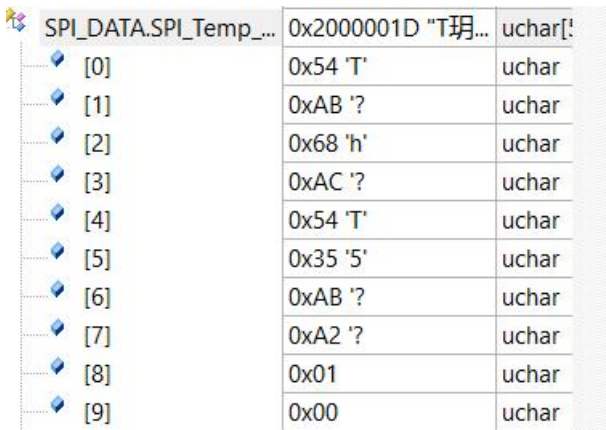
4、下载验证

图 5 为从机需要发送的 8 个字节，图 6 为数据发送前左移之后的 9 个字节。
图 7 为从模式接收到的 9 个字节，前 8 个字节为主机真实发送的字节，第 9 字节舍去即可。



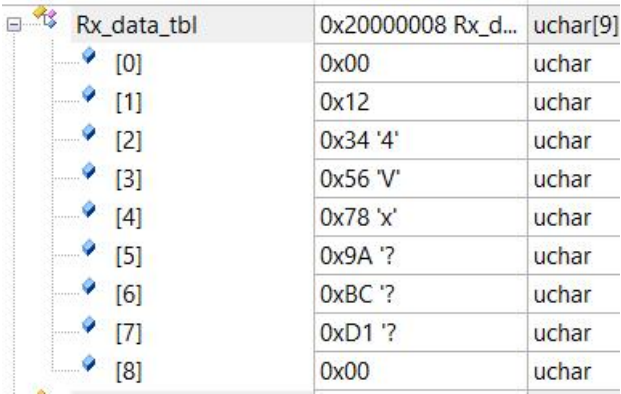
Tx_data_tbl	0x20000000 Tx_d...	uchar{
[0]	0xAA '?'	uchar
[1]	0x55 'U'	uchar
[2]	0x34 '4'	uchar
[3]	0x56 'V'	uchar
[4]	0xAA '?'	uchar
[5]	0x9A '?'	uchar
[6]	0x55 'U'	uchar
[7]	0xD1 '?'	uchar

图 5



SPI_DATA.SPI_Temp_...	0x2000001D "T玊...	uchar{!
[0]	0x54 'T'	uchar
[1]	0xAB '?'	uchar
[2]	0x68 'h'	uchar
[3]	0xAC '?'	uchar
[4]	0x54 'T'	uchar
[5]	0x35 '5'	uchar
[6]	0xAB '?'	uchar
[7]	0xA2 '?'	uchar
[8]	0x01	uchar
[9]	0x00	uchar

图 6



Rx_data_tbl	0x20000008 Rx_d...	uchar[9]
[0]	0x00	uchar
[1]	0x12	uchar
[2]	0x34 '4'	uchar
[3]	0x56 'V'	uchar
[4]	0x78 'x'	uchar
[5]	0x9A '?'	uchar
[6]	0xBC '?'	uchar
[7]	0xD1 '?'	uchar
[8]	0x00	uchar

图 7

图 8 为主模式发送数据与接收到从机发送的期望数据（正确数据），从机接收数组的第 9 字节用户在使用时可以舍弃。

rx_table	0x20000008 rx_ta...	uchar[9]
[0]	0xAA '?'	uchar
[1]	0x55 'U'	uchar
[2]	0x34 '4'	uchar
[3]	0x56 'V'	uchar
[4]	0xAA '?'	uchar
[5]	0x9A '?'	uchar
[6]	0x55 'U'	uchar
[7]	0xD1 '?'	uchar
[8]	0x00	uchar
tx_table	0x20000000 tx_ta...	uchar[8]
[0]	0x00	uchar
[1]	0x12	uchar
[2]	0x34 '4'	uchar
[3]	0x56 'V'	uchar
[4]	0x78 'x'	uchar
[5]	0x9A '?'	uchar
[6]	0xBC '?'	uchar
[7]	0xD1 '?'	uchar

图 8

图 9 为示波器抓出的主从模式发送的真实数据，从图中可以清楚看到从机发送的数据是左移 1 位后的 9 字节数据，与上述描述相符。

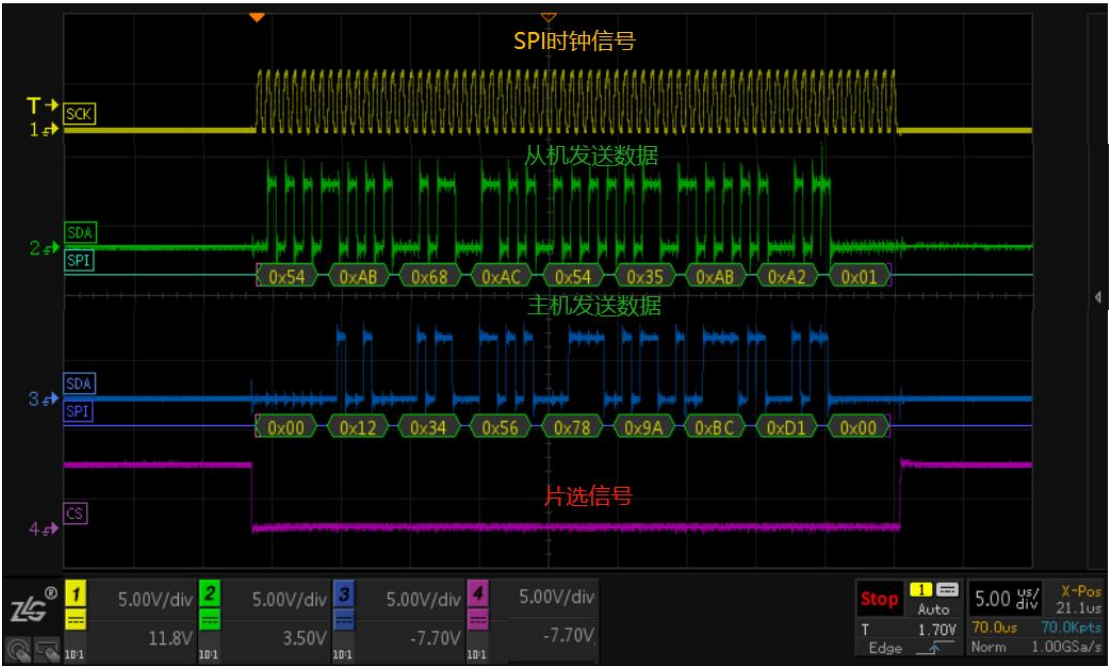


图 9