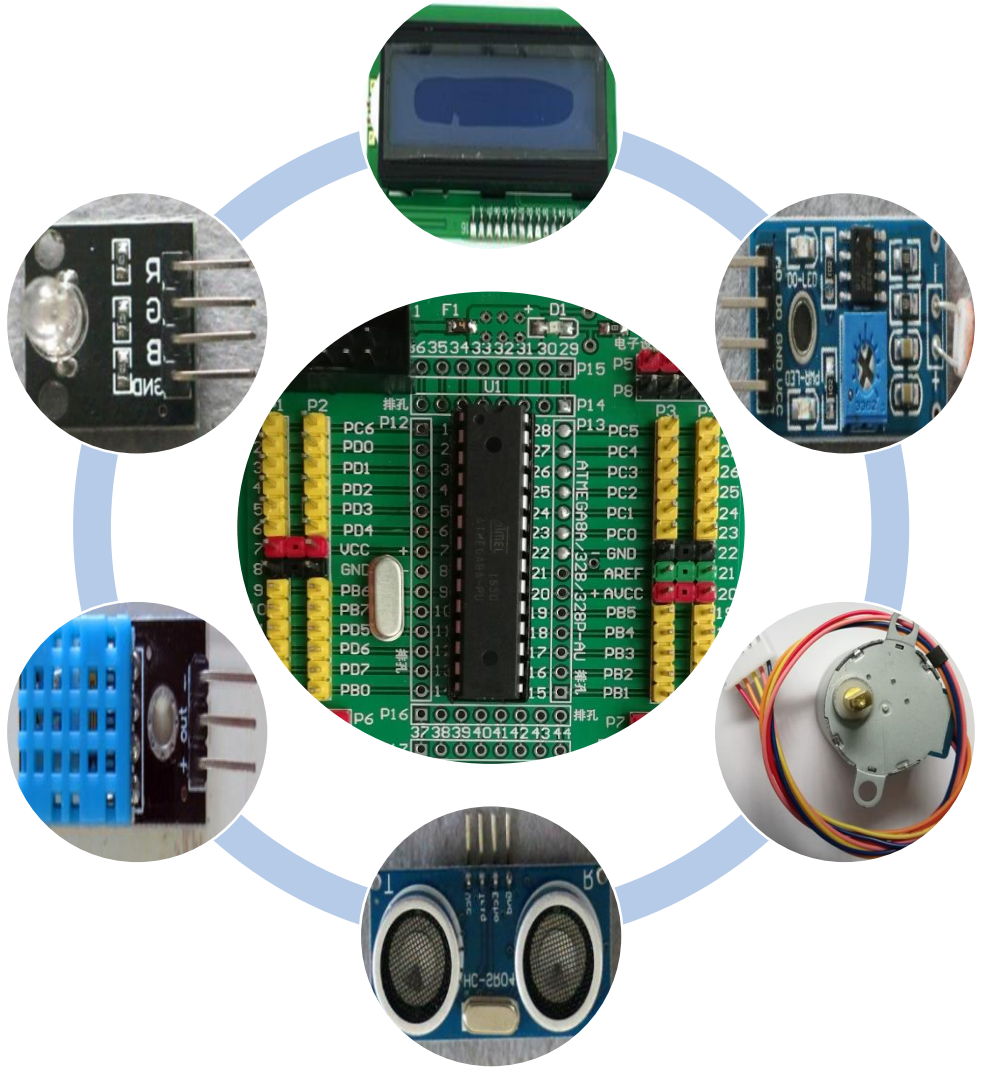
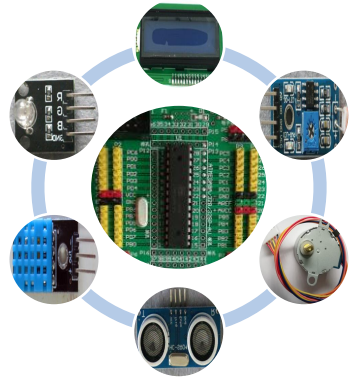
电子设计实践基础

MCU加密位、
熔丝位、
USART接口



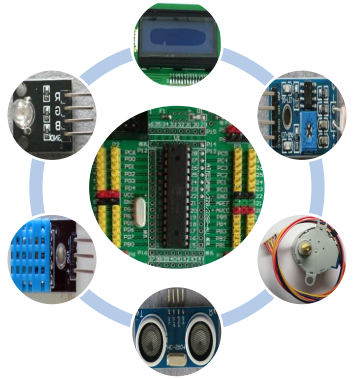
上节课内容回顾

- MCU的“OS”（打破常规）
- 定时器中断与小键盘
- LCD命令与汉字（自定义字符）
- 实时时钟模块
- 蜂鸣器发声

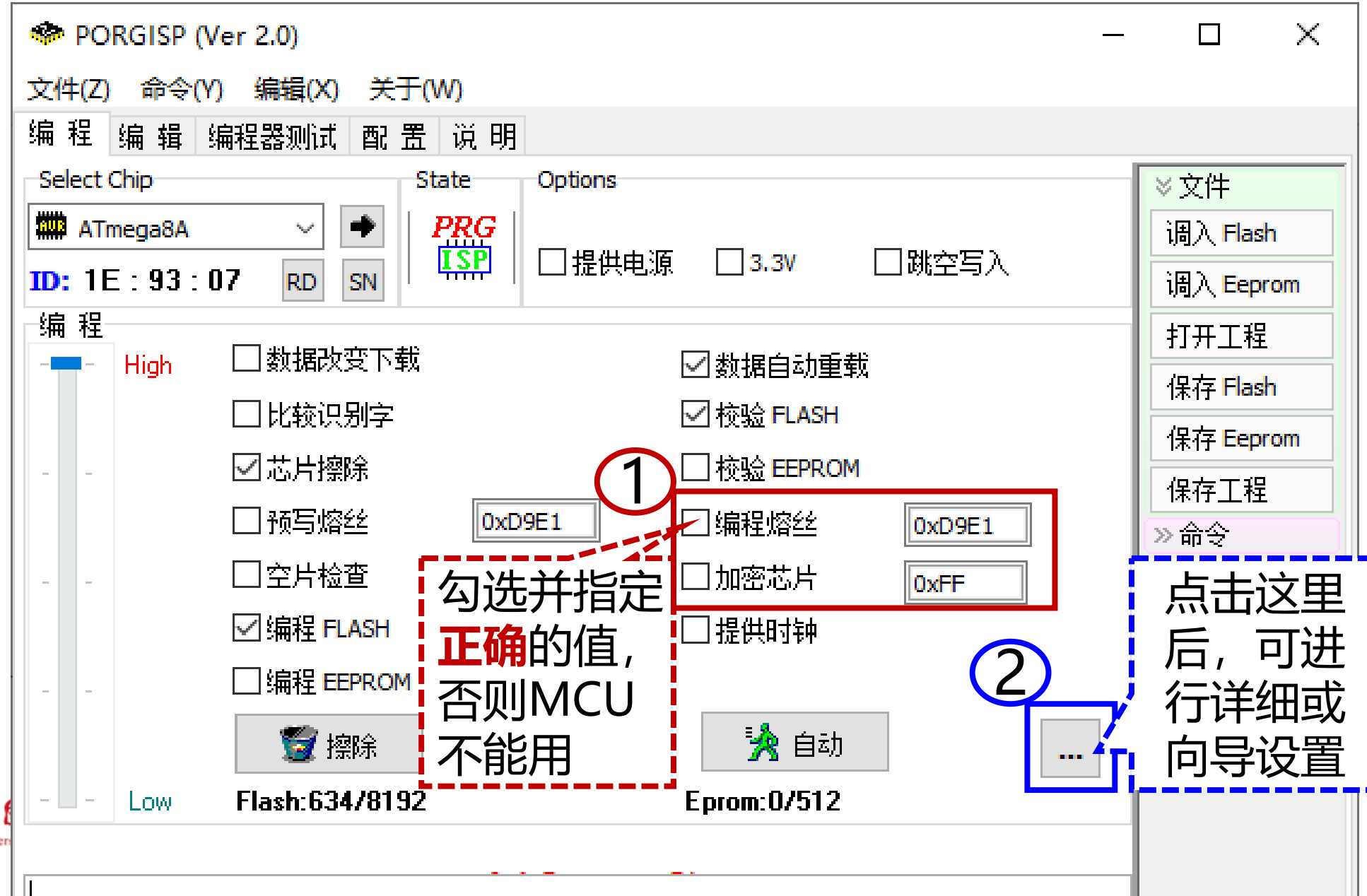


本节课主要内容

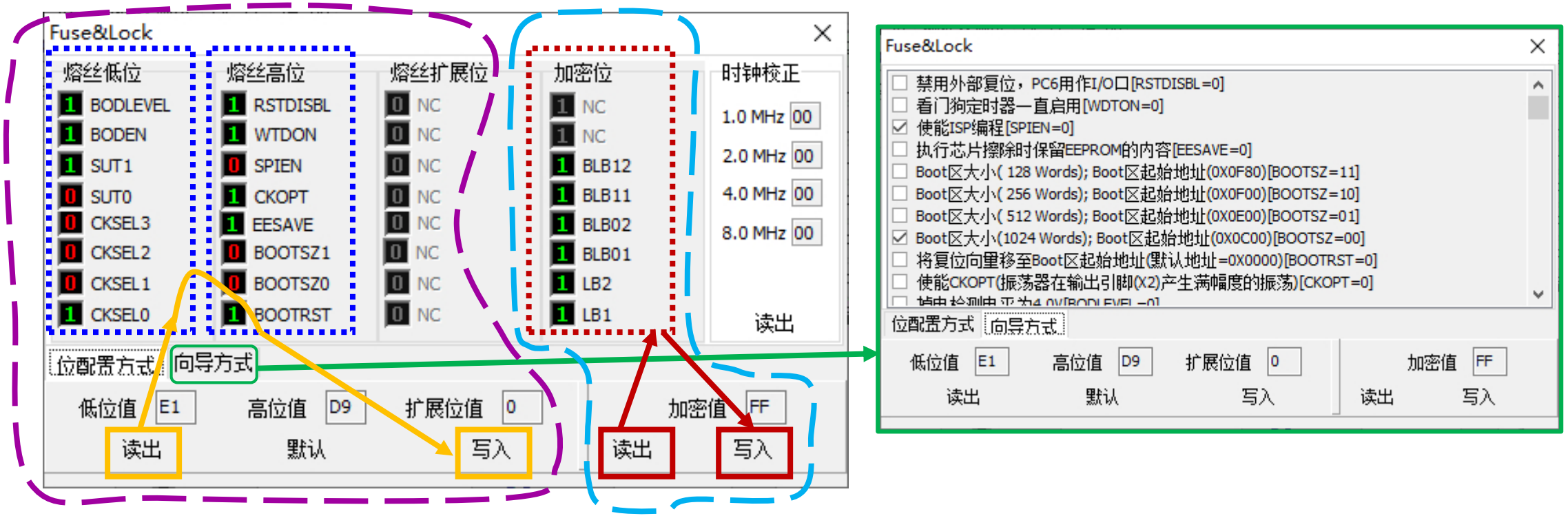
- ATMEGA8A的加密位（保护芯片内的程序）
- ATMEGA8A的熔丝位（设置工作时钟等）
- ATMEGA8A的USART接口（通用串口）



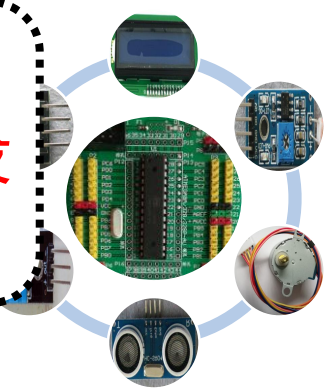
progisp主界面：加密与熔丝的2种设置方法



Progisp里熔丝与加密位的详细与向导设置



先点击“读出”正确读出后，再根据使用需求进行设置（鼠标左键点击相应位设置为0或1），确认无误后再点击“写入”；意外或不确定的设置会导致MCU芯片无法正常使用或须经过特殊处理后才能再用



ATMEGA8A的加密：熔丝加密字节（6位）

- 保护写入芯片里的程序
- 保护存储器里的数据

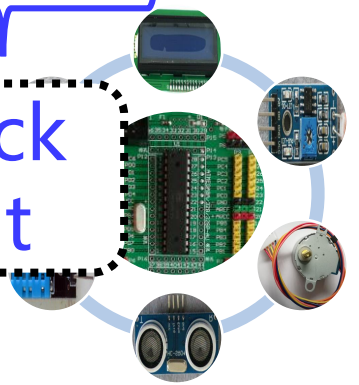
Bit	7	6	5	4	3	2	1	0
加密	-	-	BLB12	BLB11	BLB02	BLB01	LB2	LB1
编程	-	-	是	是	是	是	是	是
默认	1	1	1	1	1	1	1	1

1-未编程，未加密

只能从1编程为0，反之须擦除整个芯片，才能解除

Boot Lock Bit

Lock Bit



存储器上锁位			保护类型
LB模式	LB2	LB1	
1	1	1	存储器上锁特性没有开启
2	1	0	在并行和串行编程模式Flash和EEPROM编程被禁止，熔丝位上锁。
3	0	0	在并行和串行编程模式Flash和EEPROM编程和验证被禁止，熔丝位上锁。
BLB0模式	BLB02	BLB01	**应用区保护**
1	1	1	SPM和LPM读写程序存储器指令对应用区的访问没有限制
2	1	0	SPM不允许写应用区
3	0	0	SPM不允许写应用区，从BootLoader区执行LPM不允许读取应用区，若中断放在BootLoader区，从应用区执行时禁止中断。
4	0	1	从BootLoader区执行LPM不允许从应用区读，如中断向量放在BootLoader区，从应用区执行时禁止中断。
BLB1模式	BLB12	BLB11	**启动加载区保护**
1	1	1	SPM和LPM对BootLoader区的访问没有限制
2	1	0	SPM不允许写BootLoader区
3	0	0	SPM不允许写BootLoader区，从应用区执行LPM时不允许读取BootLoader区，若中断放在应用区，从BootLoader区执行时禁止中断。
4	0	1	从应用区执行LPM时不允许从BootLoader区读，如中断向量放在应用区，从BootLoader区执行时禁止中断。

程序存储器操作指令：SPM-Store Program Memory
LPM-Load Program Memory

ATMEGA8A的熔丝高字节（8位）

Bit	7	6	5	4	3	2	1	0
熔丝H	RSTDISBL	WDTON	SPIEN	CKOPT	EESAVE	BOOTSZ1	BOOTSZ0	BOOTRST
编程	是	是	是	是	是	是	是	是
默认	1	1	0	1	1	0	0	1

1-管脚1为/rst,
0-管脚1为PC6

1-关闭SPI,
0-SPI编程

片擦除时1不
留,0-保留EE

选择复位向
量1-0x000

1-由WDTCR控制,
0-ON

振荡器选项

选择启动区大小

1-未编程，0-编程

擦除整个芯片，不影响熔丝高字节

BOOTSZ1:0	大小(字)	范围	启动复位地址
11	128	0xF80-	0xF80
10	256	0xF00-	0xF00
01	512	0xE00-	0XE00
00	1024	0xC00-	0xC00

ATMEGA8A的熔丝低字节（8位）

Bit	7	6	5	4	3	2	1	0
熔丝L	BODLEVEL	BODEN	SUT1	SUT0	CKSEL3	CKSEL2	CKSEL1	CKSEL0
编程	是	是	是	是	是	是	是	是
默认	1	1	1	0	0	0	0	1

BOD检测触发电平

1-OFF,0-ON

选择启动时间

选择时钟源

BOD-Brown out detector

1-未编程，0-编程

擦除整个芯片，不影响熔丝低字节

时钟源	CKSEL[3:0]
外部晶体/陶瓷谐振器	1111-1010
外部低频晶体	1001
外部RC振荡器	1000-0101
校正内部RC振荡器	0100-0001
外部时钟	0000

时钟源	CKSEL3:0		频率范围 MHz	CK SEL0	SUT 1:0	STPP (CK)	ADR (ms)	推荐用法
外部陶瓷谐振器	CKOPT=1	101x	0.4~0.9	0	00/11	258/1K	4.1	陶瓷,快升
外部晶体谐振器	CKOPT=1	110x	0.9~3.0	0/1	01/00	258/1K	65	陶瓷,慢升
	CKOPT=1	111x	3.0~8.0	0	10	1K	-	陶瓷,BOD
	CKOPT=0	101x	1.0~16.0	1	01	16K	-	晶体,BOD
		110x		1	10	16K	4.1	晶体,快升
		111x		1	11	16K	65	晶体,慢升
外部低频晶体	x	1001	32.768kHz	x	00/01/10	1K/1K/32K		快/慢/稳
外部RC振荡器	x	0101	0.1-0.9	1	00	18	-	BOD开启
	x	0110	0.9-3.0	0	01	18	4.1	快升电源
	x	0111	3.0-8.0	1	10	18	65	慢升电源
	x	1000	8.0-12.0	0	11	6	4.1	快升/BOD
校准内部RC振荡器	x	0001	1.0	1	00	6	-	BOD开启
	x	0010	2.0	0	01	6	4.1	快升电源
	x	0011	4.0	1	10	6	65	慢升电源
	x	0100	8.0	0	11	保留		
外部时钟	x	0000	0.1~16.0	0	与校准内部RC振荡器相同			

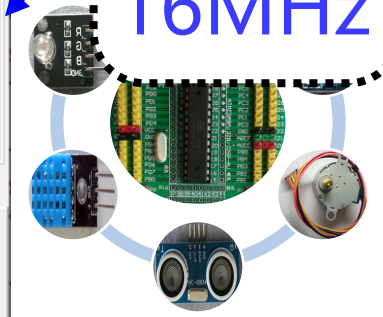
常用时钟源的熔丝位设置（位配置方式）

设置内部校正RC：8MHz

```
DDRC|=(1<<DDRC2)|(1<<DDRC1)|(1<<DDRC0);
while (1)
{  PORTC =(1<<PORTC2);
   _delay_ms(2000);
   PORTC = (1<<PORTC1);
   _delay_ms(2000);
   PORTC = (1<<PORTC0);
   _delay_ms(2000);
}
```

默认内部校正RC：1MHz

设置外部晶体：16MHz



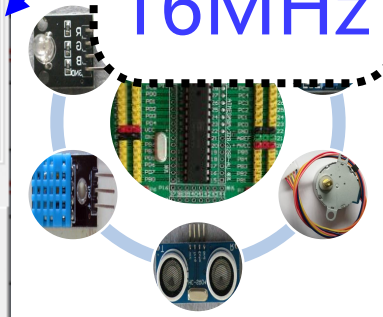
常用时钟源的熔丝位设置（向导方式）

设置内部校正RC：8MHz

```
DDRC|=(1<<DDRC2)|(1<<DDRC1)|(1<<DDRC0);
while (1)
{  PORTC =(1<<PORTC2);
   _delay_ms(2000);
   PORTC = (1<<PORTC1);
   _delay_ms(2000);
   PORTC = (1<<PORTC0);
   _delay_ms(2000);
}
```

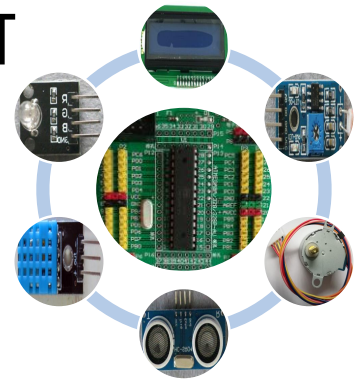
默认内部校正RC：1MHz

设置外部晶体：16MHz

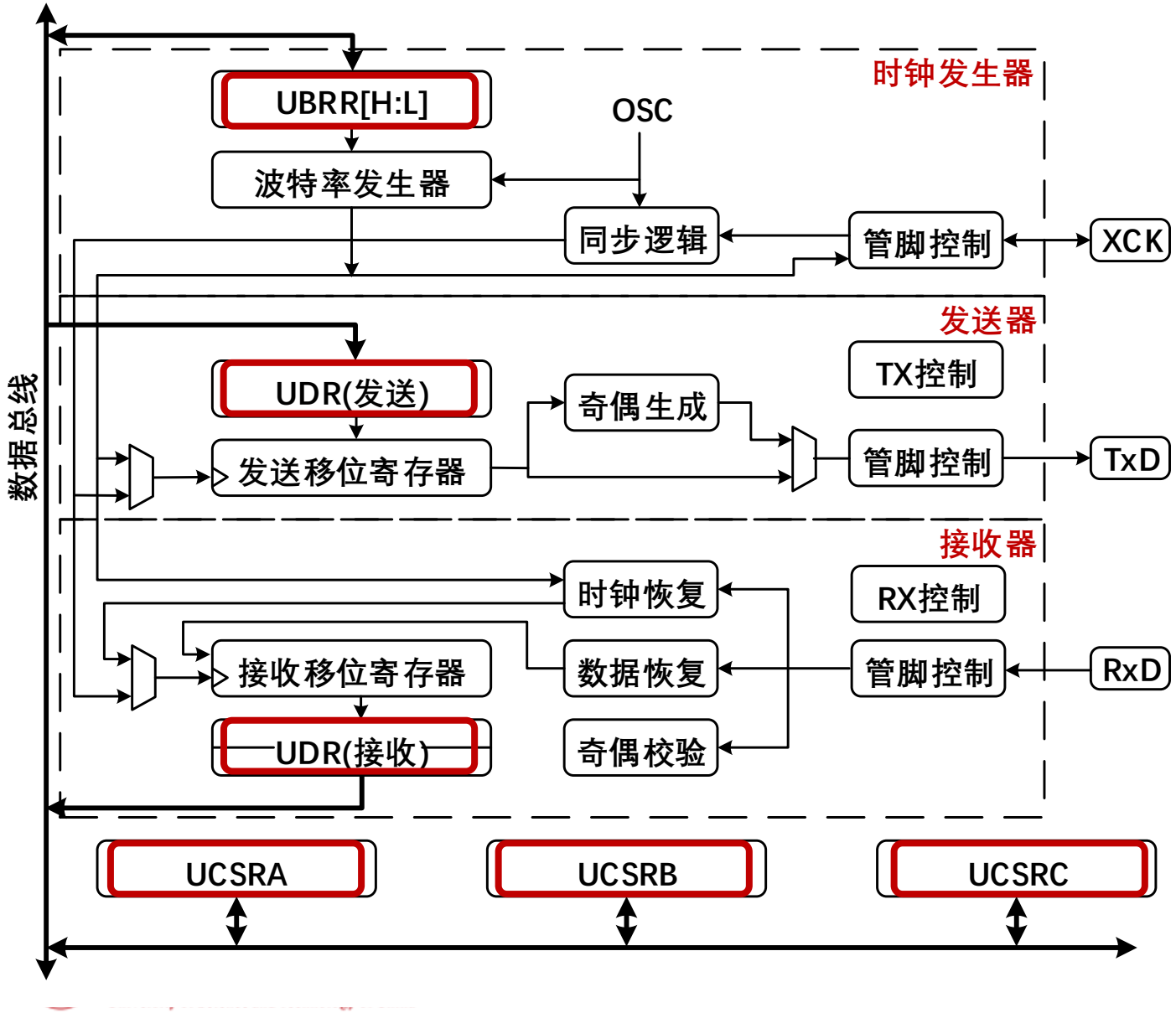


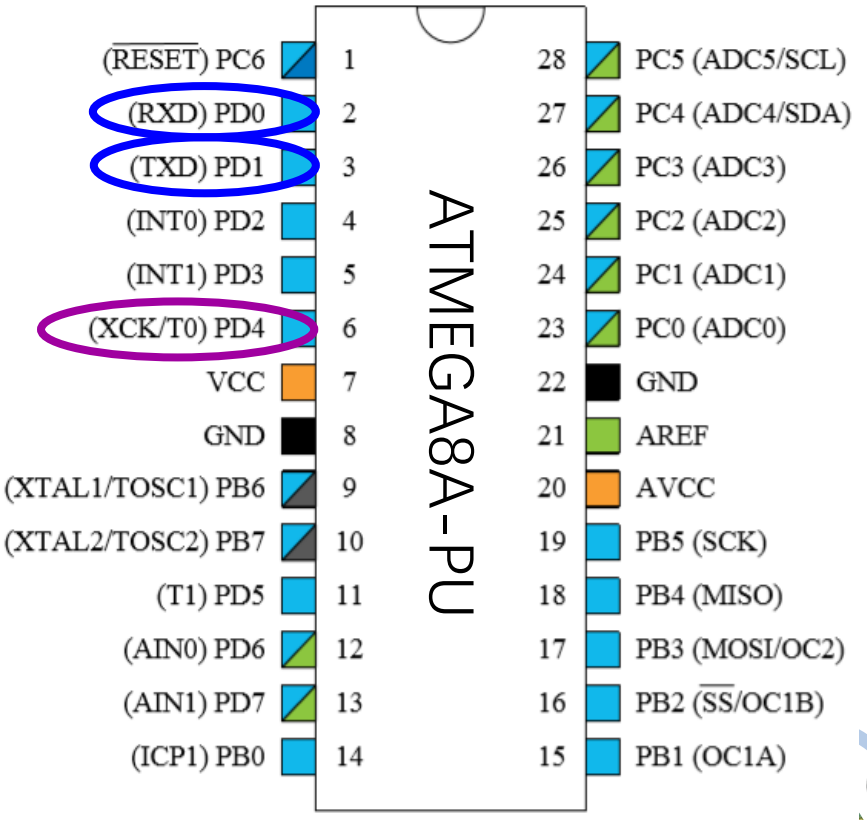
ATMEGA8A的USART接口

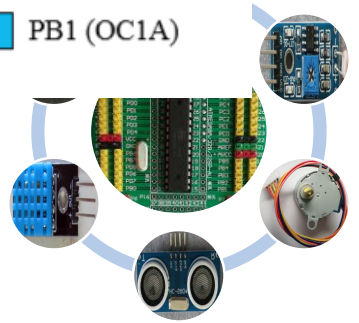
- USART：串口（兼容UART）
  - Universal Synchronous and Asynchronous serial Receiver & Transmitter
  - 同步或异步操作
  - 主/从时钟同步操作
  - 高精度波特率产生器
  - 支持5~9位数据、1或2位停止位的串行帧
  - 奇偶校验、数据过速/帧错误检测、3个独立中断
  - 多处理器模式、倍速异步通信模式



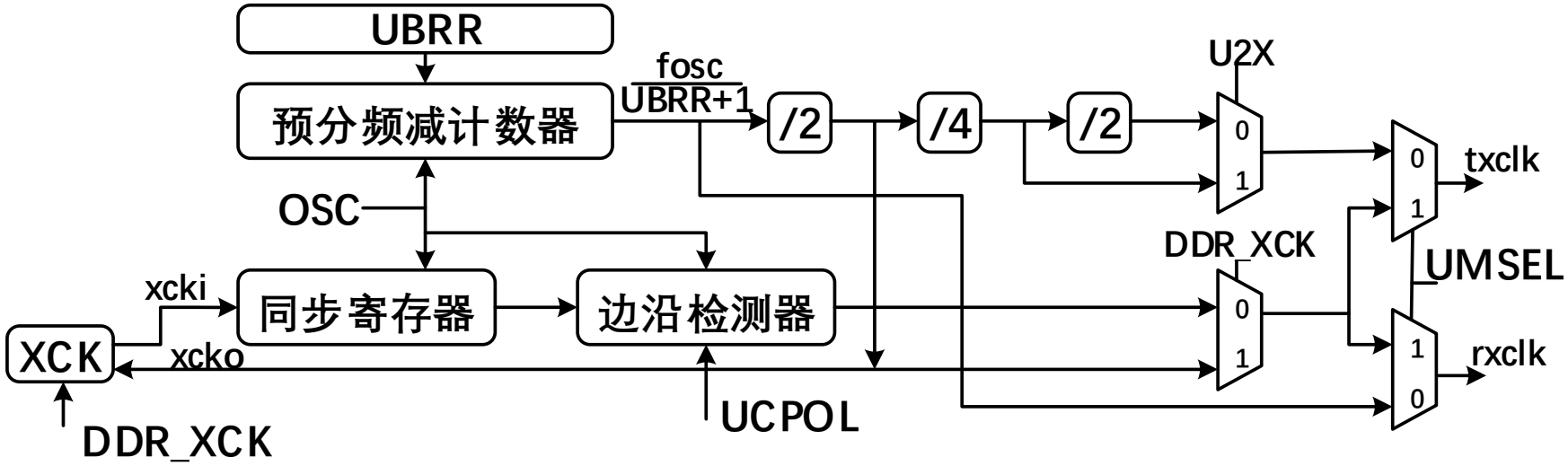
ATMEGA8A的USART结构



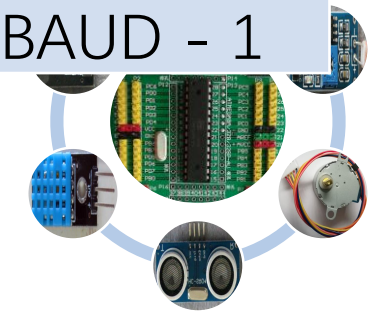




USART的时钟



操作模式	计算波特率	计算UBRR值
异步普通模式（U2X=0）	BAUD=fosc/(UBRR+1)/16	UBRR=fosc/16/BAUD - 1
异步倍速模式（U2X=1）	BAUD=fosc/(UBRR+1)/8	UBRR=fosc/8/BAUD - 1
同步主机模式	BAUD=fosc/(UBRR+1)/2	UBRR=fosc/2/BAUD - 1

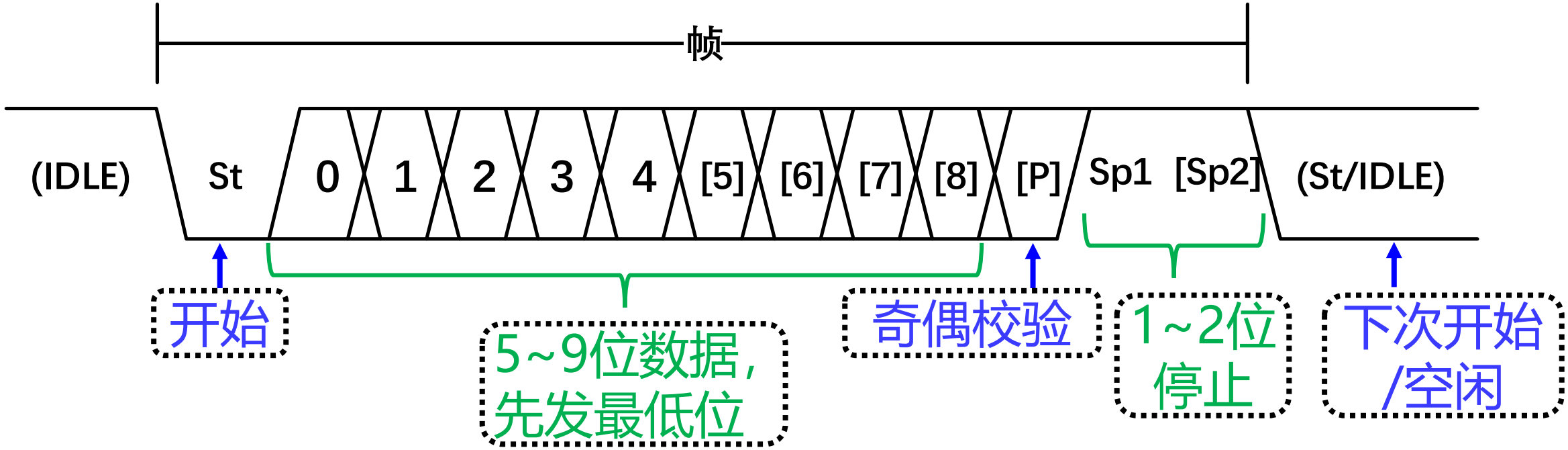


F_CPU

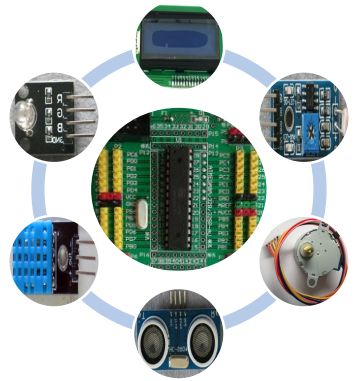
波特率 [bps]	F_{osc} = 1.0000MHz				F_{osc} = 1.8432MHz				F_{osc} = 2.0000MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	-0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max	62.5kbps		125kbps		115.2kbps		230.4kbps		125kbps		250kbps	

波特率 [bps]	F_{osc} = 8.0000MHz				F_{osc} = 11.0592MHz				F_{osc} = 16.0000MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	416	-0.1%	832	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	207	0.2%	416	-0.1%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	103	0.2%	207	0.2%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	68	0.6%	138	-0.1%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	51	0.2%	103	0.2%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	34	-0.8%	68	0.6%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	25	0.2%	51	0.2%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	16	2.1%	34	-0.8%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	12	0.2%	25	0.2%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	8	-3.5%	16	2.1%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	8.5%	8	-3.5%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	0.0%	7	0.0%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	0.0%	3	0.0%
1M	–	–	0	0.0%	–	–	–	–	0	0.0%	1	0.0%
Max	0.5Mbps		1Mbps		691.2kbps		1.3824Mbps		1Mbps		2Mbps	

USART的帧格式



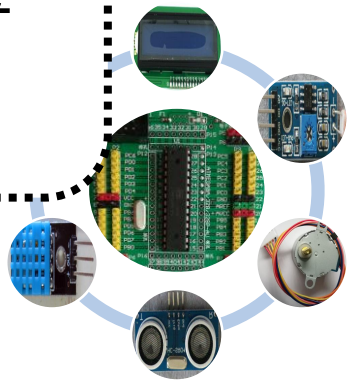
带方括号的项表示可选可不选



USART的寄存器：USART I/O数据寄存器

Bit	7	6	5	4	3	2	1	0
UDR	TXB/RXB[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

USART发送与接收数据缓冲寄存器共享地址。
硬件机制实现：写UDR就是写发送数据缓冲寄存器，读UDR就是读取接收数据缓冲寄存器



USART的寄存器：USART 控制&状态寄存器A

Bit	7	6	5	4	3	2	1	0
UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
Access	R	R/W	R	R	R	R	R/W	R/W
Reset	0	0	1	0	0	0	0	0

接收完成标志位

发送完成标志位

数据寄存器空标志位

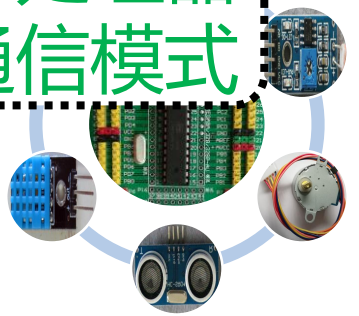
帧错误标志位

数据过速标志位

校验错标志位

传输速度加倍

多处理器通信模式



USART的寄存器：USART 控制&状态寄存器B

Bit	7	6	5	4	3	2	1	0
UCSRB	RXCIE	TXCIE	UDREIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
Access	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Reset	0	0	0	0	0	0	0	0

接收完成中断允许

发送完成中断允许

数据寄存器空中断允许

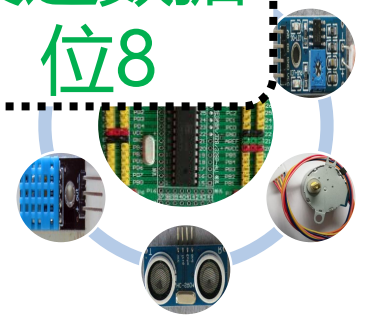
开启接收器

开启发送器

字符位数

接收数据位8

发送数据位8



USART的寄存器：USART 控制&状态寄存器C

Bit	7	6	5	4	3	2	1	0
UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	0	0	0	0	1	1	0

寄存器选择:0-UBRRH,1-UCSRC

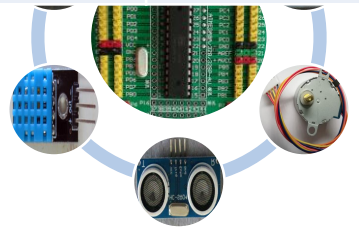
模式选择:0-异步,1-同步

1:0		奇偶校验
0	0	禁用
0	1	保留
1	0	偶校验
1	1	奇校验

停止位: 0-1位,1-2位

2:0			字符位数
0	0	0	5位
0	0	1	6位
0	1	0	7位
0	1	1	8位
1	1	1	9位
其它			保留

UCPOL	TxD输出变化	RxD输入采集
0	XCK上升沿	XCK 下降沿
1	XCK下降沿	XCK上升沿



USART的寄存器：USART波特率寄存器L

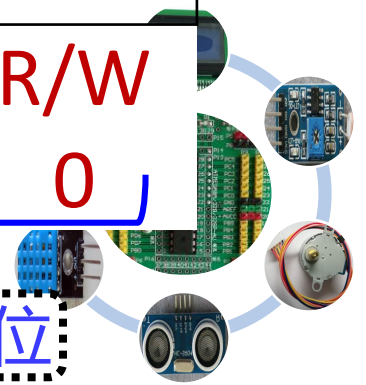
Bit	7	6	5	4	3	2	1	0
UBRRL	UBRR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

波特率寄存器低8位

Bit	7	6	5	4	3	2	1	0
UBRRH	URSEL				UBRR[11:8]			
Access	R/W				R/W	R/W	R/W	R/W
Reset	0				0	0	0	0

寄存器选择:0-UBRRH，1-UCSRC

波特率寄存器高4位



USART的使用

Fosc=1MHz，4800bps，则UBRR=Fosc/16/波特率-1=12，误差0.2%

```
UCSRC=(1<<URSEL);//先清空UCSRC，可不用
```

设置波特率 → 开启收发器 → 设置帧格式 → 收发数据

```
UCSRB=(1<<RXEN)|(1<<TXEN);
```

```
UBRRH=0;
UBRRL=12;
```

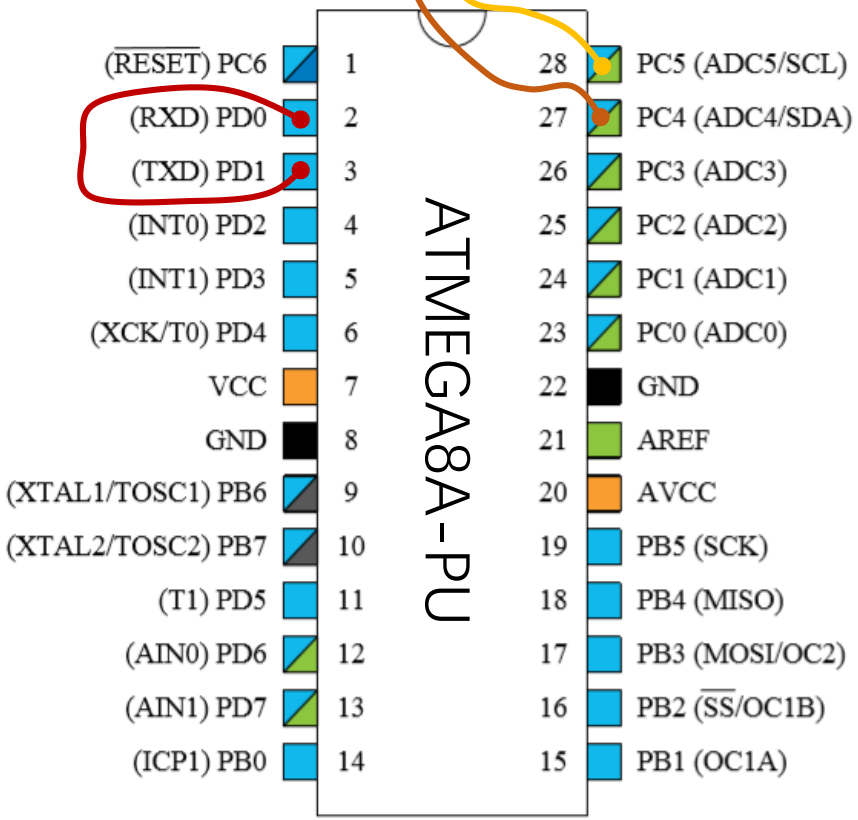
```
UCSRC=(1<<URSEL)|(3<<UCSZ0);
//异步,无校验,8位数据,1位停止位;xck上升沿输出，下降沿采集
```

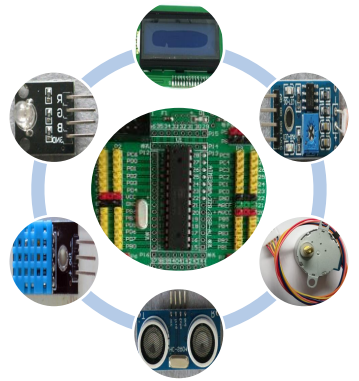
发数据:UDRE=1?UDR=数据;
收数据:RXC=1?读取UDR

UCSRA:U2X、MPCP默认为0，即非倍速，非多处理器

USART示例1

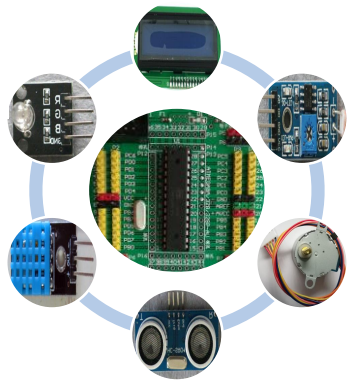
UART自测：通过TXD发送可以在LCD上显示的字符，再通过RXD接收数据并显示在LCD的第2行





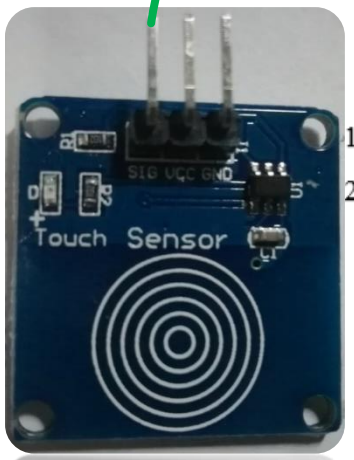
USART示例1编程代码

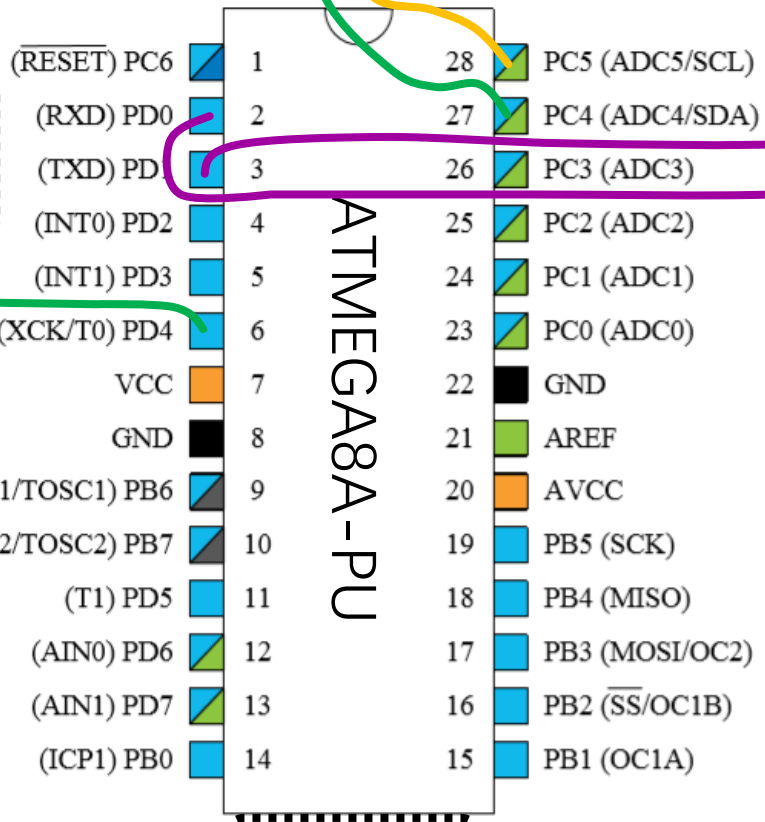
```
#define  F_CPU 1000000UL
#include <avr/io.h>
#include "twi_lcd.h"
#include <util/delay.h>
int main(void)
{ unsigned char i=2,ch=33;//变量
  //UCSRC = 0x80;//清空UCSRC
  UBRRH = 0;
  UBRRL = 12;//1MHz,4800bps,0.2%
  UCSRB = (1<<RXEN)|(1<<TXEN);
  //开启USART接收和发送
  UCSRC =(1<<URSEL)|(3<<UCSZ0);
  //异步,无校验,8位，1停止位,...
  TWI_Init();
  LCD_Init();
  LCD_Write_String(0,0,"S:");
  //LCD第1行提示
  LCD_Write_String(1,0,"R:");
  //LCD第2行提示
  while (1)
  { LCD_Write_Char(0,i,ch);//显示发送的数据
    while(!(UCSRA & (1<<UDRE)));//等待发送
    //可以发送数据?
    UDR = ch;//发送数据
    // delay_us(100);
      delay_ms(200);
    while(!(UCSRA & (1<<RXC)));//等待接收
    LCD_Write_Char(1,i,UDR);//显示接收数据
    if(i==15)i=2;//绕回来显示
    else i++;
    if(ch==127)ch=161;//跳过
    else if(ch==223)ch=33;//绕回来
    else ch++;//下一个可以显示的字符
      delay_ms(200);
   } //while结束
} //main结束
```



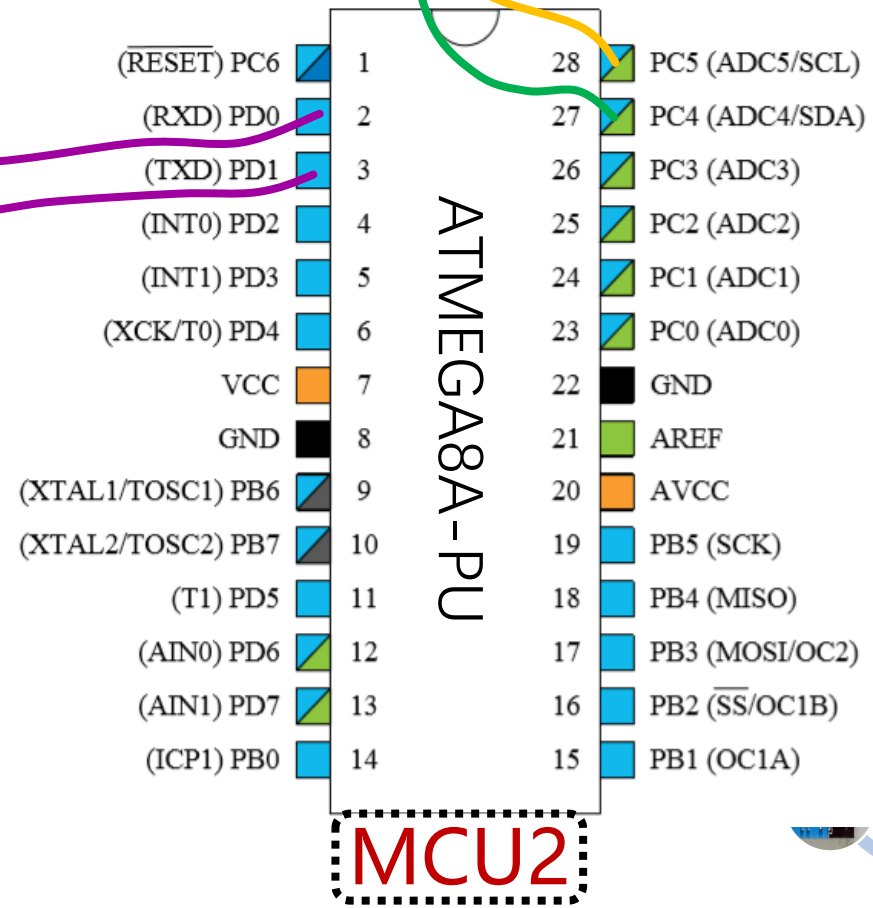
USART示例2

TC0统计
开关次数

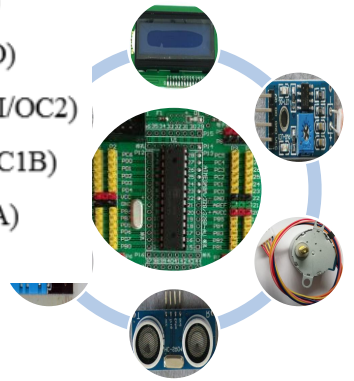




MCU1

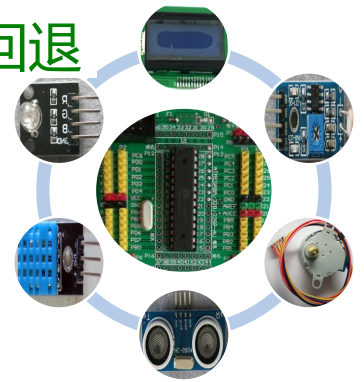


MCU2



USART示例2编程代码：MCU1

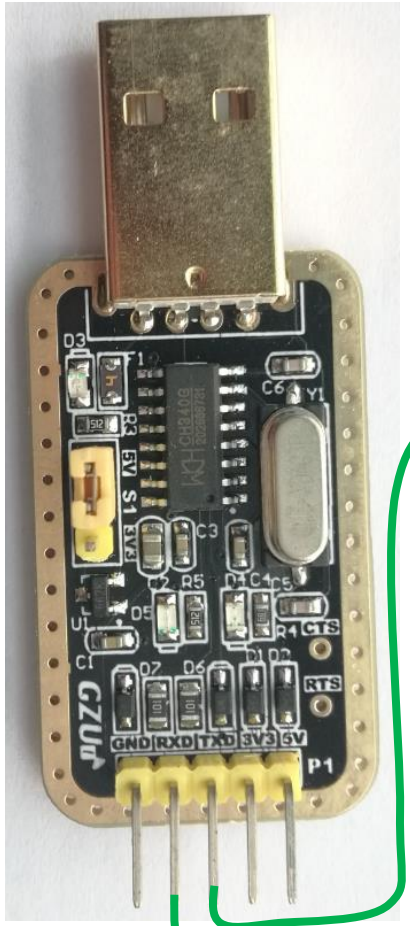
```
#define  F_CPU 1000000UL
#include <avr/io.h>
#include "twi_lcd.h"
#include <util/delay.h>

int main(void)
{ unsigned char i=0,tp_r=0;//临时变量
 TCNT0 = 0;//利用TC0统计开关的次数
 TCCR0 = (1<<CS02)|(1<<CS01)|(1<<CS00);
  //对T0上升沿计数
 UCSRC = 0x80;//清空UCSRC
 UBRRH = 0;//
 UBRRL = 12;//1MHz,4800bps,0.2%
 UCSRB = (1<<RXEN)|(1<<TXEN);
  //开启USART接收和发送
 UCSRC =(1<<URSEL)|(3<<UCSZ0);
  //异步,无校验,8位数据，1位停止位,...
 TWI_Init();
 LCD_Init();
 LCD_Write_String(0,0,"a:Touchpad");
 while (1)
 {  if(tp_r!=TCNT0)//开关次数发生变化
      { tp_r = TCNT0;
        //读取TCNT0，即新的开关次数
       while(!(UCSRA & (1<<UDRE)));
        //可以发送数据?
       UDR = tp_r;//发送数据
      }
      if(UCSRA & (1<<RXC))//有收到数据?
      { LCD_Write_Char(1,i,UDR);//显示
        i++;
      }
      if(i>15)i=0;//逐次显示，并回退
      _delay_ms(1);
  }
 }
```

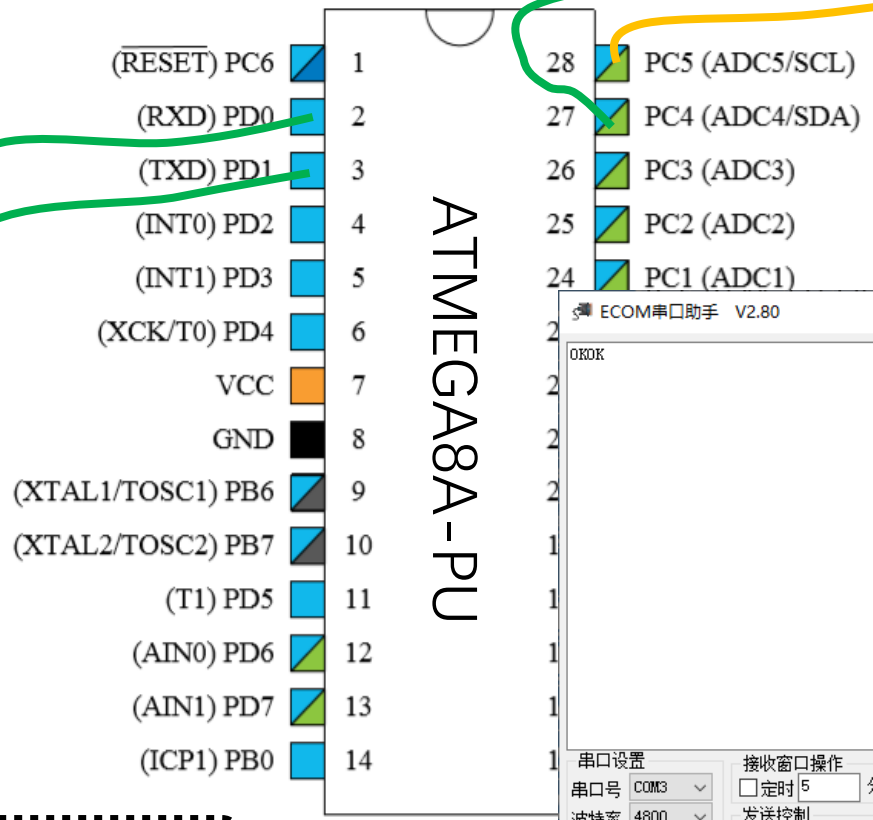


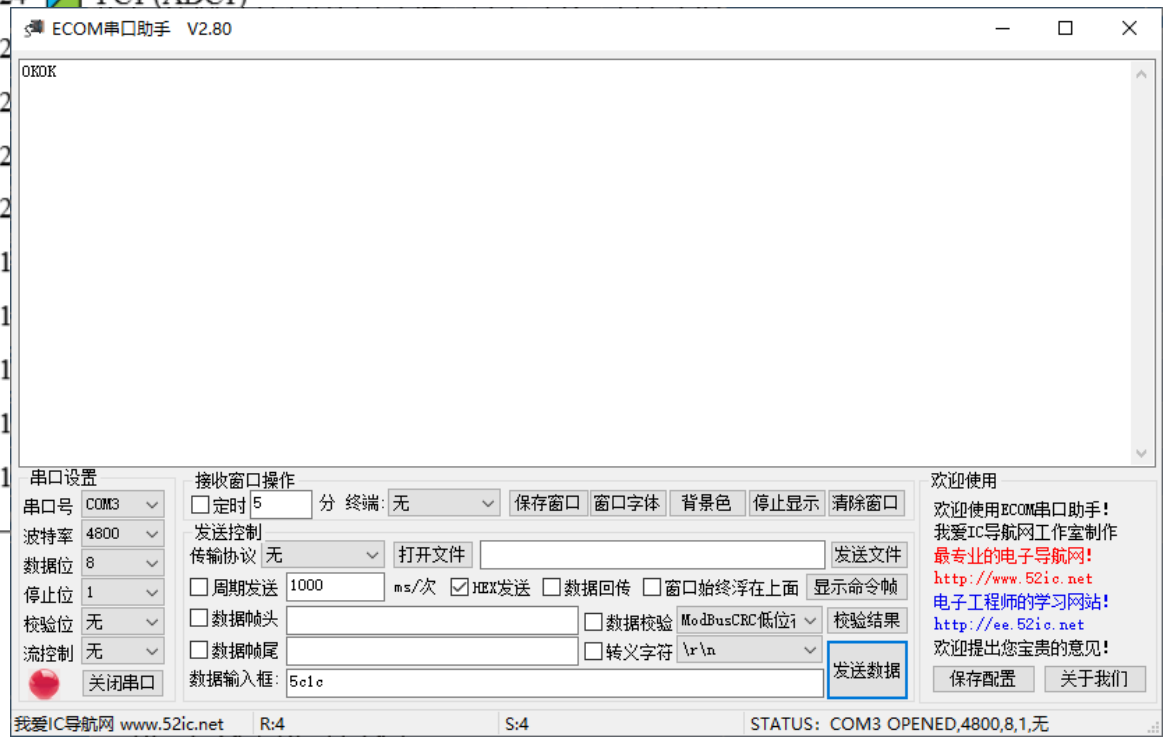
USART示例2编程代码：MCU2

```
#define  F_CPU 1000000UL
#include <avr/io.h>
#include "twi_lcd.h"
#include <util/delay.h>
int main(void)
{ unsigned char i=0,tp_r=0;//临时变量
 TCNT0 = 0;//利用TC0统计开关的次数
 TCCR0 = (1<<CS02)|(1<<CS01)|(1<<CS00);
  //对T0上升沿计数
 UCSRC = 0x80;//清空UCSRC
 UBRRH = 0;
 UBRRL = 12;//1MHz,4800bps,0.2%
 UCSRB = (1<<RXEN)|(1<<TXEN);
  //开启USART接收和发送
 UCSRC =(1<<URSEL)|(3<<UCSZ0);
  //异步,无校验,8位数据, 1位停止位,...
 TWI_Init();
 LCD_Init();
 LCD_Write_String(0,0,"b:Trans");
 while (1)
 { if(UCSRA & (1<<RXC))//有收到数据
   { tp_r = UDR;//读数据
     i=3; //1字节最多3位10进制数
     while(tp_r>0)//转换成字符
     { tmp[i]=tp_r%10+0x30;
       tp_r = tp_r/10;
       while(!(UCSRA & (1<<UDRE)));//等待
       UDR = tmp[i];//发送字符
       i--;
     }
    while(i>0)//不足的位不显示
    { tmp[i--]=0x20; }
    LCD_Write_String(1,0,tmp);
   }
   _delay_ms(1);
  } //while结束
} //main结束
```

USART示例3



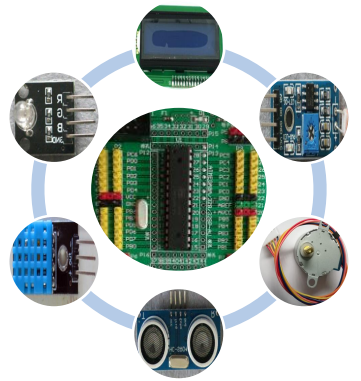




GND RXD TXD 3.3V 5V

USART示例3编程代码

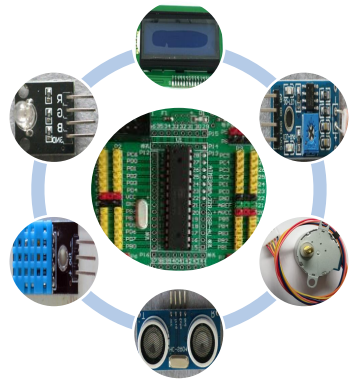
```
#define  F_CPU 1000000UL
#include <avr/io.h>
#include "twi_lcd.h"
#include <util/delay.h>
int main(void)
{  unsigned char cmd_cnt=2,cmd_data[3]={0,0,0};//接收命令和数据
   UCSRC = 0x80;//清空UCSRC
   UBRRL = 0;//
   UBRRL = 12;//1MHz,4800bps,0.2%
   UCSRB = (1<<RXEN)|(1<<TXEN);//开启USART接收和发送
   UCSRC =(1<<URSEL)|(3<<UCSZ0);//异步,无校验,8位数据，1位停止位,...
   TWI_Init();
   LCD_Init();
   while (1)
   {   if(UCSRA & (1<<RXC))//有收到数据
       { cmd_data[cmd_cnt] = UDR;//读数据
         cmd_cnt--;
       }
```



USART示例3编程代码

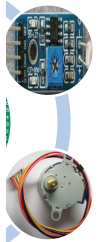
```
if(cmd_cnt<1)//收到两个字节
{   switch(cmd_data[2])
     {//case 0xc0://LCD指令直接操作
      case 0x5c://proteus:\+命令
        LCD_8Bit_Write(cmd_data[1],0);
        break;
      case 0xc1://读忙标志
        break;
       //case 0xc2://往RAM写数据
      case 0x5d://proteus 1+显示字符
        LCD_8Bit_Write(cmd_data[1],1);
        break;
      case 0xc3://从RAM读数据
       break;
}
```

```
      while(!(UCSRA & (1<<UDRE)));//等
      待发送
      UDR = 'O';//发送字符
      while(!(UCSRA & (1<<UDRE)));//等
      待发送
      UDR = 'K';//发送字符
      cmd_cnt=2;//接收下一次命令数据
   }
      delay_ms(1);
  } //while结束
 } //main结束
```

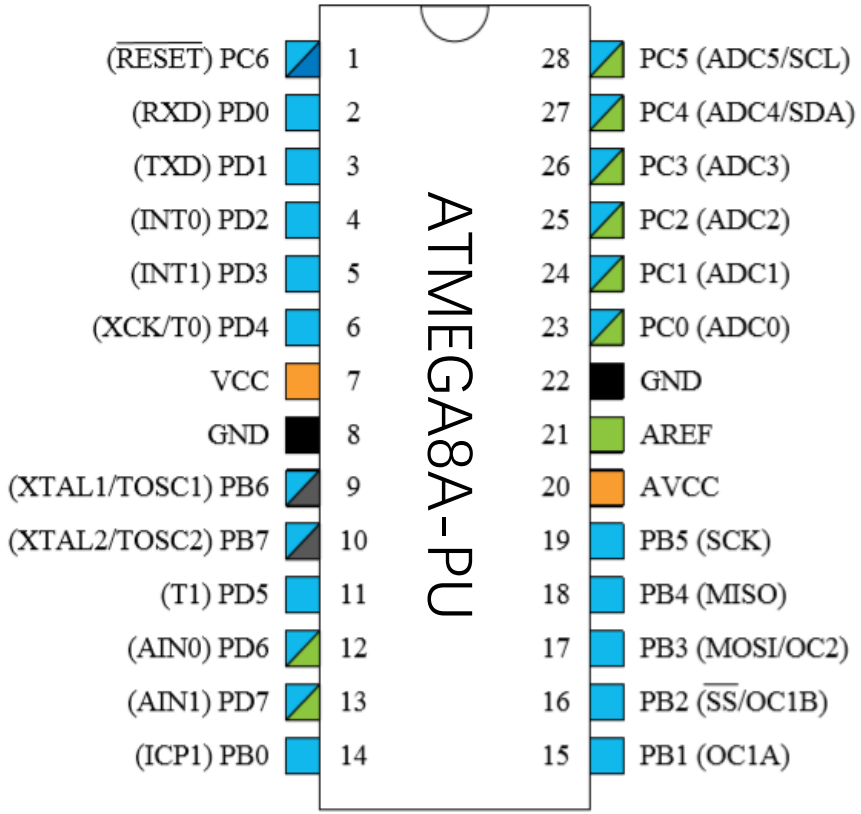


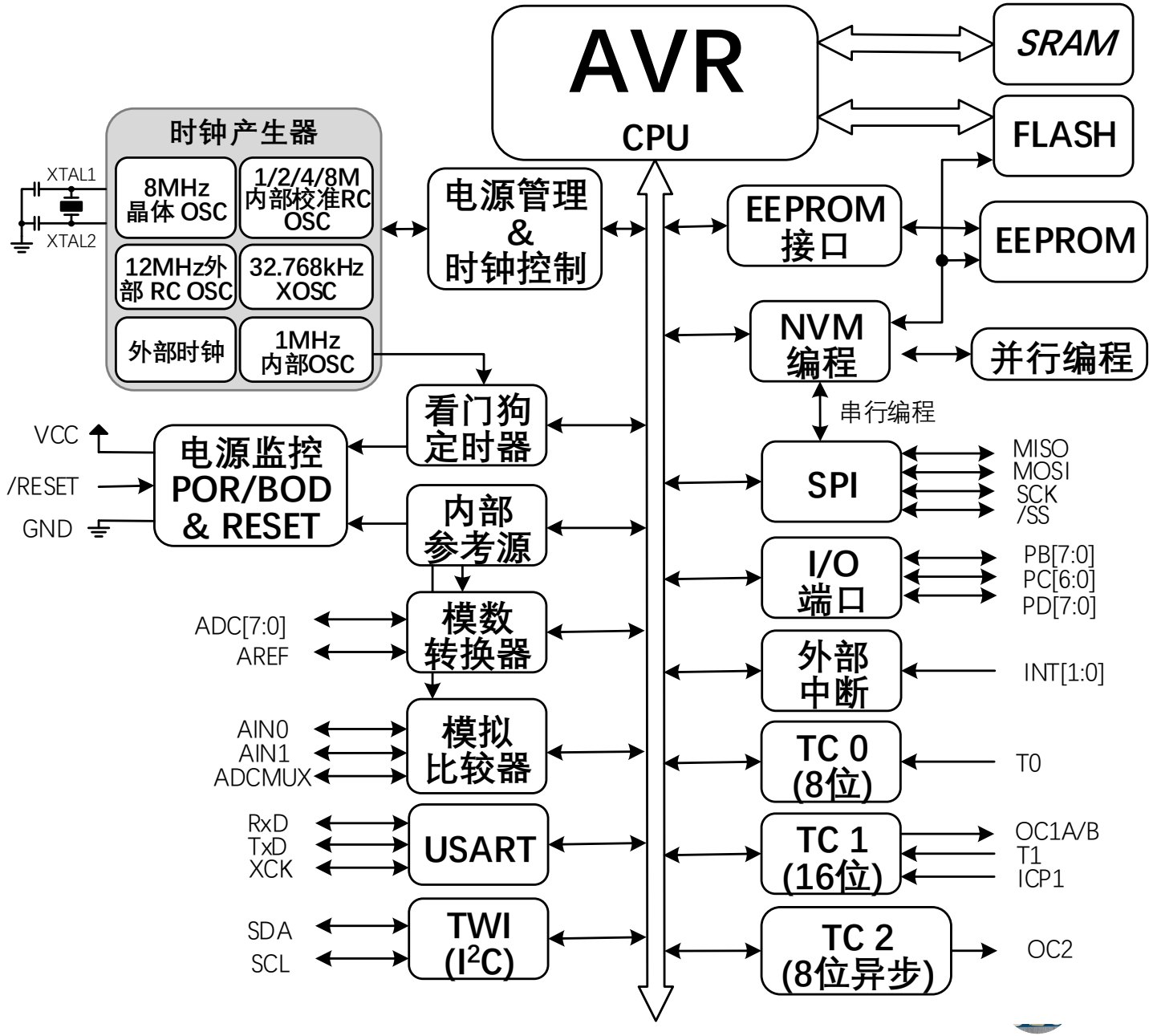
ATMEGA8A拓展USB、USART

- USBasp - USB programmer for Atmel AVR controllers
  - https://www.fischl.de/usbasp/
- V-USB（ARV-USB）
  - V-USB is a firmware-only USB driver for Atmel's AVR microcontrollers
  - https://github.com/obdev/v-usb
  - https://www.obdev.at/products/vusb/
- http://www.recursion.jp/prose/avrcdc/cdc-232.html



ATmega8A





实验内容

1，实验时间：周一晚上、周三下午和晚上、周四和周五晚上

2， 2023年5月26日前完成综合设计、设计报告等全部内容