电子科学与技术专业课

嵌入式系统

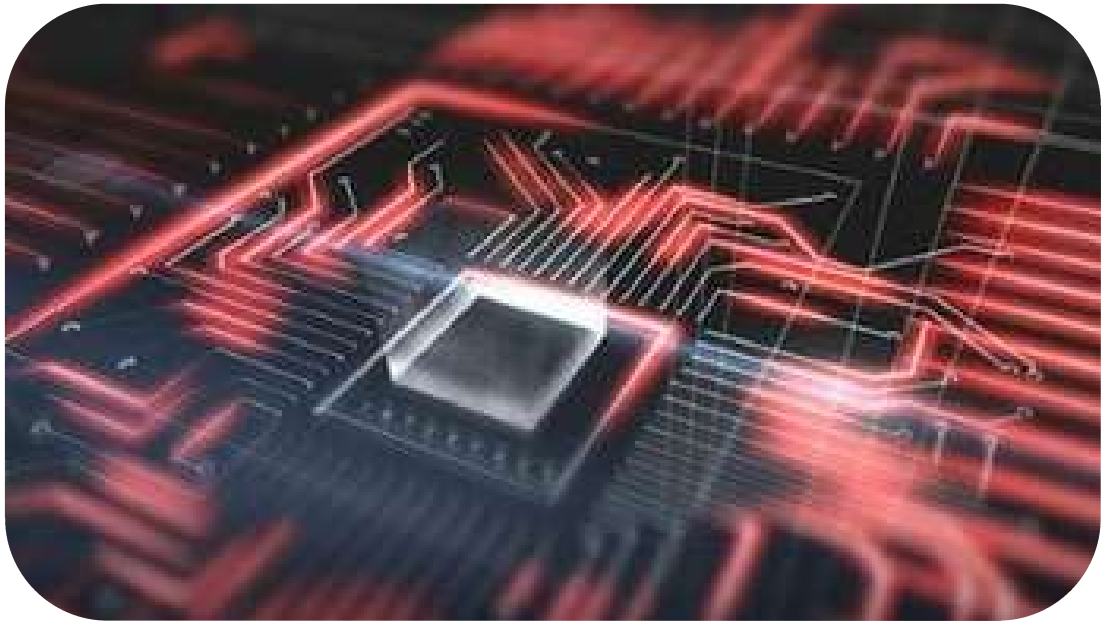
Embedded System

信息学院 光电子系

第5章 通信接口与总线

- 5.1 通信概述
- 5.2 异步串行通信UART
- 5.3 串行外设接口SPI
- 5.4 集成电路总线I^2C

嵌入式系统(EMBEDDED SYSTEM)

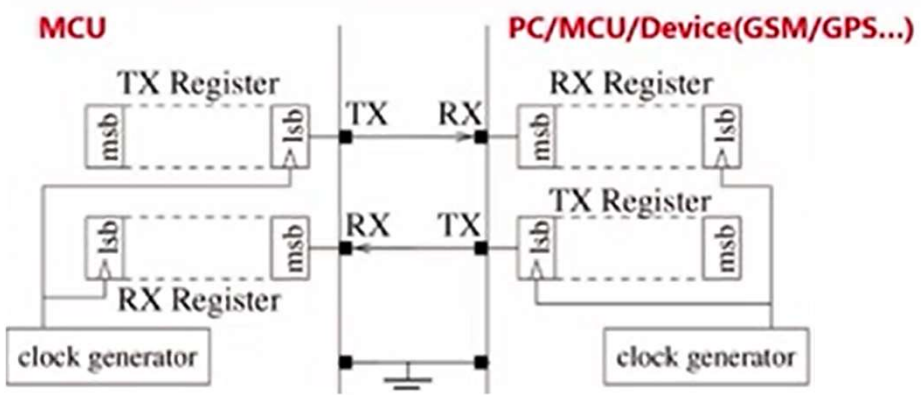


第5章 通信接口与总线

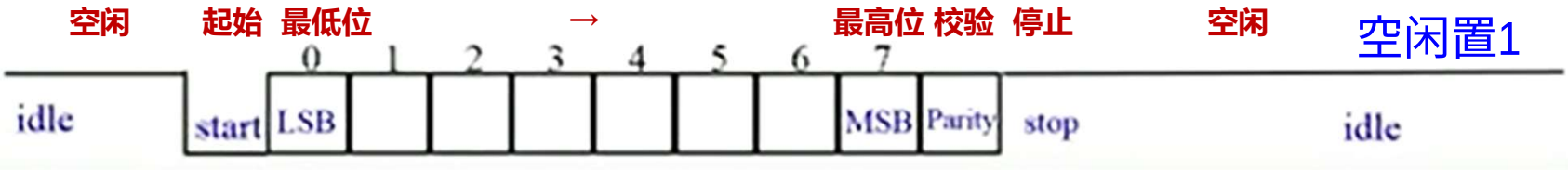
❀ **5.3 串行外设接口SPI**

回顾 UART异步串行通信

- 如何通信?
  - 发端TX，收端RX
  - 收发引脚，两对
  - 共地
  - 独立时钟，约定速率



- 帧格式 Frame format



5.3 串行外设接口SPI

- 本节内容
  - SPI通信简介
  - STM32的SPI
  - SPI常用库函数
  - SPI编程实例 – OLED/LCD显示屏驱动
  - 编程练习E4： SPI接口显示屏编程

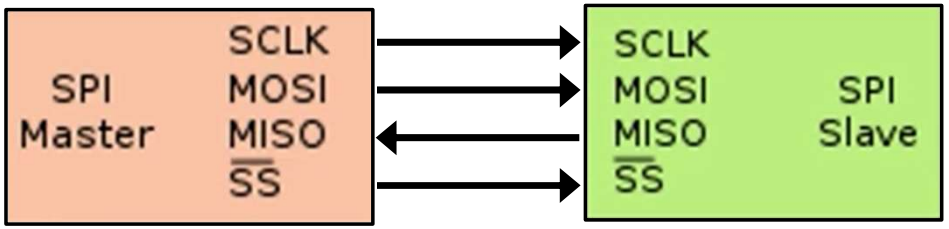
SPI通信简介

- SPI – Serial Peripheral Interface

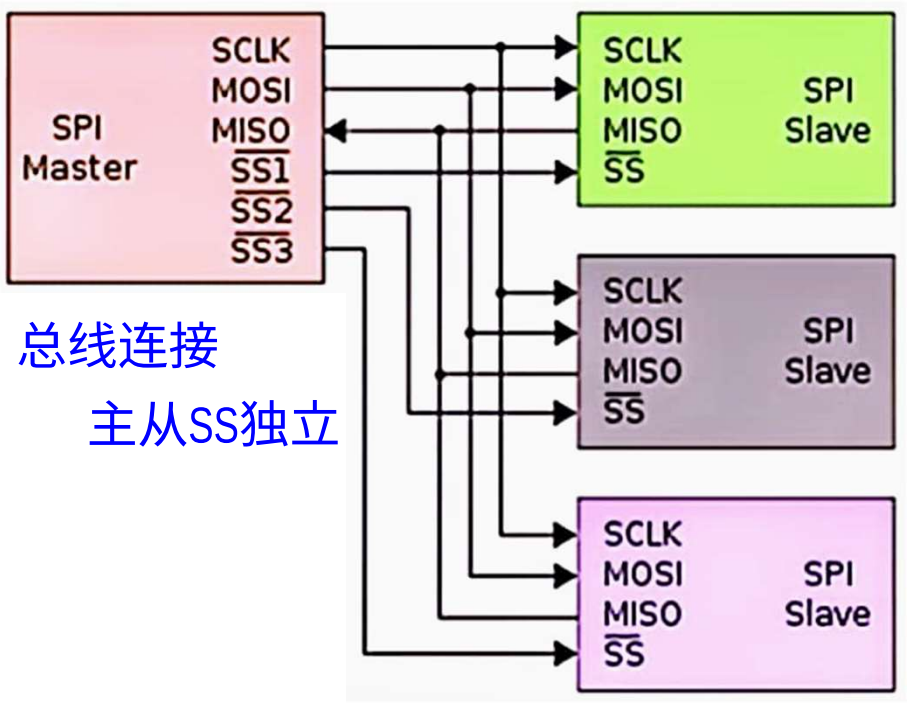
  -- 同步串行外设接口

- 方便连接各种外设/芯片

  -- ADC / LCD / ROM / DAC / Sensor ……

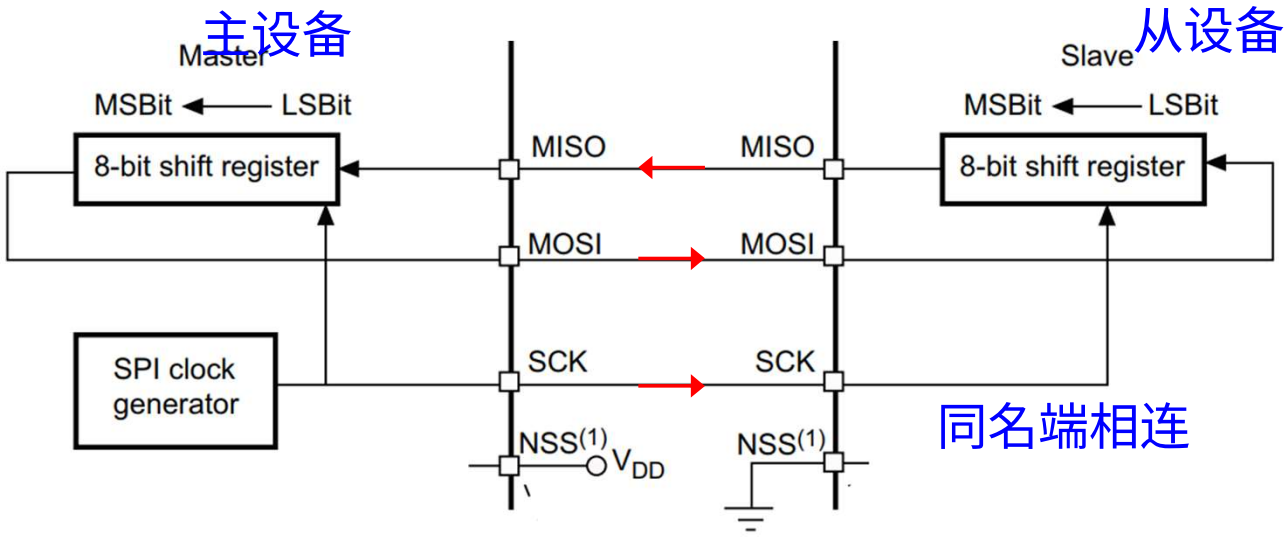


四线制连接



多机连接

SPI四线连接



NSS连接决定主从设备，接VCC为主设备，GND为从设备

- 信号定义

MOSI – Master Out Slave In，主出从入，主设备发送到从设备的信号

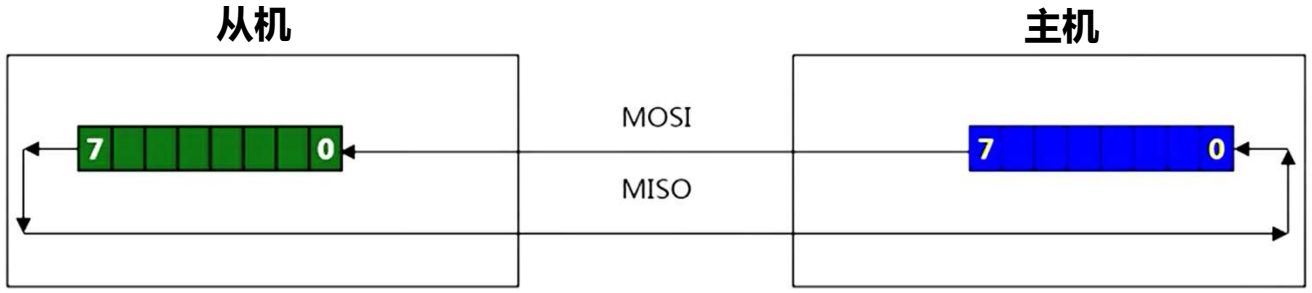
MISO – Master In Slave Out，主入从出，从设备发送到主设备的信号

SCK – Serial Clock，由主设备产生的SPI工作时钟，每个SCK周期完成一个bit的传输

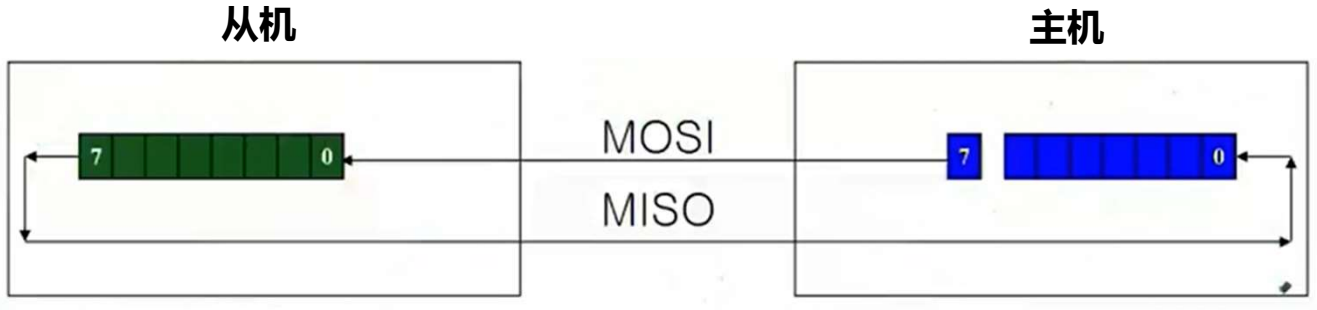
SS – Slave select，从设备选择端，低电平有效；对于主设备，该端为高电平。

如何工作?

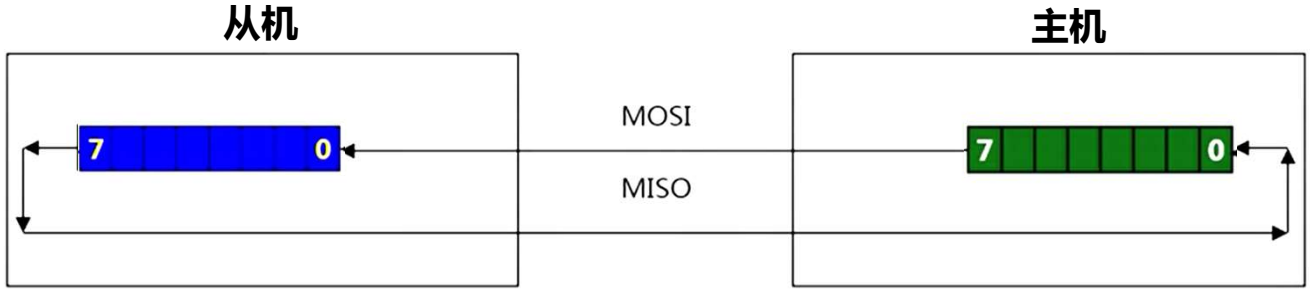
各有一个字节



完成1bit交换



重复8次之后



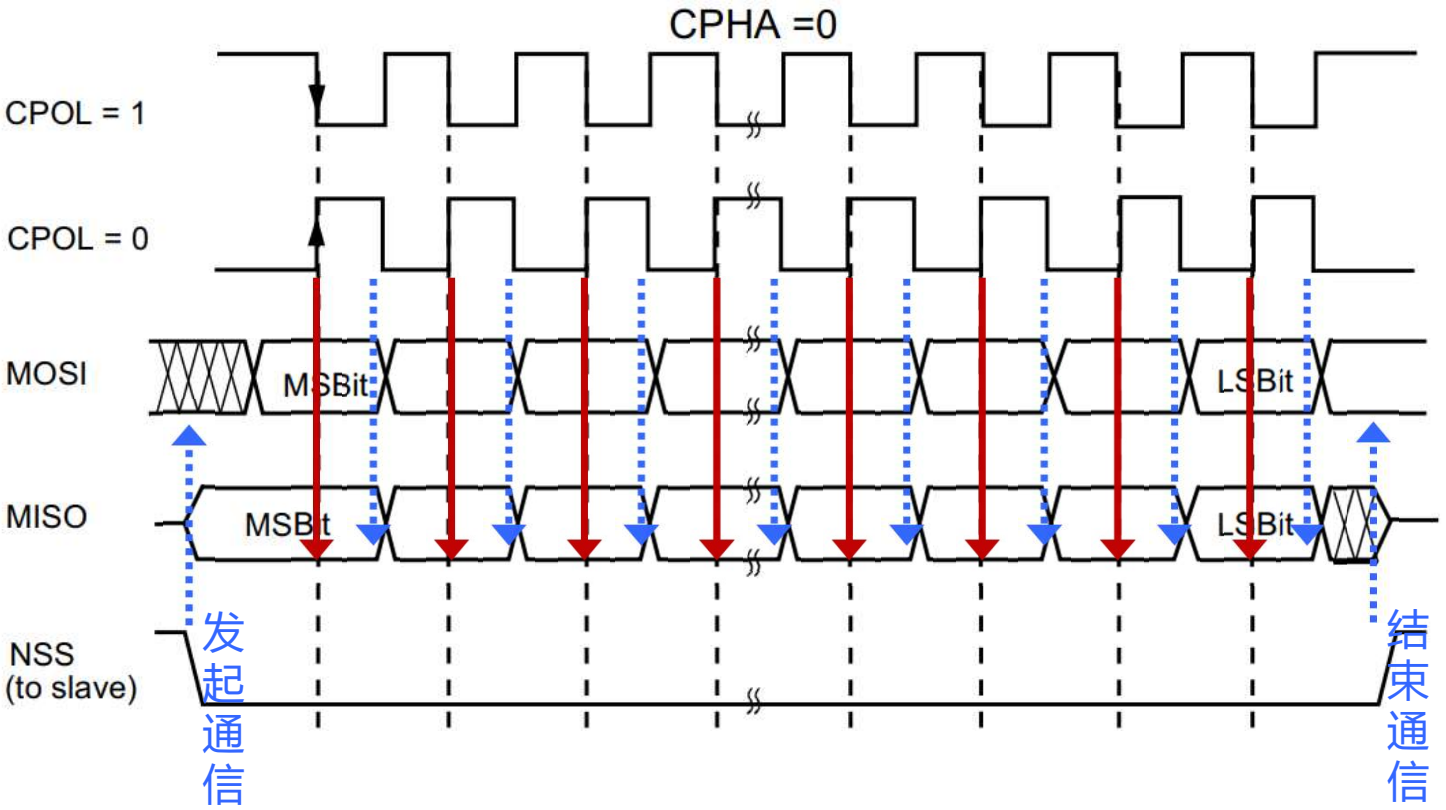
SPI通信特征

- 串行 Serial
- 同步 Synchronous
- 全双工运行 Full duplex operation (one master)
- 主模式/从模式 Master mode / slave mode
- 点对点 或 总线 Point-to-point or Bus
- 灵活的时钟极性/相位格式 Flexible clock polarity/phase format
- 帧长度从4bit到16bit可变化 Variable frame size from 4 to 16bits

SPI 信号 – SCK相位与极性（模式0）

奇数边沿采样，偶数边沿交换数据

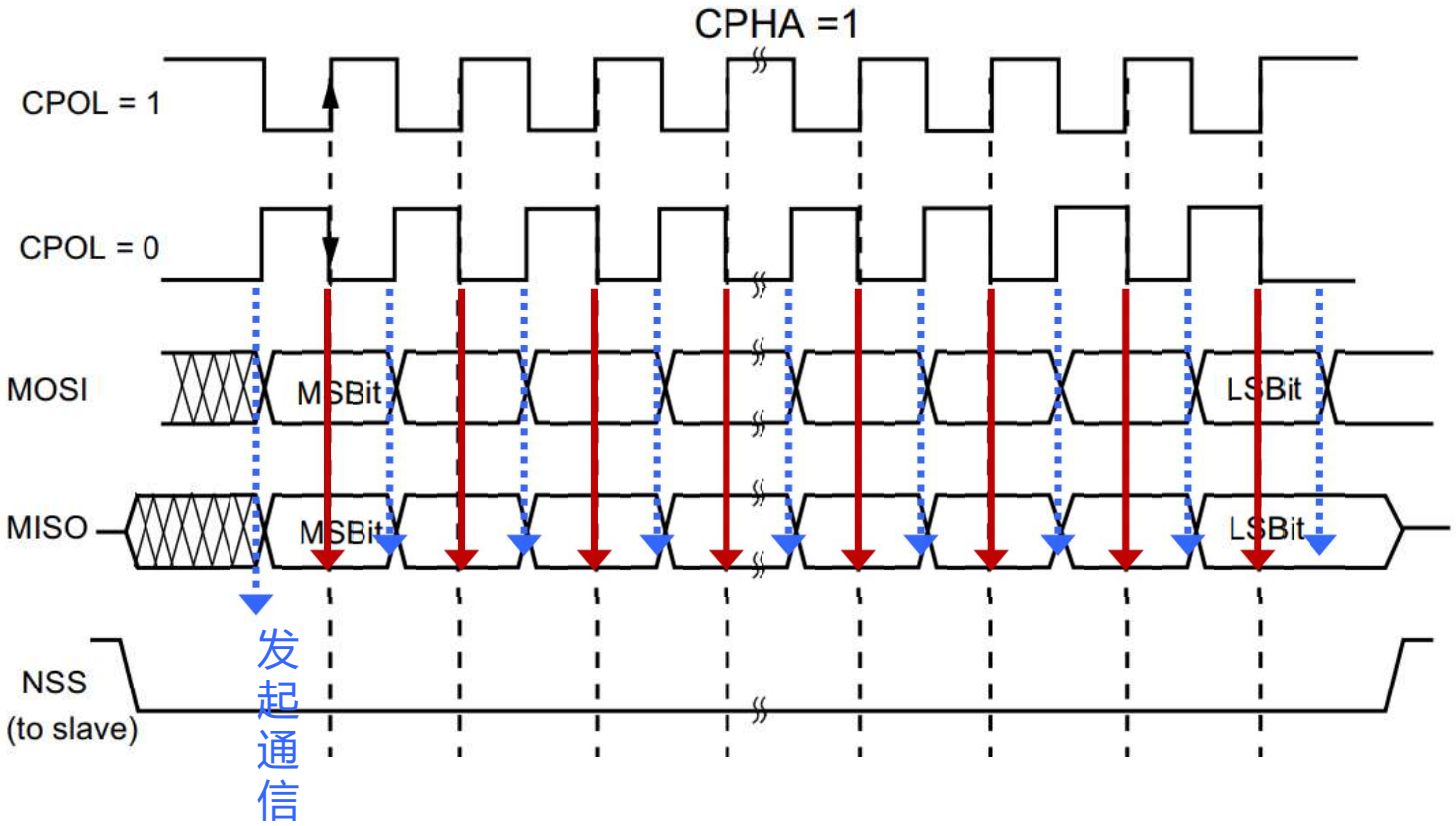
- 时钟相位 CPHA
  - -- Clock Phase 时钟相位
  - -- 决定时钟采样的边沿
  - -- 模式0：奇数边沿采样
- 时钟极性 CPOL
  - -- Clock polarity
  - -- 决定时钟空闲电平
  - -- 极性1 空闲电平为高
    反之为低



SPI 信号 – SCK相位与极性（模式1）

- 时钟相位 CPHA 模式1
  - -- 第1个边沿发起通信
  - -- 偶数边沿采样数据
- 可多字节连续通信
  - -- SS一直保持低

偶数边沿采样，奇数边沿交换数据

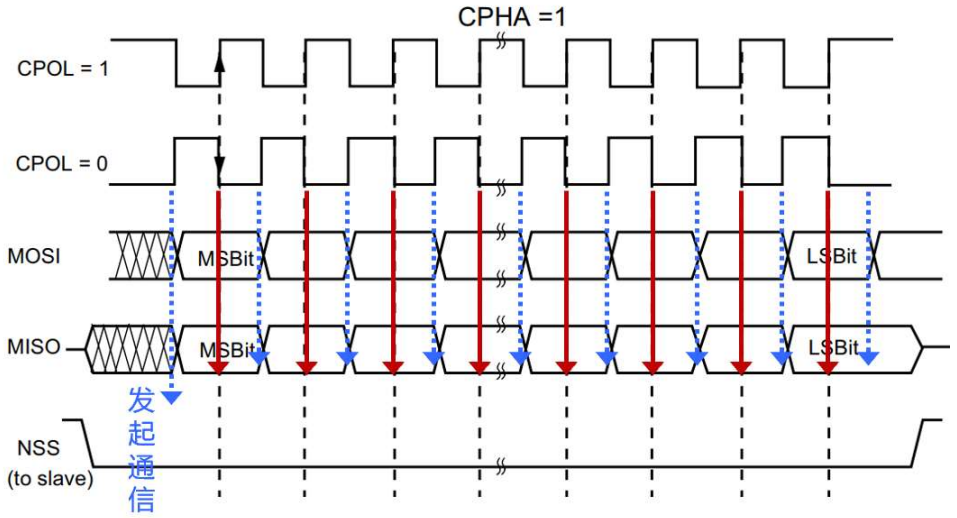


你是否清楚了解了SPI?

- 小组SPI通信游戏
  - 一个人作为master，另一人作为slave
  - Frame size =8，相位CPHA =1，CPOL =0
  - 左手代表SCK，右手代表MOSI
  - 举手代表“1”，手放下代表“0”
  - master负责向slave同伴发送信息

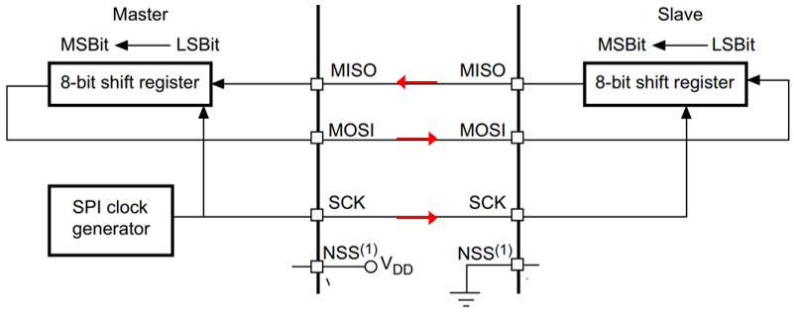
    -- ‘U’ = 0b0101010

    -- ‘Z’ = ? = 0b01011010



主模式 – Master mode

- 控制整个传输过程
  - 通过SS信号选择对应的通信从节点
  - 决定SCK的波特率，相位，极性
  - 产生SCK时钟信号
  - 驱动MOSI信号
  - 采样MISO信号



- CPU通过向SPIx_DR数据寄存器写入数据来启动一次传输过程

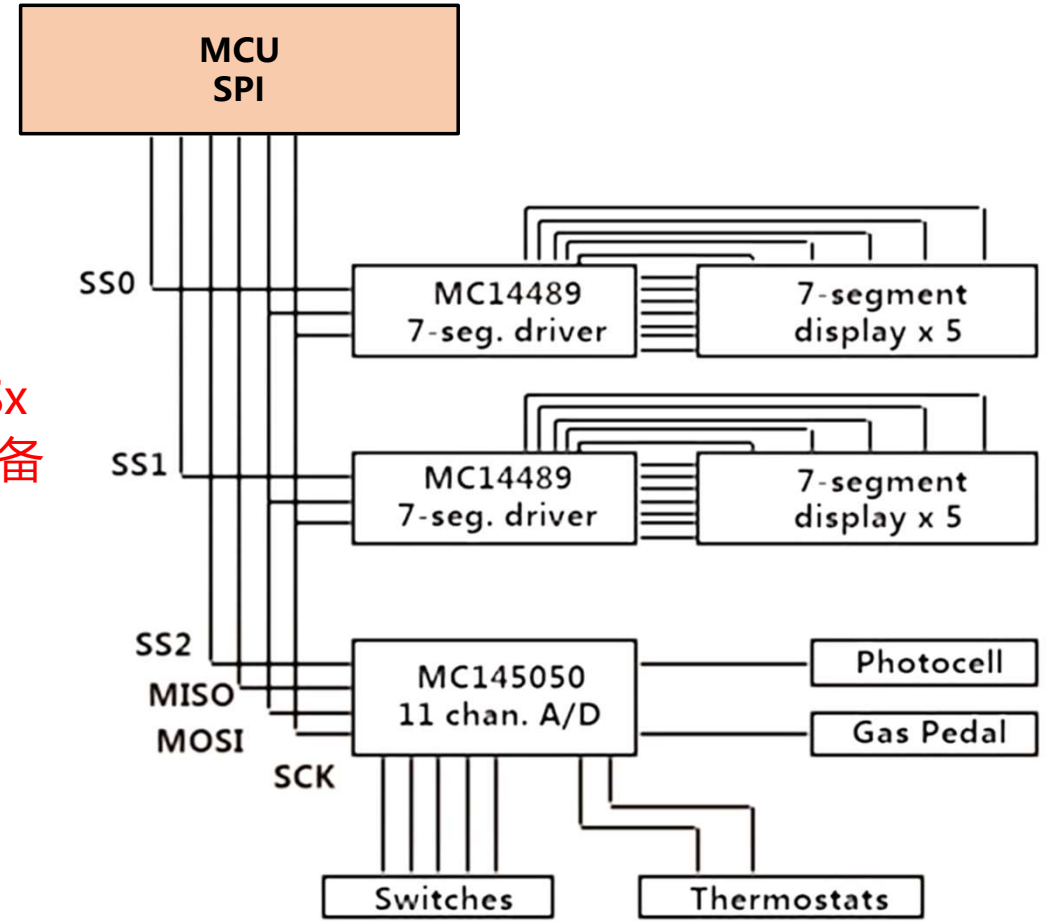
从模式 – Slave mode

- 控制整个传输过程
  - 当SS信号被选通时才激活
  - 根据预先约定的相位/极性来检测SCK信号
  - 驱动MISO信号
  - 采样MOSI信号

Points to notice

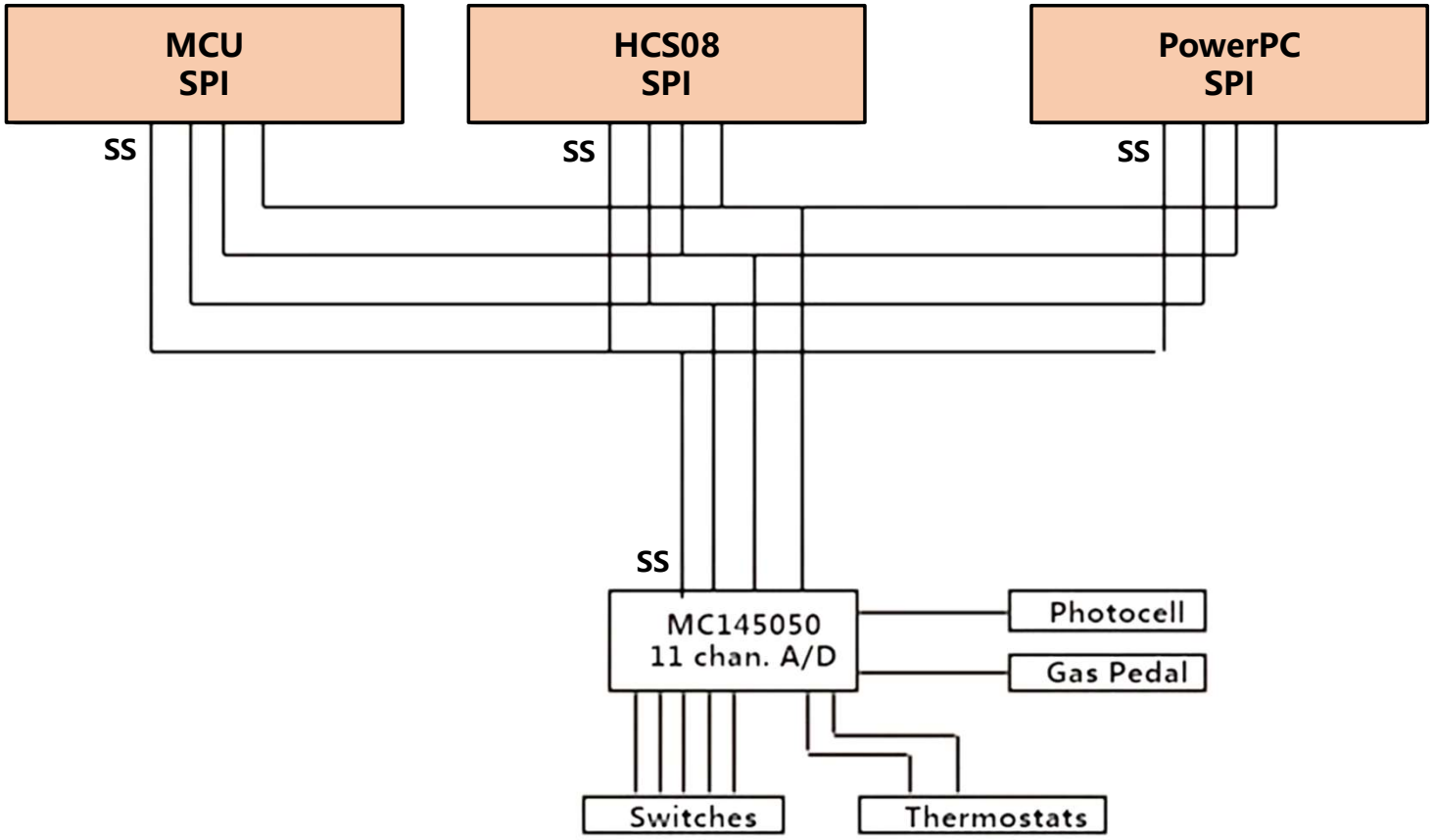
- 通常，SPI是点对点结构的
  -- 多个从节点?
  -- 多个主节点?
- 必须预先约定SCK时钟的相位/极性和数据帧位数
- 从节点的CPU需要在数据帧开始前准备好待发数据
- SCK信号必须干净，毛刺会带来严重干扰，频率双方可接受

其他：多个slave



拉低SSx
使能设备

其他：多个master

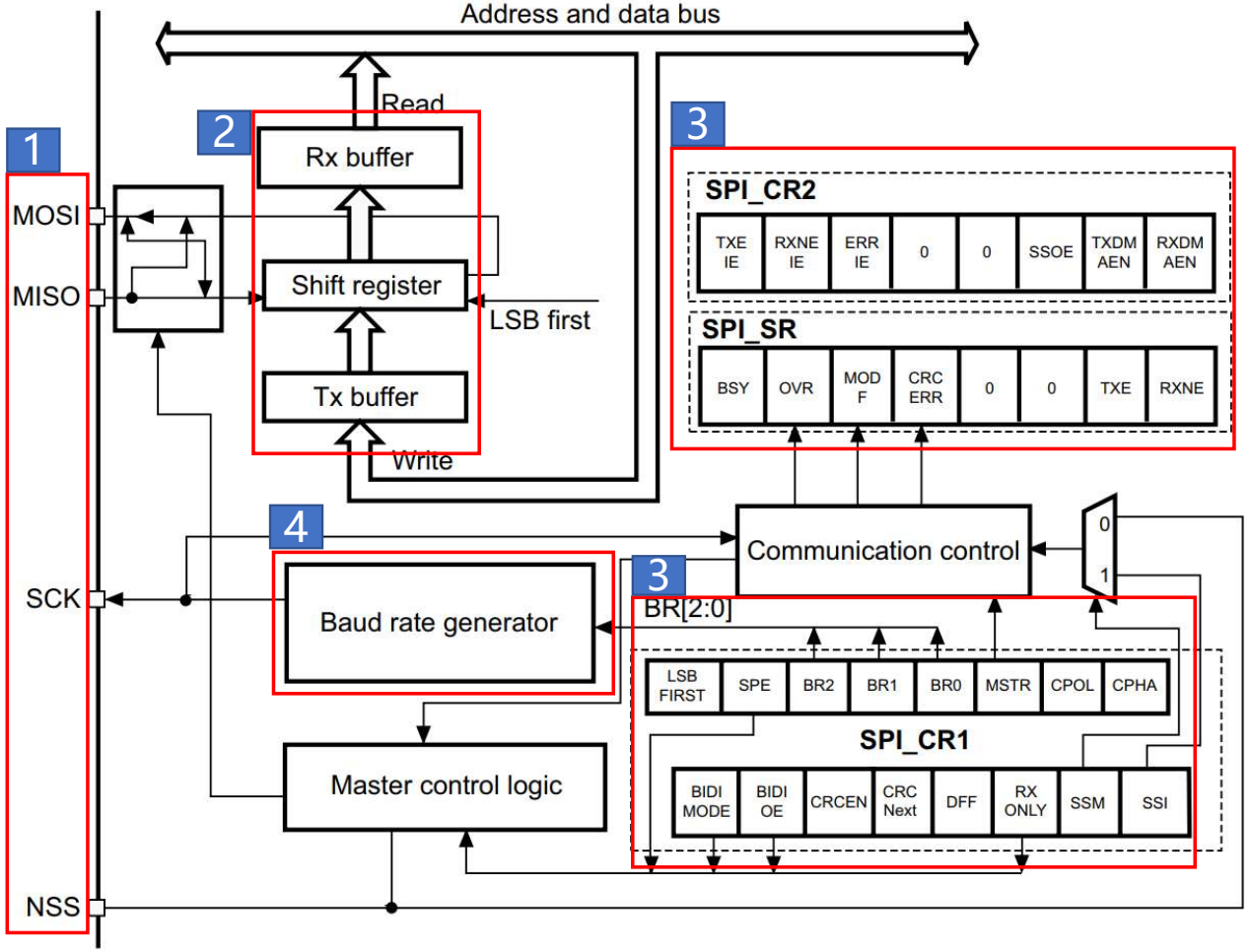

MCU
SPI
HCS08
SPI
PowerPC
SPI
SS
SS
SS
SS
MC145050
11 chan. A/D
Photocell
Gas Pedal
Switches
Thermostats

5.3 串行外设接口SPI

- Next……
  - SPI通信简介
  - STM32的SPI
  - SPI常用库函数
  - SPI编程实例 – OLED/LCD显示屏驱动
  - 编程练习E4： SPI接口显示屏编程

SPI 基本结构

- **① 功能引脚**
  - -- MOSI、MISO、SCK、NSS
- **② 数据寄存器**
  - -- Rx & Tx buf. 收发缓冲器
  - -- Shift reg. 移位寄存器
- **③ 控制/状态寄存器**
  - -- 主从配置
  - -- 收发控制
  - -- 波特率分频值2^n (n=1~8)
- **④ 波特率生成**
  - -- 生成时钟信号



SPI引脚资源

- **F407IG芯片**

  --数据手册 Chapter 3：Pinouts...description – Table 9. Alternate function mapping

引脚	APB1 总线 (max. 42MHz)		APB2 总线 (max. 84MHz)
	SPI2	SPI3	SPI1
NSS	PB9/PB12	PA4/PA15	PA4/PA15
SCK	PB10/PB13	PB3/PC10	PA5/PB3
MISO	PB14/PC2	PB4/PC11	PA6/PB4
MOSI	PB15/PC3	PB5/PC12	PA7/PB5

算一算：SPI1的最大和最小传输波特率是多少？

max. = 84M/2^1 = 42Mbps，min. = 84M/2^8 = 328.125Kbps

5.3 串行外设接口SPI

- Next……
  - SPI通信简介
  - STM32的SPI
  - SPI常用库函数
  - SPI编程实例 – OLED/LCD显示屏驱动
  - 编程练习E4： SPI接口显示屏编程

常用库函数

- SPI_Init (SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct)

  功能：将串口 SPIx 按照结构体 SPI_InitStruct 的参数进行初始化，结构体格式如下

```
typedef  struct
{
  uint32_t      SPI_Direction;          /* 数据传输方向，全双工、仅收、仅发 */
  uint16_t      SPI_Mode;               /* (主从)模式，主机、从机 */
  uint16_t      SPI_DataSize;           /* 数据帧长，16位、8位 */
  uint16_t      SPI_CPHA;               /* 时钟相位，奇数沿采样、偶数沿采样 */
  uint16_t      SPI_CPOL;               /* 时钟极性，空闲电平为低或高 */
  uint16_t      SPI_NSS;                /* 从设备选择方式，软件内置、硬件外控 */
  uint16_t      SPI_BaudRatePrescaler;  /* 波特率分频比，2^n，n = 1~8 */
  uint16_t      SPI_FirstBit;           /* 首发bit，MSB或LSB */
} SPI_InitTypeDef;                      (具体选择参见stm32f4xx_spi.h第147行起)
 //定义初始化结构体变量
 SPI_InitTypeDef   SPI_InitStructure;
```

常用库函数

- SPI_I2S_SendData(SPI_TypeDef* SPIx，uint16_t Data)

  功能：将数据Data通过串口SPIx口发送出去，Data有效部分情况

  -- Data[15:0]，16位，用于数据帧长16bits

  -- Data[7:0]，8位，用于数据帧长8bits

- uint16_t SPI_I2S_ReceiveData(SPI_TypeDef* SPIx)

  功能：从SPIx接收一个数据并将其返回，该数据两种有效位数同上

- SPI_Cmd(SPI_TypeDef* SPIx, FunctionalState NewState)

  功能：使能/关闭SPIx，NewStat = ENABLE或DISABLE

常用库函数

- FlagStatus SPI_I2S_GetFlagStatus (SPI_TypeDef* SPIx, uint16_t SPI_I2S_FLAG)
  - 检测指定的SPIx 的状态标志并返回其值， SPI_I2S_FLAG常用标志有

    -- SPI_FLAG_TXE: Transmit buffer empty，= 1 数据发送完成，=0 未发完

    -- SPI_FLAG_RXNE: Receive buffer not empty，=1 数据已接收待读取，=0 未收完
  - 返回值类型 FlagStatus 为枚举，定义如下

    typedef enum { RESET = 0, SET = 1} FlagStatus;

5.3 串行外设接口SPI

- Next.....
  - SPI通信简介
  - STM32的SPI
  - SPI常用库函数
  - SPI编程实例 – OLED/LCD显示屏驱动
  - 编程练习E4： SPI接口显示屏编程

SPI编程

初始化函数?

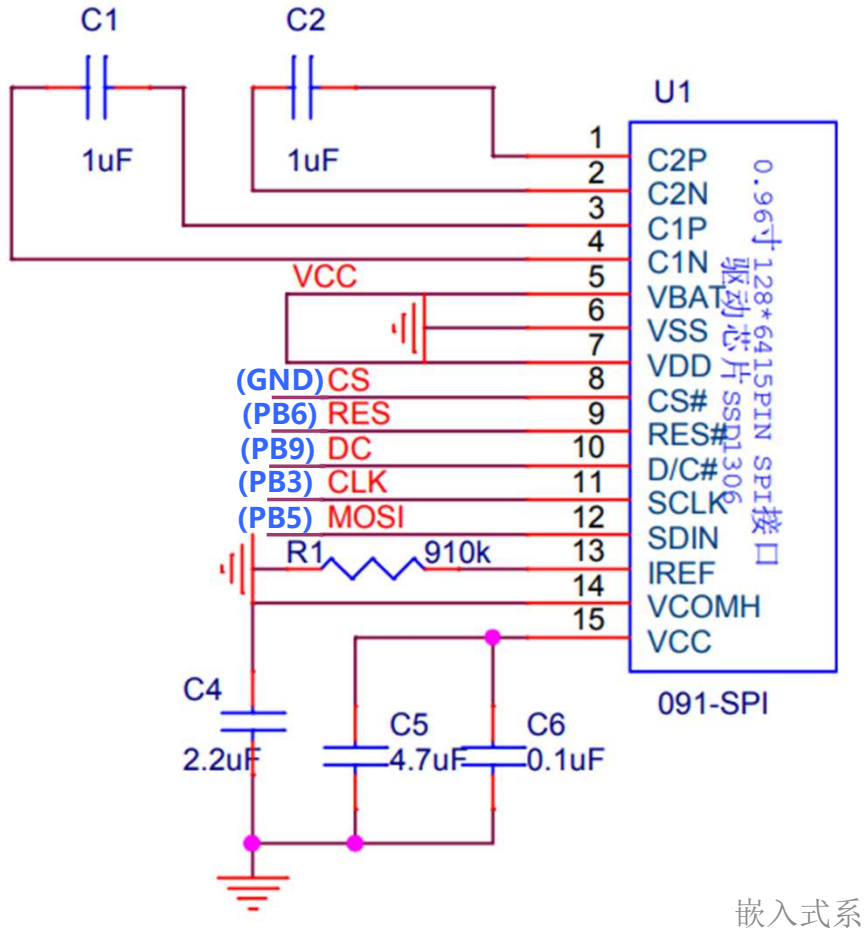
```
void SpixInit(void)
{
 ......
}
```

输入输出函数?

```
uint8_t  SpixRw (uint8_t  data)
{
 ......
}
```

SPI → OLED display

OLED驱动器SSD1306接线图



GND：CS　片选

PB6：　RES　复位脚

PB9：　DC　数据模式/命令模式切换

PB3：　CLK　时钟

PB5：　MOSI　数据输入

STM32F407IG：SPI1引脚

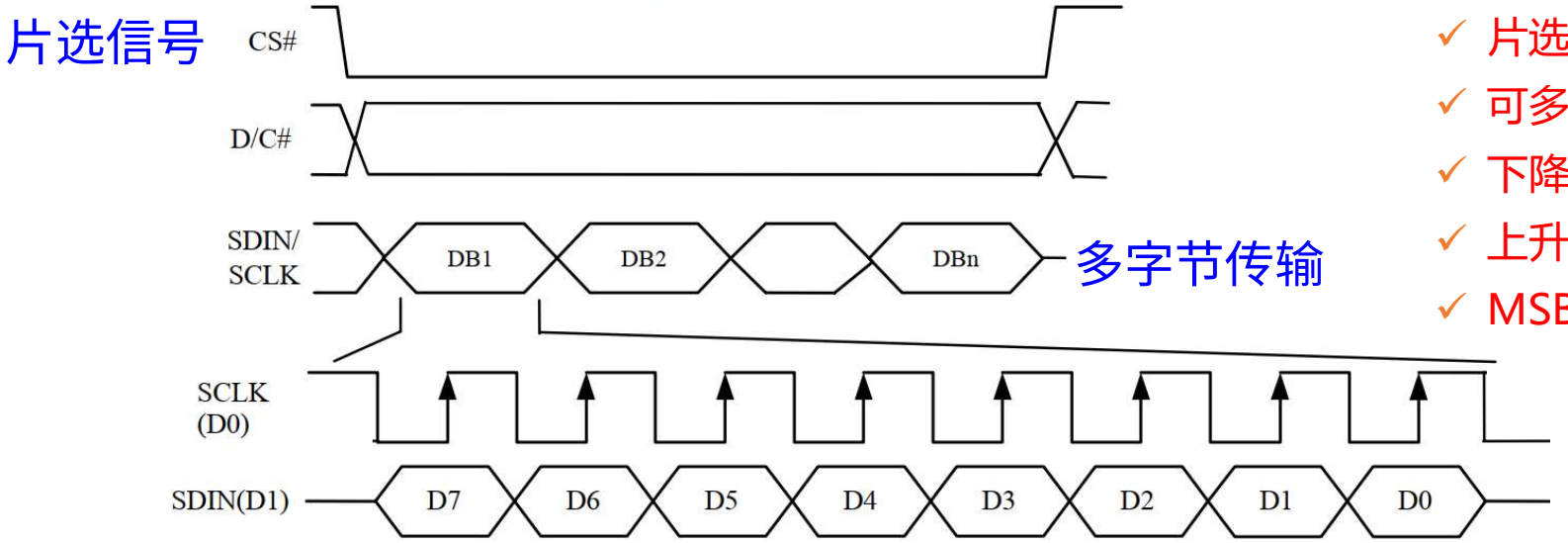
主从选择脚

SCLK	MOSI	MISO	NSS
PB3	PB5	PB4	PA15

SPI → OLED display 接口情况?

SSD1306数据手册：Chapter8.1.3-MCU Serial Interface (4-wire SPI)

Figure 8-5 : Write procedure in 4-wire Serial interface mode



- ✓ 片选拉低
- ✓ 可多字节传输
- ✓ 下降沿发出
- ✓ 上升沿采样
- ✓ MSB在先

交流特性

Chapter13-AC characteristics

Table 13-4 : 4-wire Serial Interface Timing Characteristics

(V_{DD} - V_{SS} = 1.65V to 3.3V, T_A = 25°C)

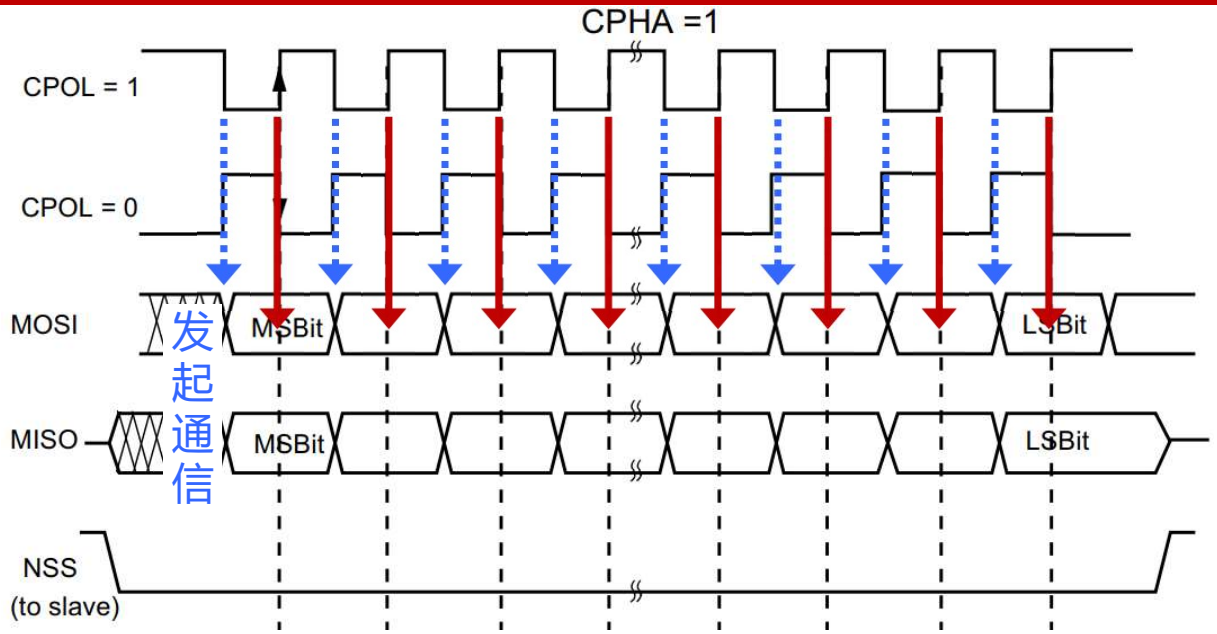
Symbol	Parameter	Min	Typ	Max	Unit
t_{cycle}	Clock Cycle Time	100	-	-	ns

波特率max. 10Mbps

限定最小周期，最大时钟速度

SSD1306时序 vs SPI时钟相位极性

SPI时序：



SPI配置：

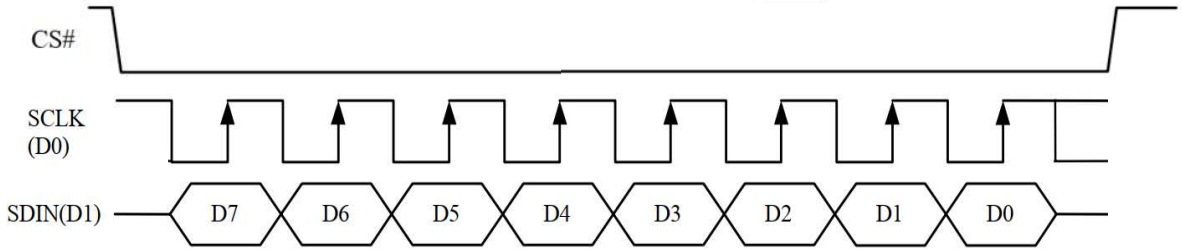
CPHA = 1（相位）

CPOL = 1（极性）

MSB首发

NSS引脚未用

SSD1306时序：



SPI编程三步走

- **Step 1：SPI初始化**
  - 打开GPIO时钟、打开SPI时钟
  - 开启GPIO引脚复用功能
  - 配置GPIO具体参数、配置SPI具体参数
- **Step 2：SPI高一级通信函数**
  - 标准函数之上的更高一级封装
  - 以字节为单位的设备读写函数
- **Step 3：SPI高级函数应用**

宏定义

```
//宏定义 – 端口和引脚
#define  OLED_Port          GPIOB

#define  OLED_SCK_Pin       GPIO_Pin_3     //PB3  时钟端
#define  OLED_SDA_Pin       GPIO_Pin_5     //PB5  数据输入-DIN-MOSI
#define  OLED_RES_Pin       GPIO_Pin_6     //PB6  复位端-RESET
#define  OLED_DC_Pin        GPIO_Pin_9     //PB9  数据/命令模式选择

//宏定义 – 基础操作
#define  OLED_RESET_L()     GPIO_ResetBits(OLED_Port,  OLED_RES_Pin)   //复位端 拉低
#define  OLED_RESET_H()     GPIO_SetBits(OLED_Port,  OLED_RES_Pin)     //复位端 升高

#define  OLED_DC_L()        GPIO_ResetBits(OLED_Port,  OLED_DC_Pin)    //命令模式
#define  OLED_DC_H()        GPIO_SetBits(OLED_Port,  OLED_DC_Pin)      //数据模式
```

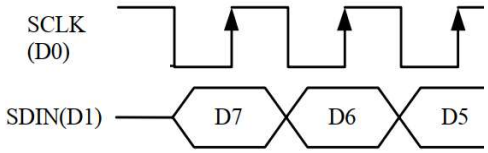
SPI编程1 – 端口初始化

```
void  OLED_SPI1_Init(void)
{   GPIO_InitTypeDef       GPIO_InitStructure;
    SPI_InitTypeDef        SPI_InitStructure;
    //打开设备时钟
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1,  ENABLE);
    //开启PB3-SCK, PB5-SDA为SPI复用功能
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource3,  GPIO_AF_SPI1);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource5,  GPIO_AF_SPI1);
    //初始化GPIO PB3-SCK, PB5-SDA 两个复用端
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_5;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    //初始化GPIO PB6-RESET, PPB9-DC 两个普通输出端
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_9;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
```

SPI编程1 – 端口初始化

……接上页

```
//配置SPI参数
SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;  /* 全双工 */
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;     /* MCU为主机模式 */
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;   /* 数据帧8bits */
SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;    /* 时钟极性，空闲(初始)电平为1*/
SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge; /* 时钟相位，偶数沿采样*/
```




```
  /*SPI1挂在APB2, BaudRate = fAPB2/16 = 84M/16 = 5.25 MHz*/
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_16;  /* 波特率分频系数 16 */
SPI_InitStructure.SPI_FirstBit=SPI_FirstBit_MSB;  /* 先发最高位MSB */
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;  /* NSS由软件控制 */
SPI_Init(SPI1, &SPI_InitStructure);  /* 完成配置 */
SPI_Cmd (SPI1, ENABLE);              /* 打开SPI1 */
}
```

最高频率10M
AC特性时钟

SPI编程2 – 高一级函数

```
// 向OLED写入一个字节“命令”
void OLED_WR_Cmd(uint8_t  cw)
{  OLED_DC_L();  // 写“命令”模式
   SPI_I2S_SendData(SPI1，cw);
   while(SPI_I2S_GetFlagStatus(SPI1，SPI_I2S_FLAG_TXE) == 0);  /* 未发完则等待 */
   delayus(2);   /* 适当延时，给OLED响应时间 */
 }
 // 向OLED写入一个字节“数据”
void OLED_WR_Dat(uint8_t  dat)
{  OLED_DC_H(); //写“数据”模式
   SPI_I2S_SendData(SPI1，dat);
   while(SPI_I2S_GetFlagStatus(SPI1，SPI_I2S_FLAG_TXE) == 0);  /* 未发完则等待 */
   delayus(2);  /* 适当延时，给OLED响应时间 */
}
```

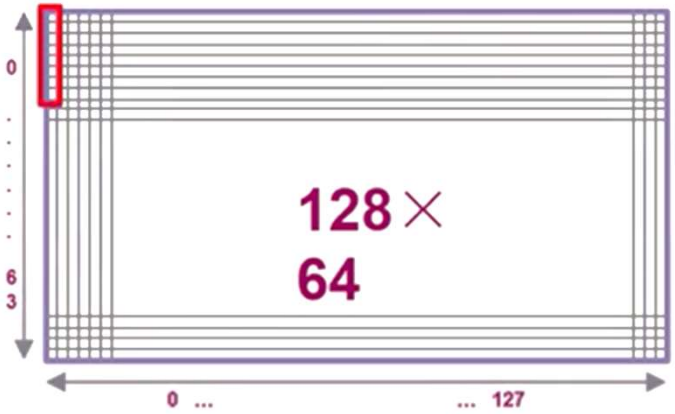
SPI编程3 - OLED初始化

```
void  OledInit (void)
{
  OLED_SPI1_Init();   //SPI端口初始化
  OLED_RESET_L();    //复位OLED
  delayus(10);          //Datasheet要求复位 >3μs
  OLED_RESET_H();  //复位完成
   //OLED初始化（参考手册要求）
  OLED_WR_Cmd(0xae);//--turn off oled panel
  OLED_WR_Cmd(0x00);//---set low column address
  OLED_WR_Cmd(0x10);//---set high column address
  OLED_WR_Cmd(0x40);//--set start line address
  OLED_WR_Cmd(0x81);//--set contrast control register
  OLED_WR_Cmd(0xcf);// Set SEG Output Current Brightness
  OLED_WR_Cmd(0xa1);//--Set SEG/Column Mapping
  OLED_WR_Cmd(0xc8);//Set COM/Row Scan Direction
  接右侧……
```

```
……接左侧
  OLED_WR_Cmd(0x80);//--set divide ratio
  OLED_WR_Cmd(0xd9);//--set pre-charge period
  OLED_WR_Cmd(0xf1);//Set Pre-Charge as 15 Clocks
  OLED_WR_Cmd(0xda);//--set com pins hardware configuration
  OLED_WR_Cmd(0x12);
  OLED_WR_Cmd(0xdb);//--set vcomh
  OLED_WR_Cmd(0x40);//Set VCOM Deselect Level
  OLED_WR_Cmd(0x20);//-Set Page Addressing Mode
  OLED_WR_Cmd(0x02);//
  OLED_WR_Cmd(0x8d);//--set Charge Pump enable/disable
  OLED_WR_Cmd(0x14);//--set(0x10) disable
  OLED_WR_Cmd(0xa4);// Disable Entire Display On (0xa4/0xa5)
  OLED_WR_Cmd(0xa6);// Disable Inverse Display On (0xa6/a7)
  OLED_WR_Cmd(0xaf);//--turn on oled panel
}
```

设置屏幕初始化

SPI编程3 – OLED字符显示



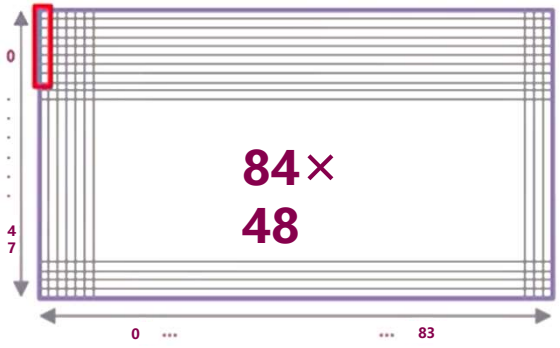
OLED

横向X：128 像素

纵向Y：64 像素

占多少字节?

128*64/8



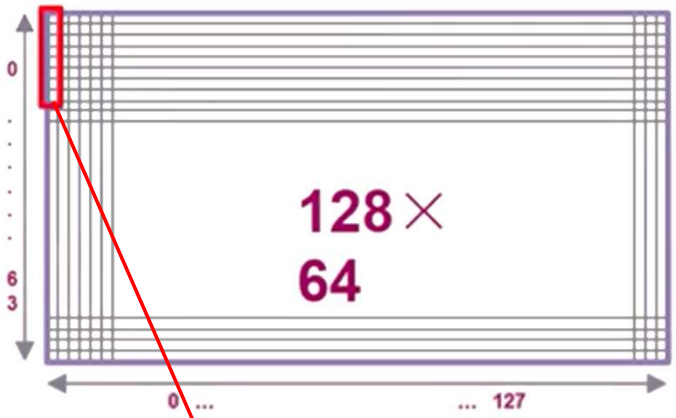
LCD

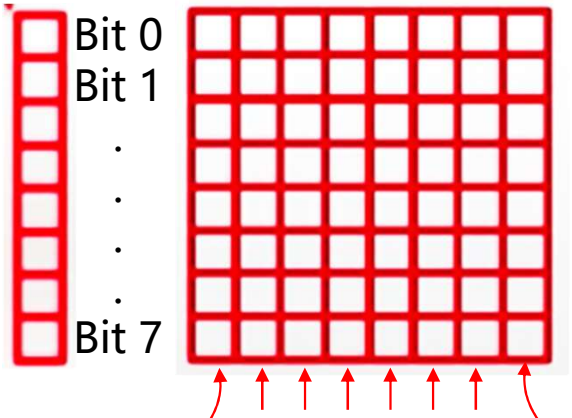
横向X：84 像素

纵向Y：48 像素

占多少字节?

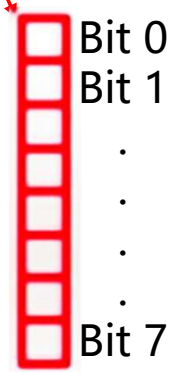
字符显示原理





如何控制一个8×8的区域?

逐列写入



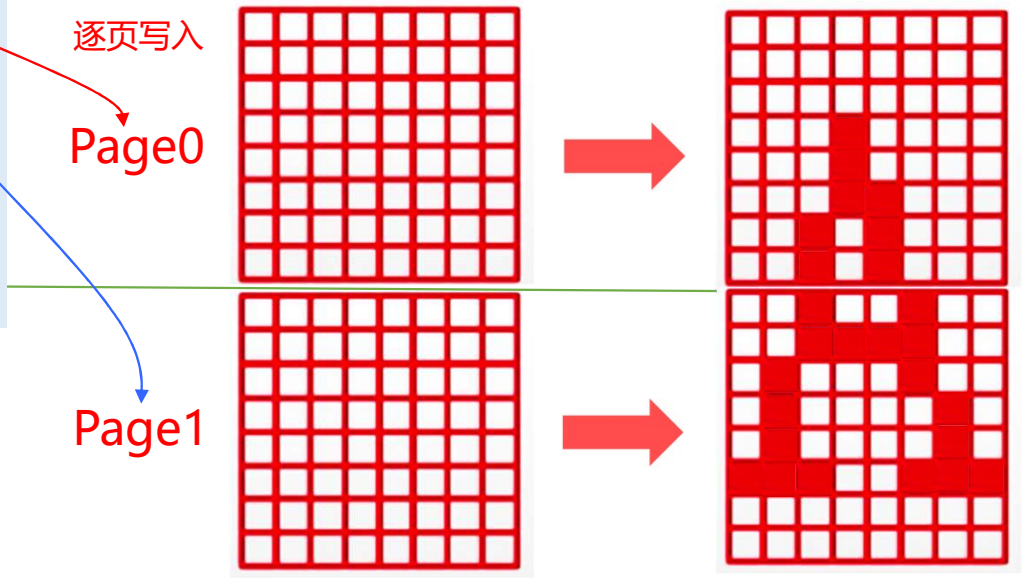
每8行一页
共8页

```
//字模数组：字母A~Z
uint8_t  Alphabet8X8[ ] = {
0x00, 0x80, 0xC0,0xBD, 0x23, 0xBC, 0xC0, 0x80,    /*"A"*/
0x00, 0x7C, 0x54, 0x54, 0x54, 0x4C, 0x7C, 0x00,   /*"B"*/
0x00, 0x3C, 0x44, 0x42, 0x42, 0x42, 0x42, 0x00,   /*"C"*/
0x00, 0x7C, 0x44, 0x44, 0x44, 0x44, 0x3C, 0x00,   /*"D"*/
0x00, 0x7C, 0x54, 0x54, 0x54, 0x5C, 0x44, 0x00,   /*"E"*/
......
```

字符显示编程

画出一个8×16的字符?

```
//字模数组：字母A~Z
uint8_t  Alphabet8X16[ ] = {
0x00, 0x00, 0xC0, 0x38, 0xE0, 0x00, 0x00, 0x00, /*"A"*/
0x20, 0x3C, 0x23, 0x02, 0x02, 0x27, 0x38, 0x20,
0x08, 0xF8, 0x88, 0x88, 0x88, 0x70, 0x00, 0x00, /*"B"*/
0x20, 0x3F, 0x20, 0x20, 0x20, 0x11, 0x0E, 0x00,
……
```



程序怎么写?

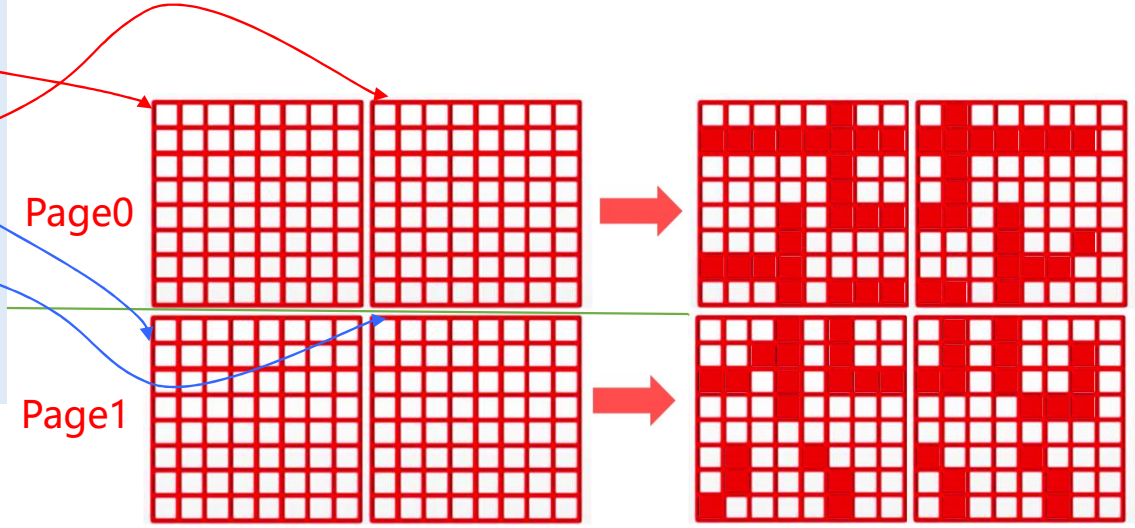
嵌套循环：外循环写“页”，内循环写“列”

```
for (i = 0, i < 2,  i++)        //页控制
{ ……
  for(j = 0, j < 8, j++)
    { OLED_WR_Dat(… ); }   //逐列写
}
```

汉字显示编程

画出一个16×16的汉字?

```
//字模数组：汉字 “燕山大学”
uint8_t  Fontysu16X16[ ] = {
0x42, 0x42, 0x42, 0xF2, 0xF2, 0x9F, 0x9F, 0x92,  /*"燕"*/
0x92, 0x9F, 0x9F, 0xF2, 0xF2, 0x42, 0x62, 0x22,
0x84, 0xE4, 0x66, 0x0F, 0x2F, 0xE7, 0xC7, 0x04,
0x24, 0xE7, 0xC7, 0x07, 0x2F, 0xE8, 0xCE, 0x0E,
……
```



程序怎么写?

```
for (i = 0, i < 2,  i++)        //页控制
{ ……
  for(j = 0, j < 16,  j++)
    { OLED_WR_Dat(… ); }   //逐列写
}
```

字模寻址编程

```
//字模数组：汉字“燕山大学”，规格16×16
uint8_t  Fontysu[ ] = {
0x42, 0x42, 0x42, 0xF2, 0xF2, 0x9F, 0x9F, 0x92, /*"燕"*, 0*/
0x92, 0x9F, 0x9F, 0xF2, 0xF2, 0x42, 0x62, 0x22,
0x84, 0xE4, 0x66, 0x0F, 0x2F, 0xE7, 0xC7, 0x04,
0x24, 0xE7, 0xC7, 0x07, 0x2F, 0xE8, 0xCE, 0x0E,
0x00, 0x00, 0xF0, 0xF0, 0x00, 0x00, 0x00, 0xFF, /*"山", 1*/
0xFF, 0x00, 0x00, 0x00, 0xF0, 0xF0, 0x00, 0x00,
0x00, 0x00, 0x3F, 0x3F, 0x20, 0x20, 0x20, 0x3F,
0x3F, 0x20, 0x20, 0x20, 0x7F, 0x7F, 0x00, 0x00,
0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0xFF, /*"大", 2*/
0xFF, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
0x80, 0x80, 0xC0, 0x60, 0x30, 0x1C, 0x0F, 0x03,
0x03, 0x0F, 0x1C, 0x30, 0x60, 0xC0, 0x80, 0x80,
0x40, 0x70, 0x31, 0x97, 0x96, 0x90, 0x91, 0x97, /*"学", 3*/
0x96, 0x90, 0x98, 0x9C, 0x17, 0x53, 0x70, 0x30,
0x04, 0x04, 0x04, 0x04, 0x04, 0x44, 0xC4, 0xFE,
0x7E, 0x07, 0x05, 0x04, 0x04, 0x04, 0x04, 0x04};
```

字模寻址的规律?

//功能：从第x列、第y行开始，显示一个16×16的字符

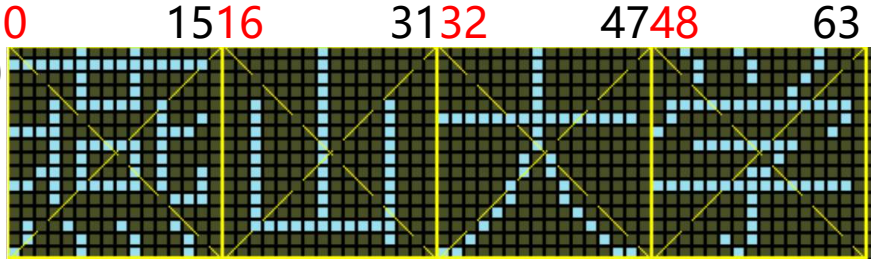
OledShowFont16X16 (uint8_t **x**, uint8_t **y**, const uint8_t ***pixel**); 数组的起始位置(指针)

//第0列、第0行开始显示“燕山大学”

```
OledShowFont16X16(0,  0, Fontysu+0);    /*燕*/
OledShowFont16X16(16, 0, Fontysu+32);   /*山*/
OledShowFont16X16(32, 0, Fontysu+64);   /*大*/
OledShowFont16X16(48, 0, Fontysu+96);   /*学*/
```

=> **for(i = 0, i < 4, i++) {**

OledShowFont16X16(16*i, 0, Fontysu+32*i); }



Questions ?

- **脑洞部分：**
  - 128 × 64 点阵显示多少个bit ？
  - 多少个字节？用什么数据结构？
  - 如何使显示的代码相对独立？
  - 如何提高和确保屏幕刷新的速度？
  - 需要每次都刷新全部屏幕吗？
  - FrameBuffer？图形引擎？

5.3 串行外设接口SPI

- Next……
  - SPI通信简介
  - STM32的SPI
  - SPI常用库函数
  - SPI编程实例 – OLED/LCD显示屏驱动
  - 编程练习E4： SPI接口显示屏编程

✓ 编程练习E4.1&E4.2
SPI接口-显示屏编程
(配套视频E4.1)

思考题 – SPI接口

- 解释SPI通信方式为什么是“同步，串行，全双工，主从”模式
- 解释SPI四根信号线的用途
- SPI通信两种时钟相位(CPHA)的区别，两种时钟极性(CPOL)的区别
- STM32F407IG的**SPI2**接口最大和最小传输波特率是多少？计算原理。
- 如果你买的一块新开发板上有一个SPI器件时，想把它用起来，需要怎么做？（比如怎么知道器件的引脚连接、器件通信所采用的时钟相位和极性、以及该器件允许的最大传输波特率）
- 描述一下规格16×16的“燕”字，在OLED/LCD屏上的显示过程（图在课件39页）
- 若想**竖排**显示“燕山大学”，该如何修改下面这条横排显示的语句？（图在课件40页）

```
for(i = 0, i < 4, i++) { OledShowFont16X16(16 * i,  0, Fontysu+32 * i ); }
```