Principles and Practices of Microcontroller (Embedded System Design I) -Comunication

Gang Chen (陈刚)

Associate Professor

Institute of Unmanned Systems
School of data and computer science
Sun Yat-Sen University

https://www.usilab.cn/team/chengang/

中山大學 数据科学与计算机学院
SUN YAT-SEN UNIVERSITY School of Data and Computer Science

UART (Universal Asynchronous Receiver/Transmitter)

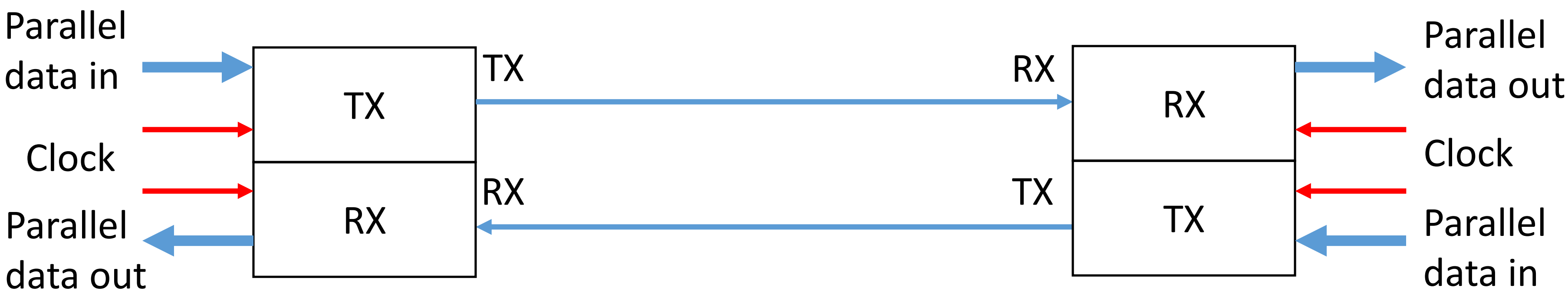
- The key component of the serial communications subsystem of a computer.

- Involves the data transfer for each direction.

- No clock is transmitted with serial data.

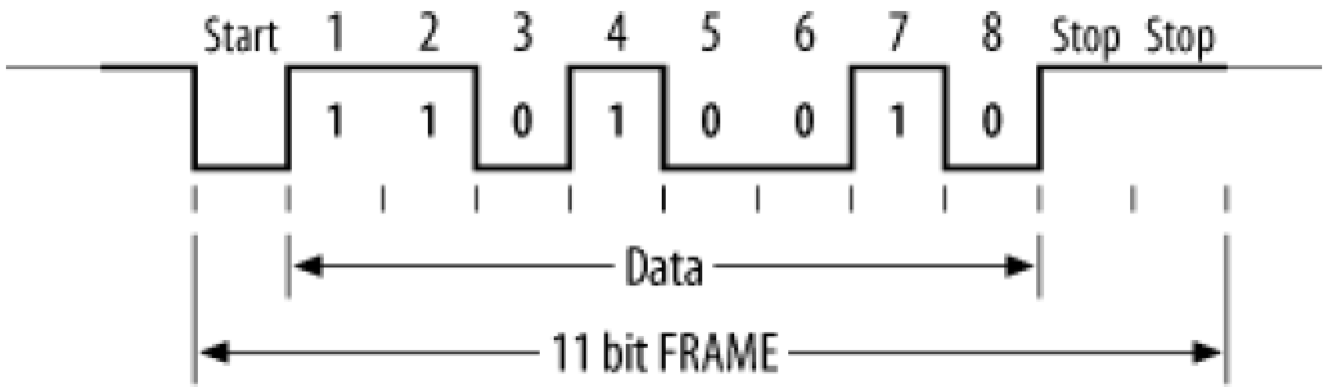
- Takes bytes of data and transmits the individual bits in a sequential fashion.

Physical UART Communication



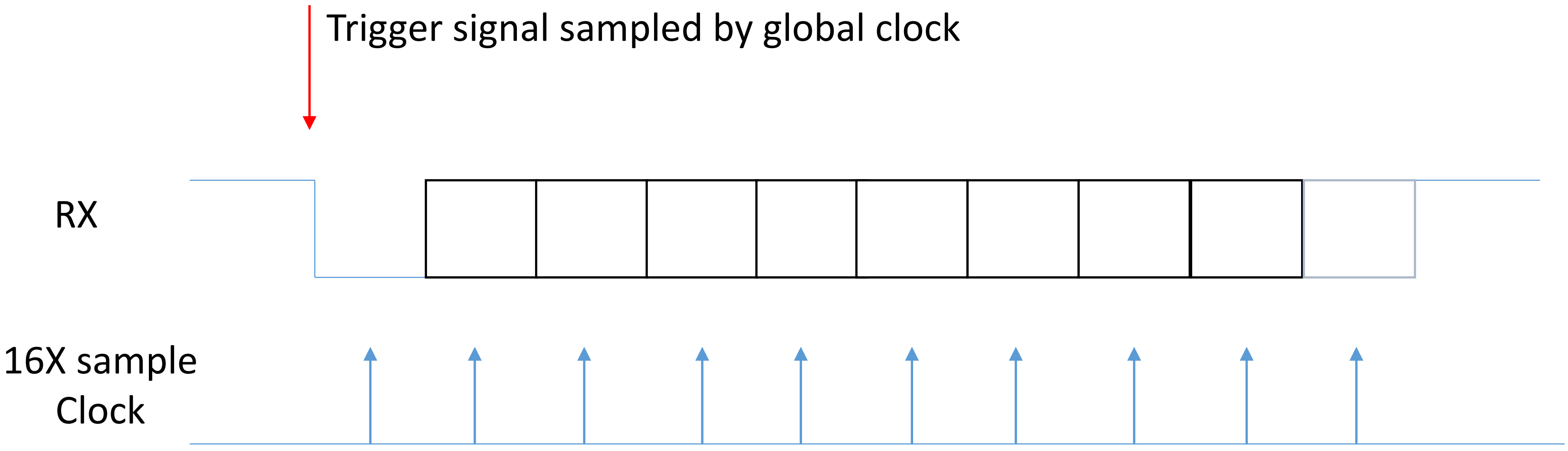
- Do not directly generate or receive the external signals used between different items of equipment.
- Separate interface devices are used to convert the logic level signals of the UART to and from the external signaling levels.
- External signals may be of many different forms.

UART Protocol

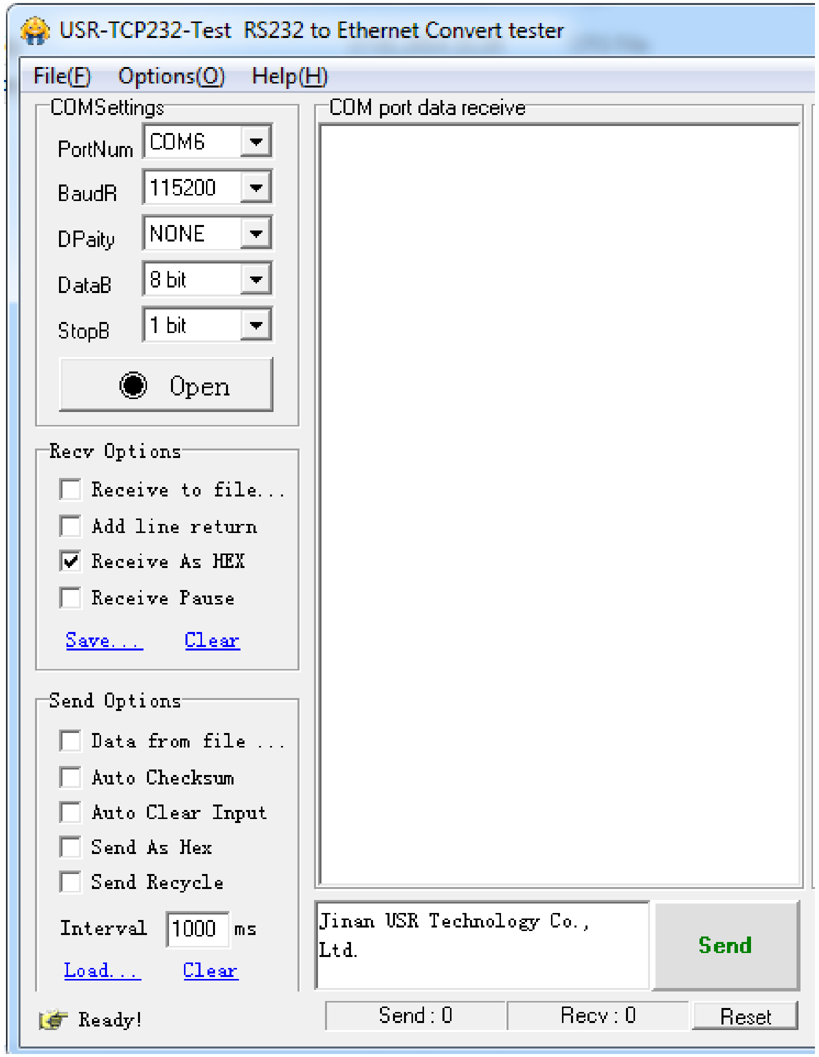


- The idle, no data state is high-voltage, or powered.
- Each character is sent as a logic low start bit.
- A configurable number of data bits.
- An optional parity bit for error detection
- One or more logic high stop bits

如何同步异步信号



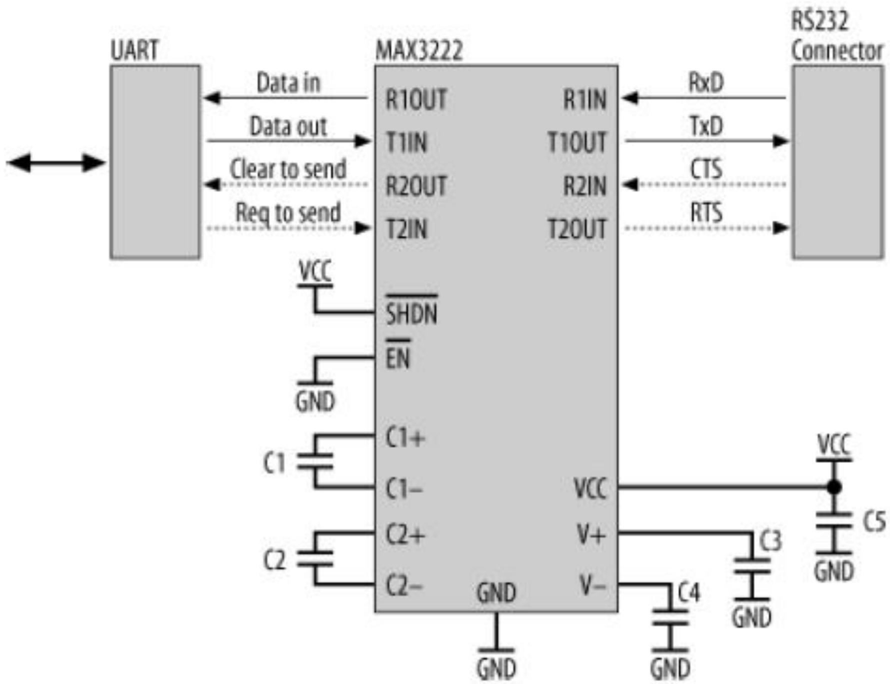
参数以及串口调试接口



USR-TCP232-Test RS232 to Ethernet Convert tester
File(F) Options(O) Help(H)
COMSettings
PortNum COM6
BaudR 115200
DPaity NONE
DataB 8 bit
StopB 1 bit
Open
Recv Options
Receive to file...
Add line return
Receive As HEX
Receive Pause
Save... Clear
Send Options
Data from file ...
Auto Checksum
Auto Clear Input
Send As Hex
Send Recycle
Interval 1000 ms
Load... Clear
COM port data receive
Jinan USR Technology Co., Ltd.
Send
Ready!
Send : 0
Recv : 0
Reset

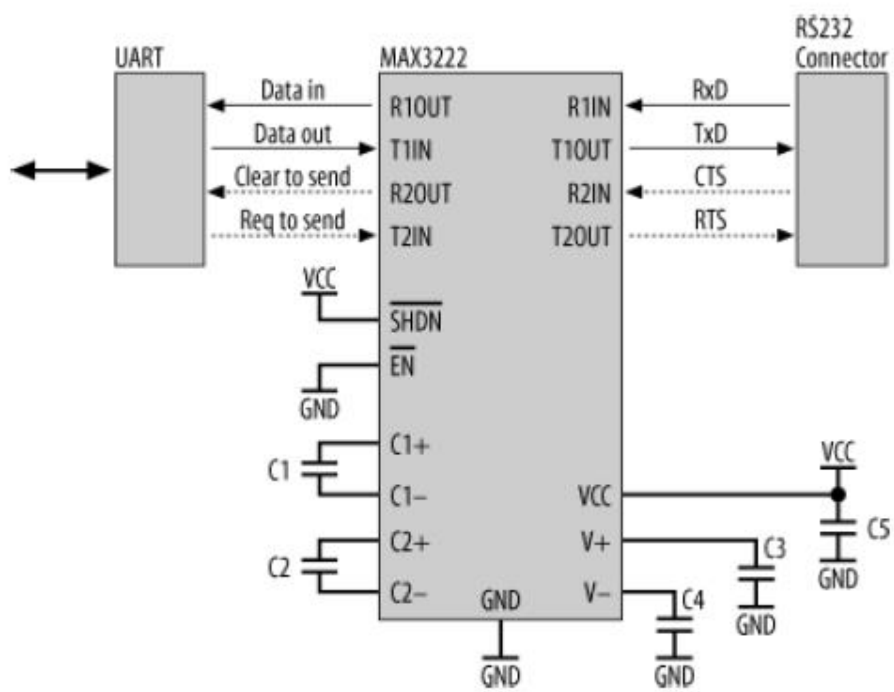
RS232

- **External Signal**
  - Logic high: -5V to -15 V (typically -12V)
  - Logic low: +5V to 15V (typically 12V)
  - Converter IC: Max232, Max3232



RS232

- **External Signal**
  - Logic high: -5V to -15 V (typically -12V)
  - Logic low: +5V to 15V (typically 12V)
  - Converter IC: Max232, Max3232

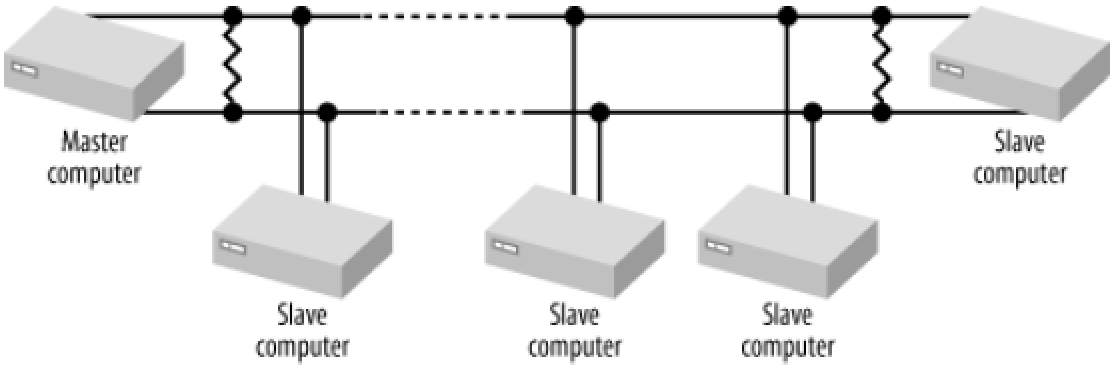


RS485

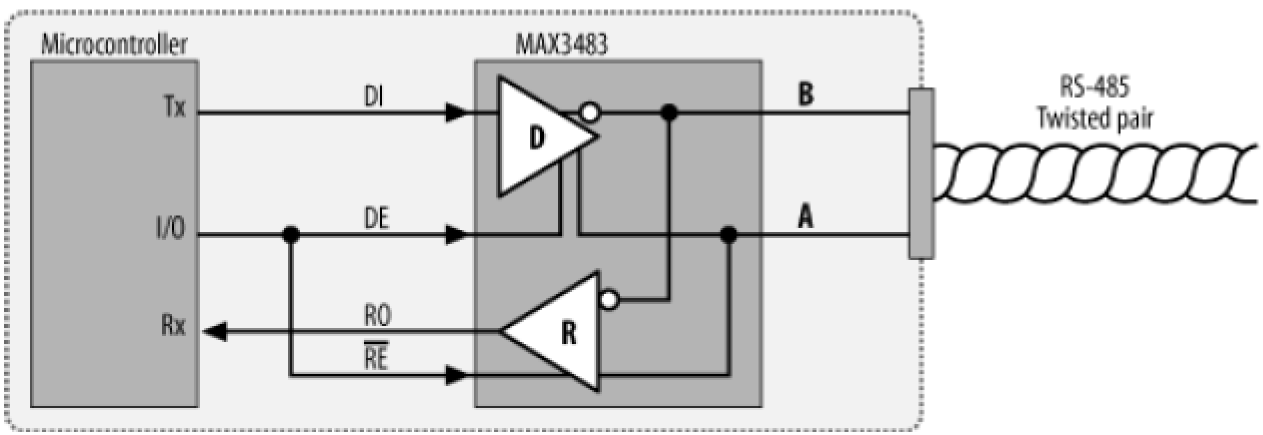
- **External Signal: use difference between two lines (Common-mode rejection)**
  - Logic high: -2V to -6 V
  - Logic low: +2V to 6V
- **Commonly used for low-cost networking and in many industrial applications**
- **Master-slave architecture (half duplex)**
- **Support data transmission over cable length up to 1200 meters**

RS485

Network:



Max3483:



STM32库函数

1 STM32串口常用寄存器和库函数

2 串口配置一般步骤(手把手写串口实例)

寄存器

- 常用的串口相关寄存器
  - USART_SR状态寄存器
  - USART_DR数据寄存器
  - USART_BRR波特率寄存器

STM32串口常用寄存器和库函数

- 串口操作相关库函数（省略入口参数）：

*void USART_Init(); //*串口初始化：波特率，数据字长，奇偶校验，硬件流控以及收发使能
*void USART_Cmd();//*使能串口
*void USART_ITConfig();//*使能相关中断

*void USART_SendData();//*发送数据到串口，**DR**
*uint16_t USART_ReceiveData();//*接受数据，从**DR**读取接受到的数据

*FlagStatus USART_GetFlagStatus();//*获取状态标志位
*void USART_ClearFlag();//*清除状态标志位
*ITStatus USART_GetITStatus();//*获取中断状态标志位
*void USART_ClearITPendingBit();//*清除中断状态标志位

STM32串口常用寄存器和库函数

状态寄存器(USART_SR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
						rc w0	rc w0	r	rc w0	rc w0	r	r	r	r	r

位31:10	保留位，硬件强制为0
位9	**CTS**: CTS 标志 (CTS flag) 如果设置了CTSE位，当nCTS输入变化状态时，该位被硬件置高。由软件将其清零。如果USART_CR3中的CTSIE为'1'，则产生中断。 0：nCTS状态线上没有变化； 1：nCTS状态线上发生变化。 注：UART4和UART5上不存在这一位。
位8	**LBD**: LIN断开检测标志 (LIN break detection flag) 当探测到LIN断开时，该位由硬件置'1'，由软件清'0'(向该位写0)。如果USART_CR3中的LBDIE = 1，则产生中断。 0：没有检测到LIN断开；

FlagStatus USART_GetFlagStatus(USART_TypeDef USARTx, uint16_t USART_FLAG);*

STM32串口常用寄存器和库函数

数据寄存器(USART_DR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

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

位31:9	保留位，硬件强制为0
位8:0	**DR[8:0]**：数据值 (Data value) 包含了发送或接收的数据。由于它是由两个寄存器组成的，一个给发送用(TDR)，一个给接收用(RDR)，该寄存器兼具读和写的功能。TDR寄存器提供了内部总线和输出移位寄存器之间的并行接口(参见图248)。RDR寄存器提供了输入移位寄存器和内部总线之间的并行接口。 当使能校验位(USART_CR1中PCE位被置位)进行发送时，写到MSB的值(根据数据的长度不同，MSB是第7位或者第8位)会被后来的校验位该取代。 当使能校验位进行接收时，读到的MSB位是接收到的校验位。

void USART_SendData(USART_TypeDef USARTx, uint16_t Data);*
uint16_t USART_ReceiveData(USART_TypeDef USARTx);*

STM32串口常用寄存器和库函数

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:16 保留，必须保持复位值

位 15:4 **DIV_Mantissa[11:0]**：USARTDIV 的尾数
这 12 个位用于定义 USART 除数 (USARTDIV) 的尾数

位 3:0 **DIV_Fraction[3:0]**：USARTDIV 的小数
这 4 个位用于定义 USART 除数 (USARTDIV) 的小数。当 OVER8 = 1 时，不考虑 DIV_Fraction3 位，且必须将该位保持清零。

void USART_Init(USART_TypeDef USARTx, USART_InitTypeDef* USART_InitStruct);*

串口配置的一般步骤

①串口时钟使能：*RCC_APBxPeriphClockCmd();*
- GPIO时钟使能：*RCC_AHB1PeriphClockCmd();*
- ② 引脚复用映射：
- *GPIO_PinAFConfig();*
- ③GPIO端口模式设置:*GPIO_Init(); 模式设置为GPIO_Mode_AF*
- ④串口参数初始化：*USART_Init();*
- ⑤开启中断并且初始化NVIC（如果需要开启中断才需要这个步骤）
- *NVIC_Init();*
- *USART_ITConfig();*
- ⑥使能串口:*USART_Cmd();*
- ⑦编写中断处理函数：*USARTx_IRQHandler();*
- ⑧串口数据收发：
- *void USART_SendData();//发送数据到串口，* DR
- *uint16_t USART_ReceiveData();//接受数据，* 从DR读取接受到的数据
- ⑨串口传输状态获取：
- *FlagStatus USART_GetFlagStatus();*
- *void USART_ClearITPendingBit();*

实验

- 找物理管脚

Bits 5:4 **USART3_REMAP[1:0]:** USART3 remapping

These bits are set and cleared by software. They control the mapping of USART3 CTS, RTS,CK,TX and RX alternate functions on the GPIO ports.

00: No remap (TX/PB10, RX/PB11, CK/PB12, CTS/PB13, RTS/PB14)
01: Partial remap (TX/PC10, RX/PC11, CK/PC12, CTS/PB13, RTS/PB14)
10: not used
11: Full remap (TX/PD8, RX/PD9, CK/PD10, CTS/PD11, RTS/PD12)

Bit 3 **USART2_REMAP:** USART2 remapping

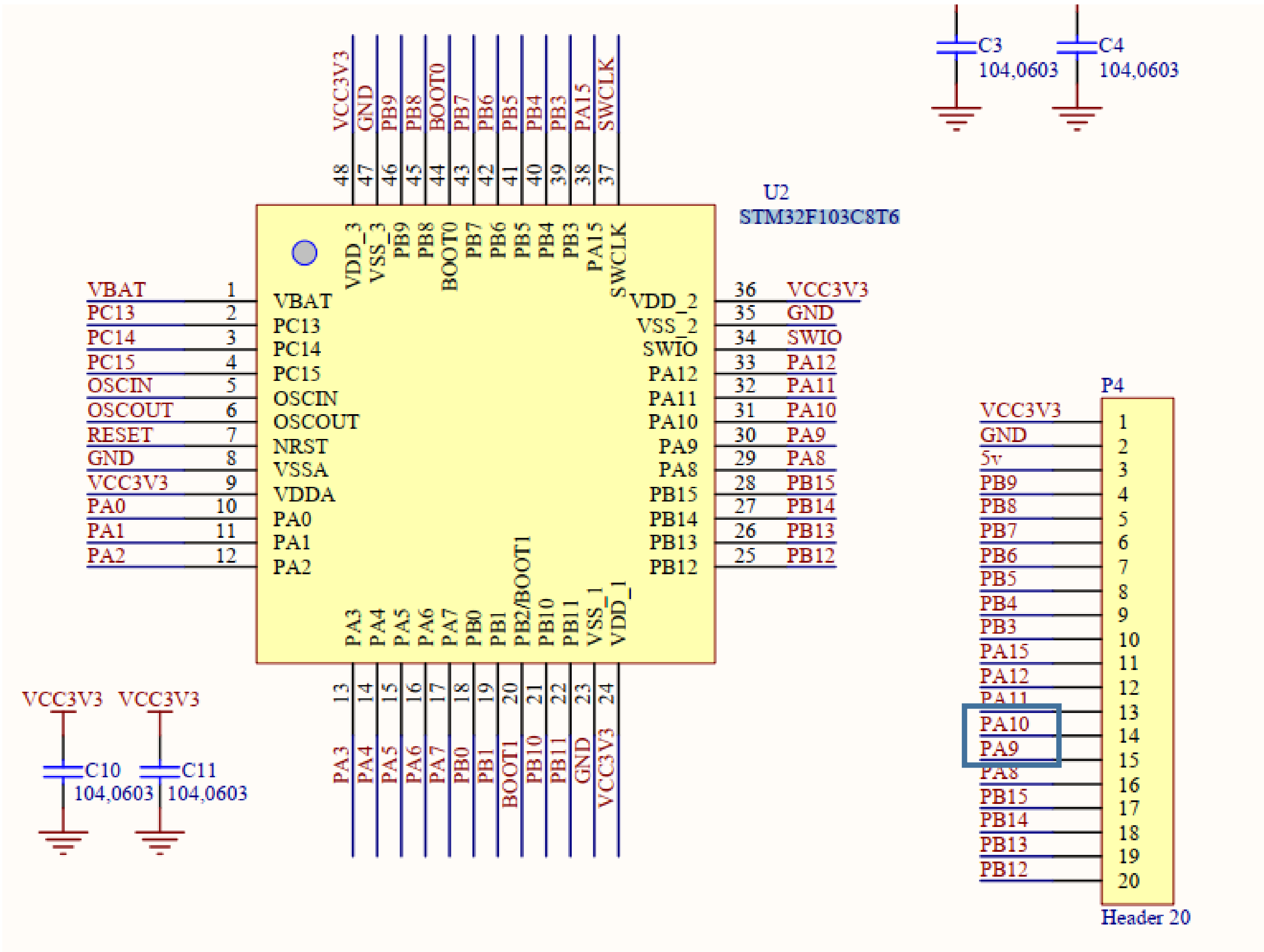
This bit is set and cleared by software. It controls the mapping of USART2 CTS, RTS,CK,TX and RX alternate functions on the GPIO ports.

0: No remap (CTS/PA0, RTS/PA1, TX/PA2, RX/PA3, CK/PA4)
1: Remap (CTS/PD3, RTS/PD4, TX/PD5, RX/PD6, CK/PD7)

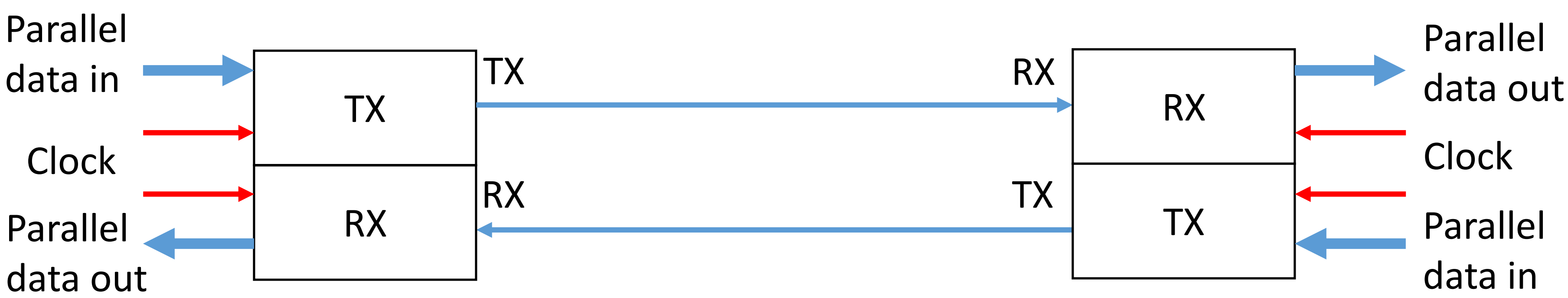
Bit 2 **USART1_REMAP:** USART1 remapping

This bit is set and cleared by software. It controls the mapping of USART1 TX and RX alternate functions on the GPIO ports.

0: No remap (TX/PA9, RX/PA10)
1: Remap (TX/PB6, RX/PB7)


U2
STM32F103C8T6
VBAT
PC13
PC14
PC15
OSCIN
OSCOUT
RESET
GND
VCC3V3
PA0
PA1
PA2
NRST
VSSA
VDDA
VDD_3
VSS_3
PB9
PB8
BOOT0
PB7
PB6
PB5
PB4
PB3
PA15
SWCLK
VDD_2
VSS_2
SWIO
PA12
PA11
PA10
PA9
PA8
PB15
PB14
PB13
PB12
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2/BOOT1
BOOT1
PB10
PB11
VSS_1
VDD_1
C3
104,0603
C4
104,0603
C10
104,0603
C11
104,0603
P4
5v
Header 20

- 交叉连接


Parallel
data in
Clock
Parallel
data out
TX
RX
TX
RX
RX
TX
RX
TX
Parallel
data out
Clock
Parallel
data in

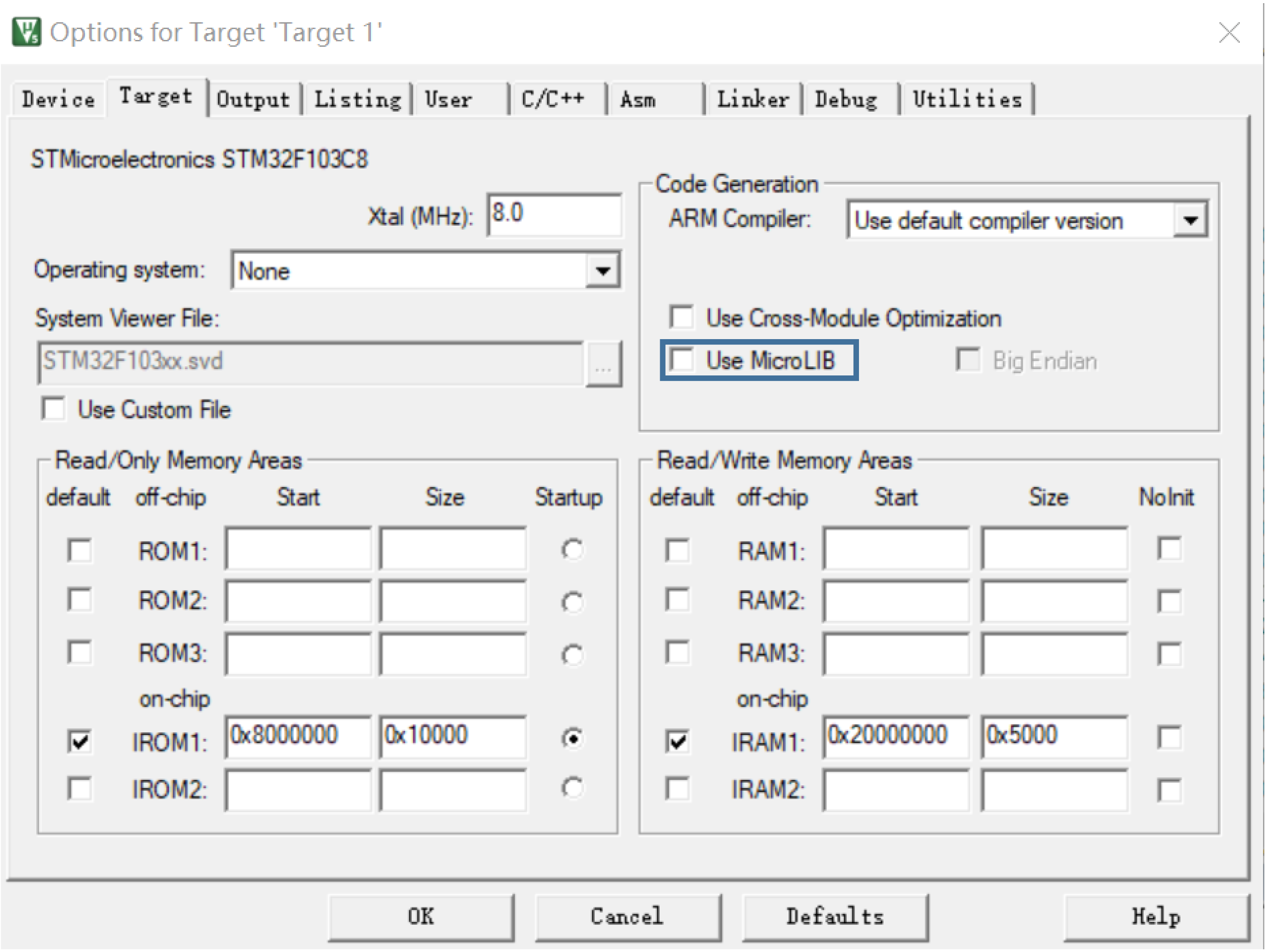
```c
void USART1_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;  //定义串口初始化结构体
    RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOA | RCC_APB2Periph_USART1,ENABLE);
  //本函数（使能时钟）参数中，RCC_APB2Periph_USART1是必不可少的，有人会问，对于串口用到的PA9和
  //PA10不用使能时钟吗？其实由于USART1默认的就是PA9和PA10，所以这一个就行了，当然你要是加上
  //这个|RCC_APB2Periph_GPIOA也是不报错的，只是重复了。
  /*  USART1_TX -> PA9 */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;         //选中串口默认输出管脚
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;   //定义输出最大速率
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//定义管脚9的模式
  GPIO_Init(GPIOA, &GPIO_InitStructure);            //调用函数，把结构体参数输入进行初始化
  /*  USART1_RX ->  PA10*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  USART_InitStructure.USART_BaudRate = 9600; //波特率
  USART_InitStructure.USART_WordLength = USART_WordLength_8b; //数据位8位
  USART_InitStructure.USART_StopBits = USART_StopBits_1;    //停止位1位
  USART_InitStructure.USART_Parity = USART_Parity_No;       //校验位 无
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;//无流控制
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;       //使能接收和发送引脚

  USART_Init(USART1, &USART_InitStructure); //将以上赋完值的结构体带入库函数USART_Init进行初始化
  USART_ClearFlag(USART1,USART_FLAG_TC);
  USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
  USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
  USART_Cmd(USART1, ENABLE);//开启USART1，注意与上面RCC_APB2PeriphClockCmd()设置的区别
}
```

printf重定向

- printf函数实际是一个宏，最终调用的是 fputc(int ch,FILE *f)这个函数

```
int fputc(int ch, FILE *f)
{
//将Printf内容发往串口
  USART_SendData(USART1, (unsigned char) ch);
  while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
  return (ch);
}
```

注意：由于fputc()函数的形参调用了C库的FILE,所以在程序中加入stdio.h这个头文件，便且在keil的编译器的设置中勾选Use MicroLIB(使用微库)

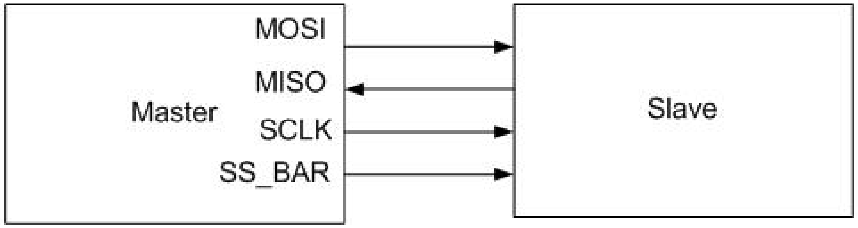

Options for Target 'Target 1'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
STMicroelectronics STM32F103C8
Xtal (MHz): 8.0
Operating system: None
System Viewer File:
STM32F103xx.svd
Use Custom File
Code Generation
ARM Compiler: Use default compiler version
Use Cross-Module Optimization
Use MicroLIB
Big Endian
Read/Only Memory Areas
default off-chip Start Size Startup
ROM1:
ROM2:
ROM3:
on-chip
IROM1: 0x8000000 0x10000
IROM2:
Read/Write Memory Areas
default off-chip Start Size NoInit
RAM1:
RAM2:
RAM3:
on-chip
IRAM1: 0x20000000 0x5000
IRAM2:
OK
Cancel
Defaults
Help

SPI Basics

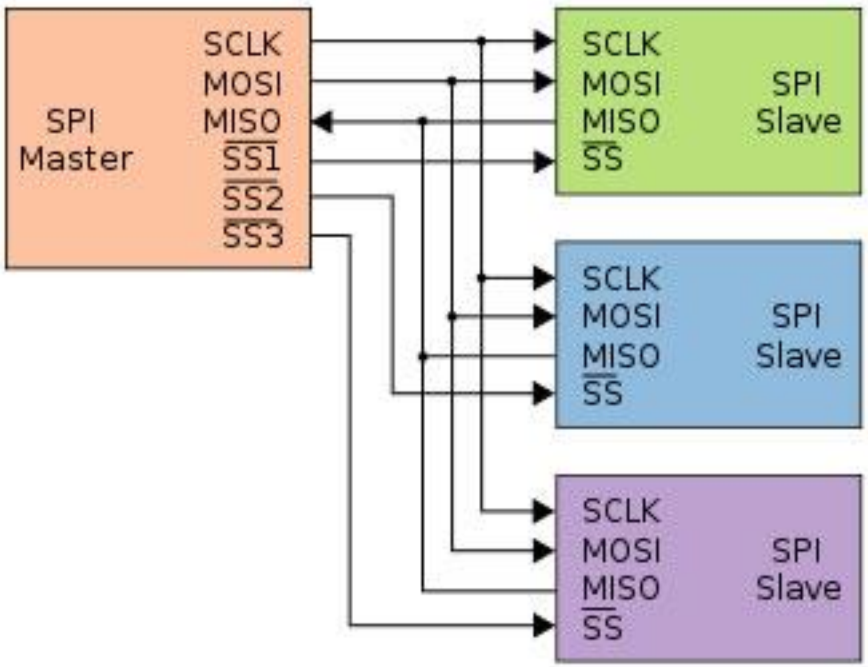
- **A communication protocol using 4 wires**
  - Also known as a 4 wire bus
- **Used to communicate across small distances**
- **Multiple Slaves, Single Master**
- **Synchronized**
- **Capabilities**
  - Always Full Duplex
  - Multiple Mbps transmission speed
  - Transfers data in 4 to 16 bit characters Multiple slaves

SPI Protocol

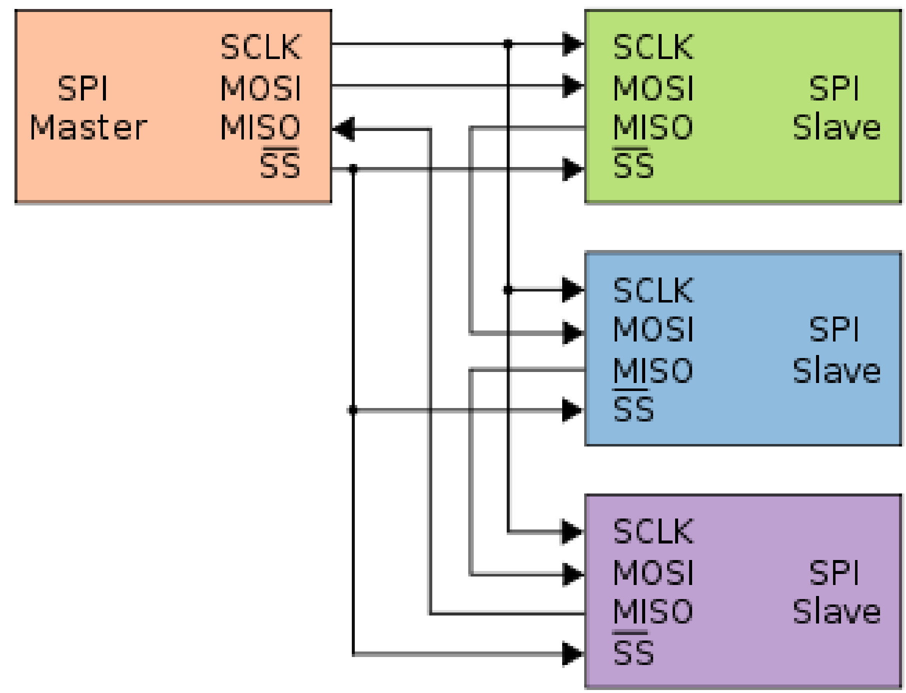
- **Wires:**
  - MOSI: Master Out Slave In
  - MISO: Master In Slave Out
  - SCLK: System Clock
  - SS: Slave Select 1…N
- **Master Set Slave Select low**
- **Master Generates Clock**
- **Shift registers shift in and out data**



Diagram



Master and multiple
independent slaves



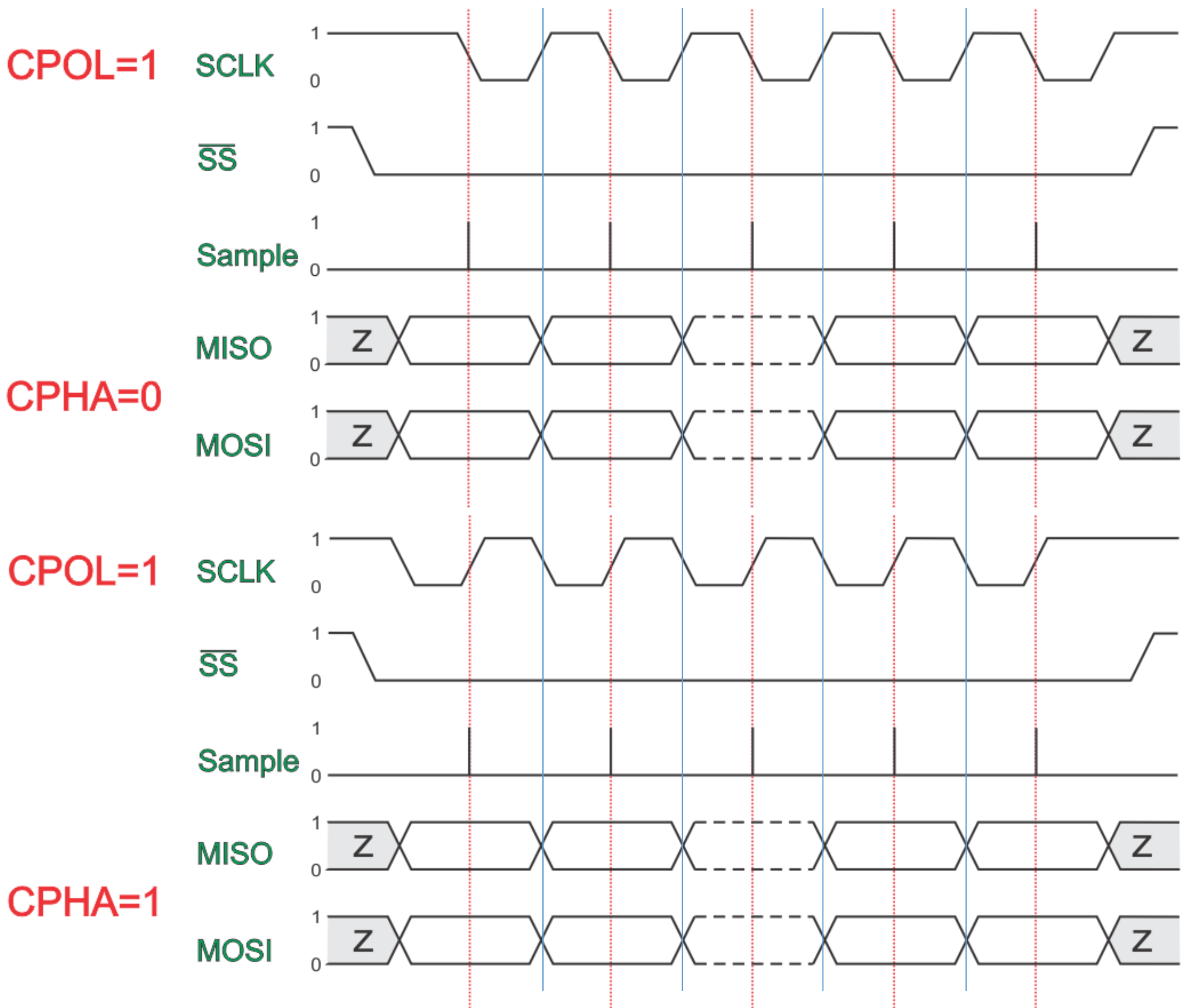
Master and multiple
daisy-chained slaves

Clock Phase

- **Two phases and two polarities of clock**
  - Four modes defined by
    - CPOL (Clock POLarity)
    - CPHA (Clock PHAse)

- **Master and selected slave must be in same mode**
- **Master must change polarity and phase to communicate with slaves of different numbers**

CPOL = 1

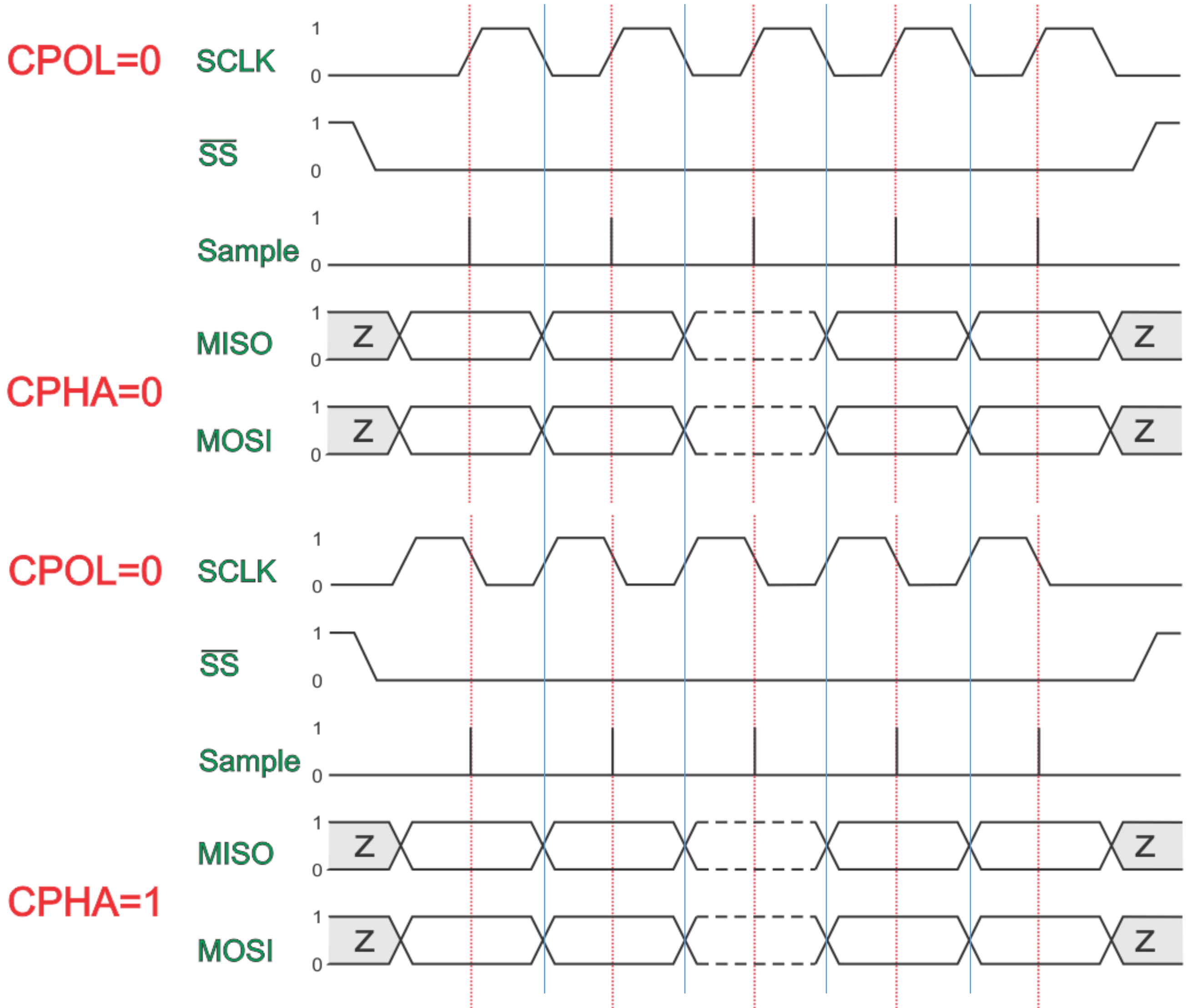


CPOL=0: Base value of the clock is one

CPHA=0:
- Data captured on rising edge,
- Data propagated at falling edge

CPHA=1:
- Data captured on falling edge,
- Data propagated at rising edge

CPOL = 0



CPOL=0: Base value of the clock is zero

CPHA=0:
- Data captured on rising edge,
- Data propagated at falling edge

CPHA=1:
- Data captured on falling edge,
- Data propagated at rising edge

Use GPIOs to mimic SPI

- **Master**
- **用GPIO来模拟SPI时序**
- **CLK:输出**
- **CS:输出**
- **MISO:???**
- **MOSI:???**
- **优点**：移植很方便，代码只需要简单修改就可以使用在其他芯片上；
- **缺点**：控制IO麻烦，对时序要求高；

初始化

- 初始化输入输出

```
GPIO_InitTypeDef GPIO_InitStructure;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC,ENABLE);
CS：
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3；
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
SCLK：
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
```

- 初始化输入输出

MOSI：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

MISO：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_In_Floating;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

定义动作

CS:

```
#define CS_H GPIO_SetBits(GPIOB, GPIO_Pin_3)
#define CS_L GPIO_ResetBits(GPIOB, GPIO_Pin_3)
```

SCLK:

```
#define SCLK_H GPIO_SetBits(GPIOB, GPIO_Pin_5)
#define SCLK_L GPIO_ResetBits(GPIOB, GPIO_Pin_5)
```

MOSI:

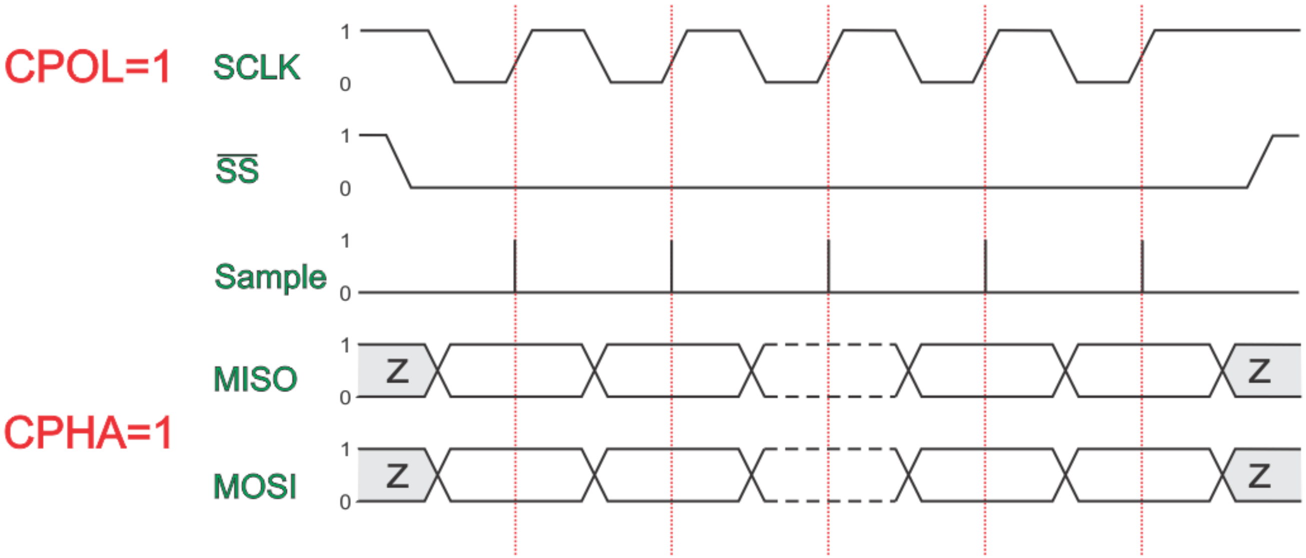
```
#define MOSI_H GPIO_SetBits(GPIOB, GPIO_Pin_4)
#define MOSI_L GPIO_ResetBits(GPIOB, GPIO_Pin_4)
```

MISO:

```
#define MISO GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_6)
```

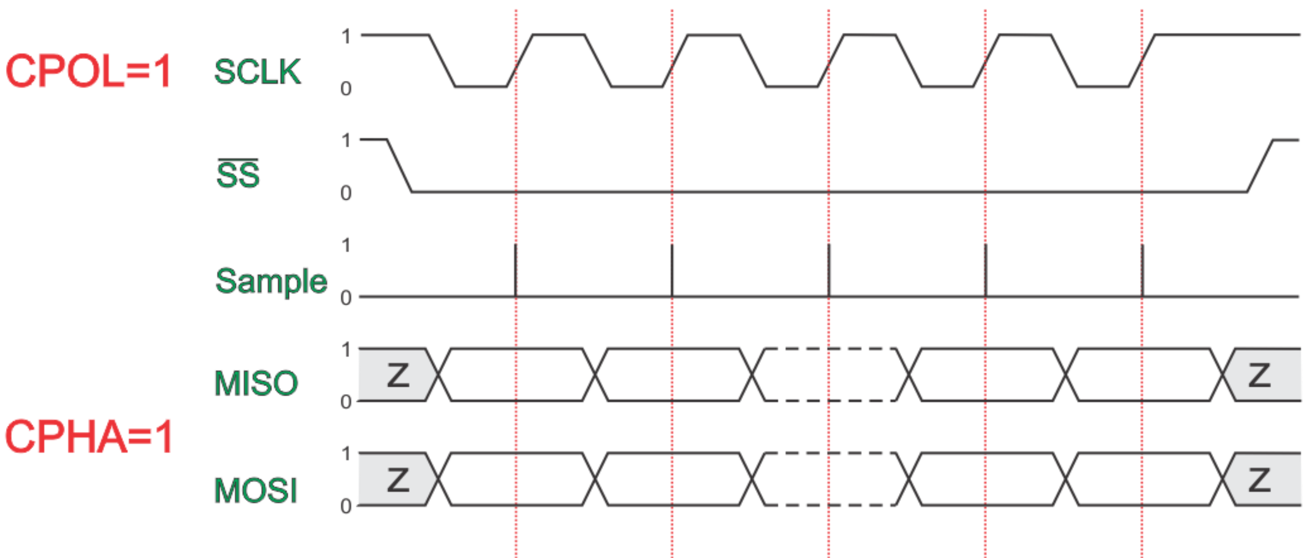
SPI发送

```
unsigned char SPI_SendByte(unsigned char data)
{
    unsigned char i;
    unsigned char temp=0;
    CS_L;
    for(i=8;i>0;i--)
    {
        if(data &0x80)
            MOSI_H;
        else
            MOSI_L;
        data <<=1;
         SCLK_L;
         __nop();
          __nop();
          __nop();
         SCLK_H;
}
          CS_H;
    return temp;
}
```



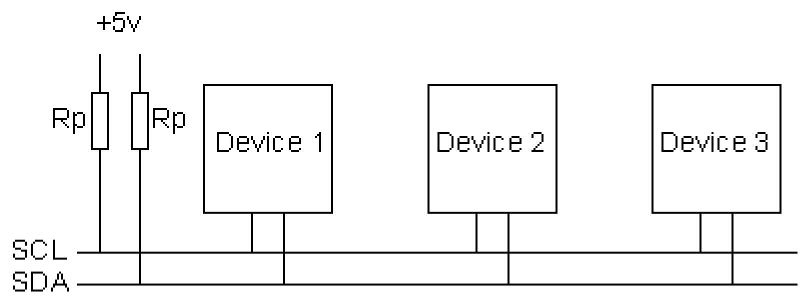
SPI接收

```
unsigned char SPI_ReadByte(void)
{
   unsigned char i;
   unsigned char temp=0;
   CS_L;
   for(i=8;i>0;i--)
   {
       SCLK_L;
       __nop();
        __nop();
        __nop(); //delay
      temp<<=1;
      if(MISO)
      {
          temp= temp|0x01;
      }
       SCLK_H;
      __nop(); //delay
}
        CS_H;
   return temp;
}
```



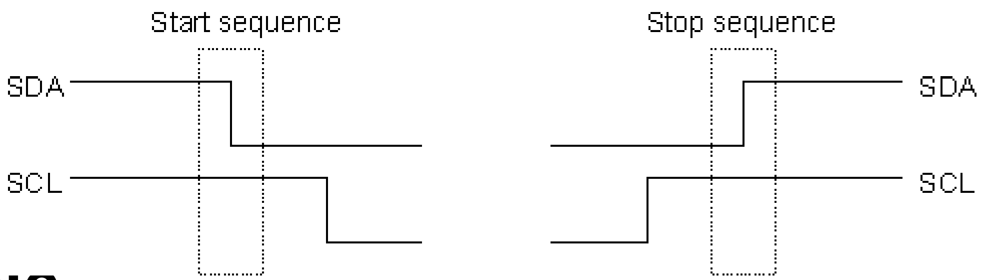
Physical I2C bus

- Two wires, called SCL and SDA.
- SCL is the clock line. It is used to synchronize all data transfers over the I2C bus.
- SDA is the data line.
- A third wire used for share GND
- Must provide pull-up resistors to the 5v supply

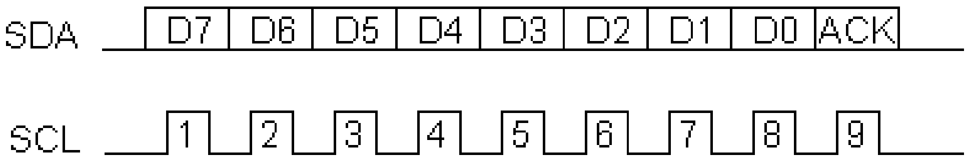


I2C Protocol (master and slave)

- **The start sequence and stop sequence**

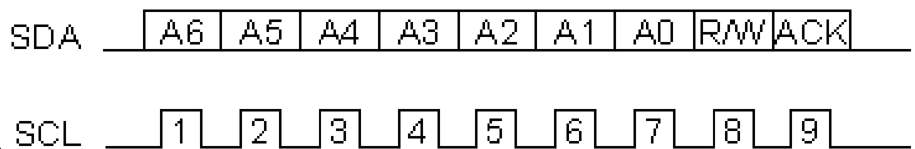


- **Data Sample**

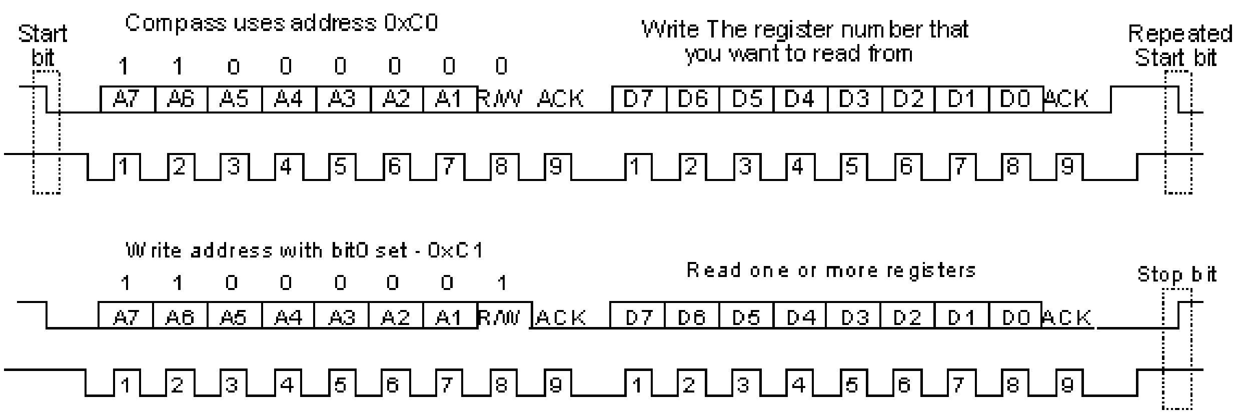


I2C Protocol (master and slave)

- **Addressing Slave Device**



- **Read data from slave device**



UART (Universal Asynchronous Receiver/Transmitter)

- The key component of the serial communications subsystem of a computer.
- Involves the data transfer for each direction.
- No clock is transmitted with serial data.
- Takes bytes of data and transmits the individual bits in a sequential fashion.

Use GPIOs to mimic I2C

- **Write by yourself!!!**