实验1跑马灯(蜂鸣器)实验

1.函数操作来源：

①stm32f10x\_rcc.h

I.void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);(操作RCC\_APB2ENR寄存器)

②stm32f10x\_gpio.h

I.uint8\_t GPIO\_ReadOutputDataBit(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin);（读取GPIO\_ODR寄存器）

II. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);(操作GPIO\_CRL/CRH寄存器)

III. void GPIO\_SetBits(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin);(操作GPIO\_BSRR寄存器)

IV. void GPIO\_ResetBits(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin);(操作GPIO\_BRR寄存器)

2.实验操作步骤

① void LED\_Init(void)

I.使能端口时钟。

II.端口初始化。

实验2按键输入实验

1.函数操作来源

①stm32f10x\_rcc.h

I. I.void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);(操作RCC\_APB2ENR寄存器)

②stm32f10x\_gpio.h

I.uint8\_t GPIO\_ReadInputDataBit(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin);（读取GPIO\_IDR寄存器）

II. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);(操作GPIO\_CRL/CRH寄存器)

2.实验操作步骤

①void KEY\_Init(void)

I.使能端口时钟。

II.端口初始化。

② u8 KEY\_Scan(u8 mode) //mode=0不支持连按，mode=1支持连按

{

static u8 key\_up = 1; //静态变量key\_up=1

if(mode) key\_up = 1; //mode=1时执行此句

if(key\_up && (KEY0==0||KEY1==0||KEY2==0||WK\_UP==1)) //判断是否有按键按下

{

delay\_ms(10);

key\_up=0;

if(KEY0==0) return 1; //KEY0是否按下

if(KEY1==0) return 2; //KEY1是否按下

if(KEY2==0) return 3; //KEY2是否按下

if(WK\_UP==1) return 4; //WK\_UP是否按下

}else if

(KEY0==1 && KEY1==1 && KEY2==1 && WK\_UP==0) key\_up=1; //没有按键按下

return 0;

}

实验3串口实验

1.函数操作来源

①stm32f10x\_rcc.h

I.void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);(操作RCC\_APB2ENR/APB2RSTR寄存器)

②stm32f10x\_gpio.h

I. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);(操作GPIO\_CRL/CRH寄存器)

③stm32f10x\_usart.h

I. void USART\_Init(USART\_TypeDef\* USARTx, USART\_InitTypeDef\* USART\_InitStruct);(操作USART\_BRR/CR1)

II. ITStatus USART\_GetITStatus(USART\_TypeDef\* USARTx, uint16\_t USART\_IT);(读取USART\_SR寄存器)

III. uint16\_t USART\_ReceiveData(USART\_TypeDef\* USARTx);(读取USART\_DR寄存器)

IV. void USART\_SendData(USART\_TypeDef\* USARTx, uint16\_t Data);(操作USART\_DR寄存器)

V. void USART\_ITConfig(USART\_TypeDef\* USARTx, uint16\_t USART\_IT, FunctionalState NewState);( 操作USART\_CR1寄存器)

VI. void USART\_Cmd(USART\_TypeDef\* USARTx, FunctionalState NewState);（操作USART\_CR1寄存器）

④misc.h

I. void NVIC\_Init(NVIC\_InitTypeDef\* NVIC\_InitStruct);（）

II. void NVIC\_PriorityGroupConfig(uint32\_t NVIC\_PriorityGroup);()

2.实验操作步骤

① void uart\_init(u32 bound)

I.使能串口和端口时钟。(RCC\_APB2ENR)

II.端口初始化。(GPIOx\_CRL/CRH)

III.串口初始化。(USART\_BRR/CR1)

IV.中断分组初始化。

V.开启串口中断。(USART\_CR1)

VI.使能串口 。()

②串口中断服务函数。

void USART1\_IRQHandler(void) //串口1中断服务程序

{

u8 Res;

if(USART\_GetITStatus(USART1, USART\_IT\_RXNE) != RESET) //接收中断(接收到的数据必须是0x0d 0x0a结尾)

{

Res =USART\_ReceiveData(USART1); //读取接收到的数据

if((USART\_RX\_STA&0x8000)==0) //接收未完成

{

if(USART\_RX\_STA&0x4000) //接收到了0x0d

{

if(Res!=0x0a)USART\_RX\_STA=0; //接收错误,重新开始

else USART\_RX\_STA|=0x8000; //接收完成了

}

else //还没收到0X0D

{

if(Res==0x0d)USART\_RX\_STA|=0x4000;

else

{

USART\_RX\_BUF[USART\_RX\_STA&0X3FFF]=Res ;

USART\_RX\_STA++;

if(USART\_RX\_STA>(USART\_REC\_LEN-1))USART\_RX\_STA=0; //接收数据错误,重新开始接收

}

}

}

}

实验4外部中断实验

1.函数操作来源

①stm32f10x\_rcc.h

I.void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);(操作RCC\_APB2ENR寄存器,一定要使能端口复用)

②stm32f10x\_gpio.h

I. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);(操作GPIO\_CRL/CRH寄存器)

II. void GPIO\_EXTILineConfig(uint8\_t GPIO\_PortSource, uint8\_t GPIO\_PinSource);(操作EXTI\_IMR)

③stm32f10x\_exti.h

I. void EXTI\_Init(EXTI\_InitTypeDef\* EXTI\_InitStruct);（操作EXTI\_RTSR/FTSR）

II. ITStatus EXTI\_GetITStatus(uint32\_t EXTI\_Line);(读取EXTI\_SWIER)

III. void EXTI\_ClearITPendingBit(uint32\_t EXTI\_Line);(操作EXTI\_PR)

④stm32f10x.h

I. void EXTI0\_IRQHandler( )

II. void EXTI2\_IRQHandler ( )

III. void EXTI3\_IRQHandler( )

IV. void EXTI4\_IRQHandler( )

2.实验操作步骤

① void EXTIX\_Init(void)

I.使能端口时钟和端口复用时钟。(RCC\_APB2ENR)

II.端口初始化。(GPIO\_CRL/CRH)

III.设置IO口和中断线的映射关系。

IV.初始化线上中断。

V.配置中断分组和使能中断。

②编写服务中断函数。

void EXTI0\_IRQHandler(void)

{

delay\_ms(10);//消抖

if(WK\_UP==1) //WK\_UP按键

{

BEEP=!BEEP;

}

EXTI\_ClearITPendingBit(EXTI\_Line0); //清除LINE0上的中断标志位

}

实验5独立看门狗

1.函数操作来源

①stm32f10x\_iwdg.h

I. void IWDG\_WriteAccessCmd(uint16\_t IWDG\_WriteAccess);(操作IWDG\_KR寄存器)

II. void IWDG\_SetPrescaler(uint8\_t IWDG\_Prescaler);(操作IWDG\_PR寄存器)

III.void IWDG\_SetReload(uint16\_t Reload);（操作IWDG\_RLR寄存器）

IV.void IWDG\_ReloadCounter(void);（操作IWDG\_KR寄存器）

V.void IWDG\_Enable(void);（操作IWDG\_KR寄存器）

2.实验操作步骤

① void IWDG\_Init(u8 prer,u16 rlr) //独立看门狗初始化，分频系数及重装载值

I.取消PR和RLR寄存器写保护（向KR寄存器写入0X5555）

II.设置预分频系数（写入PR寄存器）

III.设置重装载值（写入RLR寄存器）

IV.开始加入重装载值（向KR寄存器写入0XAAA）

V.使能看门狗（向KR寄存器写入0XCCCC）

② void IWDG\_Feed(void) //喂狗函数

I.应用程序喂狗（向KR寄存器写入0XAAAA）

3.计算公式

①溢出时间：Tout=((4 x 2^prer) x rlr)/40khz

②最长超时时间=（IWDG\_RLR的最大值）X看门狗的时钟周期

实验6窗口看门狗

1.函数操作来源

①stm32f10x\_rrc.h

I. void RCC\_APB1PeriphClockCmd(uint32\_t RCC\_APB1Periph, FunctionalState NewState);(操作RCC\_APB1ENR寄存器)

②stm32f10x\_wwdg.h

I. void WWDG\_SetPrescaler(uint32\_t WWDG\_Prescaler);(操作WWDG\_CFR寄存器)

II. void WWDG\_SetWindowValue(uint8\_t WindowValue);(操作WWDG\_CFR寄存器)

III. void WWDG\_EnableIT(void);(操作WWDG\_CFR寄存器)

IV. void WWDG\_SetCounter(uint8\_t Counter);(操作WWDG\_CR寄存器)

V. void WWDG\_Enable(uint8\_t Counter);(操作WWDG\_CR寄存器)

VI. void WWDG\_ClearFlag(void);(操作WWDG\_SR寄存器)

③misc.h

I. void NVIC\_Init(NVIC\_InitTypeDef\* NVIC\_InitStruct);（中断初始化）

II. void NVIC\_PriorityGroupConfig(uint32\_t NVIC\_PriorityGroup);(分组)

④ stm32f10x.h

I.WWDG\_IRQHandler(窗口看门狗中断服务函数)

2.实验操作步骤

① void WWDG\_Init(u8 tr,u8 wr,u32 fprer) //窗口看门狗初始化，当前计数值，上窗口值，分频时基

I.使能窗口看门狗时钟（APB1ENR,36MHZ）

II.设置分频系数（WWDG\_CFR）

III.设置上窗口值（WWDG\_CFR）

IV.开启提前唤醒中断及清除标志并分组(WWDG\_CFR/SR)

V.使能看门狗(WWDG\_CR)

② void WWDG\_Set\_Counter(u8 cnt) //喂狗函数

I.喂狗(WWDG\_CR)

③ void WWDG\_IRQHandler(void) //提前中断服务函数

{

// WWDG\_Set\_Counter(WWDG\_CNT); //（如果注释掉这段，则程序复位）

WWDG\_ClearFlag(); //清除提前中断标志

LED1=!LED1; //翻转LED1

}

3.计算公式

①超时时间：Twwdg =（4096 X 2^WDGTB X (T[5:0]+1)）/36mhz

实验7定时器中断

1.函数操作来源

①stm32f10x\_rcc.h

I. void RCC\_APB1PeriphClockCmd(uint32\_t RCC\_APB1Periph, FunctionalState NewState);( 操作RCC\_APB1ENR寄存器)

②stm32f10x\_tim.h

I. void TIM\_TimeBaseInit(TIM\_TypeDef\* TIMx, TIM\_TimeBaseInitTypeDef\* TIM\_TimeBaseInitStruct);(操作TIMx\_ARR/PSC寄存器)

II. void TIM\_ITConfig(TIM\_TypeDef\* TIMx, uint16\_t TIM\_IT, FunctionalState NewState);(操作TIMx\_DIER寄存器)

III. void TIM\_Cmd(TIM\_TypeDef\* TIMx, FunctionalState NewState);(操作TIMx\_CR1寄存器)

IV. FlagStatus TIM\_GetFlagStatus(TIM\_TypeDef\* TIMx, uint16\_t TIM\_FLAG);(读取TIMx\_SR寄存器)

V. void TIM\_ClearFlag(TIM\_TypeDef\* TIMx, uint16\_t TIM\_FLAG);（操作TIMx\_SR寄存器）

③ misc.h

I. void NVIC\_Init(NVIC\_InitTypeDef\* NVIC\_InitStruct);（中断初始化）

II. void NVIC\_PriorityGroupConfig(uint32\_t NVIC\_PriorityGroup);(分组)

④ stm32f10x.h

I.TIM\_IRQHandler(窗口看门狗中断服务函数)

2.实验操作步骤

① void TIM3\_Int\_Init(u16 arr,u16 psc)

I.TIM时钟使能（APB1ENR,72MHZ）

II.设置重装载值和分频系数（arr，psc）

III.开启定时器更新中断（TIMx\_DIER）

IV.TIM中断分组设置及初始化

V.使能TIM工作(TIMx\_CR1)

② void TIM3\_IRQHandler(void) //定时器3更新中断服务函数

{

if(TIM\_GetFlagStatus(TIM3,TIM\_FLAG\_Update)!=RESET) //判断是否有中断

{

TIM\_ClearFlag(TIM3,TIM\_FLAG\_Update); //清除更新中断标志

LED0=!LED0;

}

}

3.计算公式

①Tout（溢出时间）=（ARR+1）(PSC+1)/72mhz

实验8 PWM的输出

1.函数操作来源

①stm32f10x\_rcc.h

I. void RCC\_APB1PeriphClockCmd(uint32\_t RCC\_APB1Periph, FunctionalState NewState);( 操作RCC\_APB1ENR寄存器)

II. void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);(操作RCC\_APB2ENR寄存器)

②stm32f10x\_gpio.h

I. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);（操作GPIO\_CRL/CRH寄存器）

II. void GPIO\_PinRemapConfig(uint32\_t GPIO\_Remap, FunctionalState NewState);（操作AFIO\_MAPR寄存器）

③stm32f10x\_tim.h

I. void TIM\_TimeBaseInit(TIM\_TypeDef\* TIMx, TIM\_TimeBaseInitTypeDef\* TIM\_TimeBaseInitStruct);（操作TIMx\_ARR/PSC寄存器）

II. void TIM\_OC2Init(TIM\_TypeDef\* TIMx, TIM\_OCInitTypeDef\* TIM\_OCInitStruct);(操作TIMx\_CCMR1寄存器)

III. void TIM\_OC2PreloadConfig(TIM\_TypeDef\* TIMx, uint16\_t TIM\_OCPreload);（操作TIMx\_CCMR1寄存器）

IV. void TIM\_Cmd(TIM\_TypeDef\* TIMx, FunctionalState NewState); （操作TIMx\_CR1寄存器）

2.实验操作步骤

① void TIM3\_PWM\_Init(u16 arr,u16 psc)

I.使能端口时钟，复用时钟，TIM时钟（RCC\_APB1ENR、APE2ENR）

II.配置GPIO复用推挽输出(GPIO\_CRL/CRH)

III.将通道映射到端口上(AFIO\_MAPR)

IV.设置重装载值和分频系数(TIMx\_ARR/PSC)

V.设置PWM模式(TIMx\_CCMR1)

VI.预装载使能(TIMx\_CCMR1)

VII.TIM计数器和输出使能(TIMx\_CR1/CCER)

VIII.修改占空比(TIMx\_CCR2)

实验9输入捕获

1.函数操作来源

①stm32f10x\_rcc.h

I. void RCC\_APB1PeriphClockCmd(uint32\_t RCC\_APB1Periph, FunctionalState NewState);(操作RCC\_APB1ENR寄存器)

II. void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);(操作APB2ENR寄存器)

②stm32f10x\_gpio.h

I. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);（操作GPIO\_CRL/CRH寄存器）

③stm32f10x\_tim.h

I. void TIM\_TimeBaseInit(TIM\_TypeDef\* TIMx, TIM\_TimeBaseInitTypeDef\* TIM\_TimeBaseInitStruct);（操作TIMx\_ARR/PSC寄存器）

II. void TIM\_ICInit(TIM\_TypeDef\* TIMx, TIM\_ICInitTypeDef\* TIM\_ICInitStruct);(操作TIMx\_CCMR1/CCER寄存器)

III. void TIM\_ITConfig(TIM\_TypeDef\* TIMx, uint16\_t TIM\_IT, FunctionalState NewState);(操作TIMx\_DIER寄存器)

V. void TIM\_Cmd(TIM\_TypeDef\* TIMx, FunctionalState NewState);(操作TIMx\_CR1寄存器)

VI. ITStatus TIM\_GetITStatus(TIM\_TypeDef\* TIMx, uint16\_t TIM\_IT);(操作TIMx\_SR寄存器)

VI.uint16\_t TIM\_GetCapture1(TIM\_TypeDef\* TIMx);(操作TIM\_CCR1寄存器)

VIII. void TIM\_OC1PolarityConfig(TIM\_TypeDef\* TIMx, uint16\_t TIM\_OCPolarity);(操作TIMx\_CCER寄存器)

IX. void TIM\_ClearITPendingBit(TIM\_TypeDef\* TIMx, uint16\_t TIM\_IT);(操作TIMx\_SR寄存器)

④ stm32f10x.h

I.TIM\_IRQHandler(窗口看门狗中断服务函数)

2.实验操作步骤

① void TIM5\_Cap\_Init(u16 arr,u16 psc) //输入捕获函数初始化

I.开启TIM5时钟，配置P0为下拉输入

II.设置PSC和ARR

III.设置TIM5的CCMR1(映射关系，分频，滤波)

IV.设置TIM5的CCER，开启输入捕获，并设置位上升沿捕获

V.设置TIM5的DIER，使能捕获和更新中断

VI.设置TIM5的CR1使能定时器

②void TIM5\_IRQHandler(void)

{

if(TIM\_GetITStatus(TIM5,TIM\_IT\_CC1)!=RESET) //捕获1事件发生

{

if(TIM5CH1\_CAPTURE\_STA&0x40) //捕获到高电平标志(也就是已经有了下降沿，下一次准备上升)

{

TIM5CH1\_CAPTURE\_STA|=0x80;

TIM5CH1\_CAPTURE\_VAL=TIM\_GetCapture1(TIM5);

TIM\_OC1PolarityConfig(TIM5,TIM\_ICPolarity\_Rising); //设置下一次上升沿捕获

}else //没有捕获到高电平标志，意在这是第一次准备上升

{

TIM5CH1\_CAPTURE\_STA=0;

TIM5CH1\_CAPTURE\_VAL=0;

TIM\_SetCounter(TIM5,0);

TIM5CH1\_CAPTURE\_STA|=0x40; //表示这时有了上升沿

TIM\_OC1PolarityConfig(TIM5,TIM\_ICPolarity\_Falling); //设置下一次下降沿捕获

}

}

if((TIM5CH1\_CAPTURE\_STA&0x80)==0) //还未成功捕获

{

if(TIM\_GetITStatus(TIM5,TIM\_IT\_Update)!=RESET) //定时器更新发生

{

if(TIM5CH1\_CAPTURE\_STA&0x40) //捕获到上升沿

{

if((TIM5CH1\_CAPTURE\_STA&0x3f)==0x3f) //溢出时间超出bit5~bit0

{

TIM5CH1\_CAPTURE\_STA|=0x80; //此时就让它捕获完成一次

TIM5CH1\_CAPTURE\_VAL=0xffff;

}else TIM5CH1\_CAPTURE\_STA++; //没有超出时间就计入bit5~bit0的定时器溢出次数

}

}

}

TIM\_ClearITPendingBit(TIM5,TIM\_IT\_Update|TIM\_IT\_CC1);

}

实验10电容触摸按键

1.函数操作来源

①stm32f10x\_rcc.h

I.void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);(操作RCC\_APB2ENR寄存器)

II. void RCC\_APB1PeriphClockCmd(uint32\_t RCC\_APB1Periph, FunctionalState NewState);（操作RCC\_APB1ENR寄存器）

②stm32f10x\_gpio.h

I. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);（操作GPIO\_CRL/CRH寄存器）

II. void GPIO\_ResetBits(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin);（操作GPIO\_ODR寄存器）

③stm32f10x\_tim.h

I. void TIM\_SetCounter(TIM\_TypeDef\* TIMx, uint16\_t Counter);（操作TIM\_CNT寄存器）

II. void TIM\_ClearITPendingBit(TIM\_TypeDef\* TIMx, uint16\_t TIM\_IT);（操作TIM\_SR寄存器）

III. FlagStatus TIM\_GetFlagStatus(TIM\_TypeDef\* TIMx, uint16\_t TIM\_FLAG);（读取TIM\_SR寄存器）

IV. uint16\_t TIM\_GetCapture2(TIM\_TypeDef\* TIMx);（读取TIM\_CCR2寄存器）

V. void TIM\_TimeBaseInit(TIM\_TypeDef\* TIMx, TIM\_TimeBaseInitTypeDef\* TIM\_TimeBaseInitStruct);（操作TIM\_CCMR1寄存器）

VI. void TIM\_ICInit(TIM\_TypeDef\* TIMx, TIM\_ICInitTypeDef\* TIM\_ICInitStruct);（操作TIM\_CCMR1/CCER寄存器）

VII. void TIM\_Cmd(TIM\_TypeDef\* TIMx, FunctionalState NewState);（操作TIM\_CR1寄存器）

2.实验操作步骤

①void TPAD\_Reset(void); //TPAD复位

I.开启端口A时钟，设置推挽输出0

II.延时5ms后，再设置浮空输入

III.关闭定时器更新中断和捕获中断

IV.定时计数器归零

②u16 TPAD\_Get\_Val(void) //获取电容充电时间

I.复位TPAD

II.等待通道2溢出

III.还没溢出就如果超时就返回计数器的值，如果发生溢出就返回捕获值

③u16 TPAD\_Get\_MaxVal(u8 n) //返回n个电容充电的最大时间（也是一个重要例子）

while(n--)

{

temp = TPAD\_Get\_Val();

if(temp>res) res = temp;

}

return res;

④void TIM5\_CH2\_Cap\_Init(u16 arr,u16 psc) //TIM5通道2的初始化

I.开启端口A时钟和TIM5时钟

II.PA1设置浮空输入

III.TIM5初始化（向上计数，arr，psc）

IV.TIM5通道2初始化（滤波长度8，不分频，上升沿触发）

V.开启定时计数器

⑤u8 TPAD\_Init(u8 psc) //TPAD空载初始化

I.捕获十个数据到buf[i]

for(i=0;i<10;i++)

{

buf[i]=TPAD\_Get\_Val(); //捕获到的值存入buf数组中

delay\_ms(10);

}

II.捕获10个数据升序排列

for(i=0;i<9;i++) //给这十个值进行升序排列

{

for(j=i+1;j<10;j++)

{

if(buf[i]>buf[j])

{

temp = buf[i];

buf[i]=buf[j];

buf[j]=temp;

}

}

}

III.保留中间6个数据求平均值

for(i=2;i<8;i++) temp+=buf[i]; //对中间的6个数据进行累加

tpad\_default\_val = temp/6; //此时已经得出这6个数据的平均值

IV.如果tpad\_default\_val >TPAD\_ARR\_MAX\_VAL/2 return 1，反之return 0

⑥ u8 TPAD\_Scan(u8 mode) //mode=0不支持连续触发，mode=1则支持

I. static u8 keyen = 0;

u8 sample =3; //默认采样次数为3次

II.if(mode)

{

sample = 6; //支持连按的时候，采样为6次

keyen = 0; //支持连按

}

III. if(rval>(tpad\_default\_val+TPAD\_GATE\_VAL)) //充电时间大于初始平均值+门限值

{

if(keyen==0) res = 1; //keyen=0,有效

printf("r:%d\r\n",rval);

keyen = 3; //至少要过3次按键才有效，巧妙的避免了按键连续触发的效果

}

if(keyen) keyen--;

return res;

实验11 OLED显示

1.接口模式

①8080并行接口

I.信号线

CS：OLED 片选信号。

WR：向 OLED 写入数据。

RD：从 OLED 读取数据。

D[7：0]：8 位双向数据线。

RST(RES)：硬复位 OLED。

DC：命令/数据标志（0，读写命令；1，读写数据）。

II.8080并口读写过程

先根据要写入/读取的数据的类型，设置 DC 为高（数据）/低（命令），然后拉低片选，选中 SSD1306，接着我们根据是读数据，还是要写数据置 RD/WR为低，然后：

在 RD 的上升沿，使数据锁存到数据线（D[7：0]）上；

在 WR的上升沿，使数据写入到 SSD1306 里面；

②四线串行（SPI）方式

I. CS：OLED 片选信号。

RST(RES)：硬复位 OLED。

DC：命令/数据标志（0，读写命令；1，读写数据）。

SCLK：串行时钟线。在4 线串行模式下，D0 信号线作为串行时钟线 SCLK。

SDIN：串行数据线。在 4 线串行模式下，D1 信号线作为串行数据线 SDIN。

II. 在 4 线串行模式下，只能往模块写数据而不能读数据。

2.OLED模块显存

①存储器分类

ROM：只读存储器

RAM：随机存储器

SRAM：静态随机存储器

DRAM：动态随机存储器

PROM：可编程只读存储器

EPROM：可擦可编程序只读存储器

EEPROM：可擦可编程只读存储器

②SSD1306 的显存总共为 128\*64bit 大小，SSD1306 将这些显存分为了 8 页。

③常采用在 STM32 的内部建立一个 OLED 的 GRAM（共 128\*8 个字节），在每次修改的时候，只是修改 STM32 上的 GRAM（实际上就是 SRAM），在修改完了之后一次

性把STM32上的GRAM写入到OLED的GRAM。

④SSD1306常用指令

0X81：用于设置对比度

0XAE/0XAF：0XAE 为关闭显示命令；0XAF 为开启显示命令

0X8D：第二个字节的 BIT2 表示电荷泵的开关状态，该位为 1，则开启电荷泵，为 0 则关闭

0XB0~B7：用于设置页地址，其低三位的值对应着 GRAM 的页地址

0X00~0X0F：用于设置显示时的起始列地址低四位

0X10~0X1F：用于设置显示时的起始列地址高四位

⑤实验操作函数

void OLED\_Init(void); //初始化SSD1306

void OLED\_Display\_On(void); //开启OLED显示

void OLED\_Display\_Off(void); //关比OLED显示

void OLED\_WR\_Byte(u8 dat,u8 cmd); //向SSD1306写入一个字节

void OLED\_DrawPoint(u8 x,u8 y,u8 t); //画点（填充或清空）

void OLED\_Fill(u8 x1,u8 y1,u8 x2,u8 y2,u8 dot); //（填充或清空）区域对角坐标

void OLED\_ShowChar(u8 x,u8 y,u8 chr,u8 size,u8 mode); //指定位置显示字符

void OLED\_ShowNum(u8 x,u8 y,u32 num,u8 len,u8 size); //显示数字

void OLED\_ShowString(u8 x,u8 y,const u8 \*p,u8 size); //显示字符串

void OLED\_Refresh\_Gram(void); //更新显存到LCD

void OLED\_Clear(void); //清屏函数

实验12 TFTLCD显示

1.接口模式

CS：TFTLCD 片选信号。

WR：向 TFTLCD 写入数据。

RD：从 TFTLCD 读取数据。

D[15：0]：16 位双向数据线。

RST：硬复位 TFTLCD。

RS：命令/数据标志（0，读写命令；1，读写数据）。

2.驱动芯片

ILI9341/ILI9325/RM68042/RM68021/ILI9320/ILI9328/LGDP4531/LGDP4535/SPFD5408

/SSD1289/1505/B505/C505/NT35310/NT35510

3. RGB565 格式存储颜色数据4.



4. ILI9341的重要指令

0XD3：读 ID4 指令，用于读取 LCD 控制器的 ID

0X36：存储访问控制指令，控制 GRAM 指针的增长方向

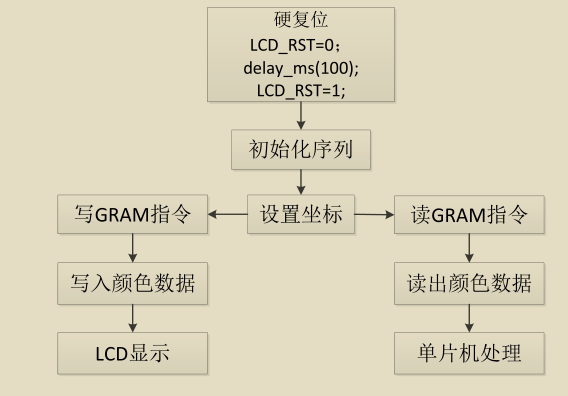
0X2A：列地址设置指令，在从左到右，默认方式下该指令用于设置横坐标（x 坐标）

0X2B：页地址设置指令，在从左到右，默认方式下该指令用于设置纵坐标（y 坐标）

0X2C：写GRAM 指令，便可以往GRAM 里面写入颜色数据了，可连续写

0X2E：是读 GRAM 指令，用于读取 ILI9341 的显存（GRAM）

5.TFTLCD模块使用流程



6.画点和读点

画点过程：设置坐标，写 GRAM 指令，写入颜色数据

读点过程：设置坐标，读 GRAM 指令，读取颜色数据

7.TFTLCD函数

void LCD\_Init(void); //初始化

void LCD\_DisplayOn(void); //开显示

void LCD\_DisplayOff(void); //关显示

void LCD\_Clear(u16 Color); //清屏

void LCD\_SetCursor(u16 Xpos, u16 Ypos); //设置光标

void LCD\_DrawPoint(u16 x,u16 y); //画点

void LCD\_Fast\_DrawPoint(u16 x,u16 y,u16 color); //快速画点

u16 LCD\_ReadPoint(u16 x,u16 y); //读点

void LCD\_Draw\_Circle(u16 x0,u16 y0,u8 r); //画圆

void LCD\_DrawLine(u16 x1, u16 y1, u16 x2, u16 y2); //画线

void LCD\_DrawRectangle(u16 x1, u16 y1, u16 x2, u16 y2); //画矩形

void LCD\_Fill(u16 sx,u16 sy,u16 ex,u16 ey,u16 color); //填充单色

void LCD\_Color\_Fill(u16 sx,u16 sy,u16 ex,u16 ey,u16 \*color); //填充指定颜色

void LCD\_ShowChar(u16 x,u16 y,u8 num,u8 size,u8 mode); //显示一个字符

void LCD\_ShowNum(u16 x,u16 y,u32 num,u8 len,u8 size); //显示一个数字

void LCD\_ShowxNum(u16 x,u16 y,u32 num,u8 len,u8 size,u8 mode); //显示数字

void LCD\_ShowString(u16 x,u16 y,u16 width,u16 height,u8 size,u8 \*p); //显示一个字符串,12/16字体

void LCD\_WriteReg(u16 LCD\_Reg, u16 LCD\_RegValue);

u16 LCD\_ReadReg(u16 LCD\_Reg);

void LCD\_WriteRAM\_Prepare(void);

void LCD\_WriteRAM(u16 RGB\_Code);

void LCD\_SSD\_BackLightSet(u8 pwm); //SSD1963 背光控制

void LCD\_Scan\_Dir(u8 dir); //设置屏扫描方向

void LCD\_Display\_Dir(u8 dir); //设置屏幕显示方向

void LCD\_Set\_Window(u16 sx,u16 sy,u16 width,u16 height); //设置窗口

实验13 USMART调试组件

1.USMART简介

USMART是正点原子团队为其STM32平台开发类似Linux的shell的调试工具，通过串口发送命令给单片机，然后单片机收到命令之后调用单片机里面对应的相关函数，并执行，同时支持返回结果。

2.USMART应用场景

开发过程中，经常需要修改函数入口参数，查看运行效果的情况下应用非常方便。不用多次下载代码，或者多次用JLINK调试。

3.USART系统命令

①？：帮助获取信息

②help：获取帮助信息

③list：可用的函数列表

④id：可用函数的ID列表

⑤hex：参数16进制显示，后跟空格+数字即执行进制转换

⑥dec：参数10进制显示，后跟空格+数字即执行进制转换

⑦runtime：1，开启函数运行计时，0，关闭函数运行计时

实验14 RTC实时时钟

1.函数操作来源：

①stm32f10x\_rcc.h

I. void RCC\_RTCCLKConfig(uint32\_t RCC\_RTCCLKSource);（操作RCC\_BDCR寄存器）

II.void RCC\_RTCCLKCmd(FunctionalState NewState);（操作RCC\_BDCR寄存器）

III. void RCC\_APB1PeriphClockCmd(uint32\_t RCC\_APB1Periph, FunctionalState NewState);（操作RCC\_APB1ENR寄存器）

IV. void RCC\_LSEConfig(uint8\_t RCC\_LSE);（操作RCC\_BDCR寄存器）

V. FlagStatus RCC\_GetFlagStatus(uint8\_t RCC\_FLAG);（操作RCC\_CR寄存器）

②stm32f10x\_rtc.h

I. void RTC\_ITConfig(uint16\_t RTC\_IT, FunctionalState NewState);（操作RTC\_CRH寄存器）

II.void RTC\_EnterConfigMode(void);（操作RTC\_CRL寄存器）

III.void RTC\_ExitConfigMode(void);（操作RTC\_CRL寄存器）

IV.uint32\_t RTC\_GetCounter(void);（读取RTC\_CNTL寄存器）

V.void RTC\_SetCounter(uint32\_t CounterValue);（操作RTC\_CNTH/CNTL寄存器）

VI.void RTC\_SetPrescaler(uint32\_t PrescalerValue);（操作RTC\_PRLH/PRLL寄存器）

VII.void RTC\_SetAlarm(uint32\_t AlarmValue);（操作RTC\_ALRH/ALRL寄存器）

VIII. void RTC\_WaitForLastTask(void);（操作RTC\_CRL寄存器）

IX.void RTC\_WaitForSynchro(void); （操作RTC\_CRL寄存器）

X.ITStatus RTC\_GetITStatus(uint16\_t RTC\_IT);（读取RTC\_CRL寄存器）

XI.void RTC\_ClearITPendingBit(uint16\_t RTC\_IT); （操作RTC\_CRL寄存器）

③.stm32f10x\_pwr.h

I.void PWR\_BackupAccessCmd(FunctionalState NewState);（操作PWR\_CR寄存器）

④stm32f10x\_bkp.h

I. void BKP\_DeInit(void);（操作RCC\_BDCR寄存器）

II. void BKP\_WriteBackupRegister(uint16\_t BKP\_DR, uint16\_t Data);（操作BKP\_DR1寄存器）

2.RTC函数

①u8 RTC\_Init(void); //初始化RTC,返回0,失败;1,成功;

I.使能PWR和BKP时钟：RCC\_APB1PeriphClockCmd();

II.使能后备寄存器访问：PWR\_BackupAccessCmd();

III.检查晶振

while (RCC\_GetFlagStatus(RCC\_FLAG\_LSERDY) == RESET&&temp<250) //检查指定的RCC标志位设置与否,等待低速晶振就绪

{

temp++;

delay\_ms(10);

}

if(temp>=250)return 1;//初始化时钟失败,晶振有问题

IV.配置RTC时钟源，使能RTC时钟：

RCC\_RTCCLKConfig();

RCC\_RTCCLKCmd();

V.等待最近一次对RTC寄存器的写操作完成

VI.等待RTC寄存器同步

VII.使能RTC秒中断

VIII.等待最近一次对RTC寄存器的写操作完成

IX.允许配置

X.设置RTC预分频的值

XI.退出配置模式

XII.向指定的后备寄存器中写入用户程序数据

②u8 Is\_Leap\_Year(u16 year);//平年,闰年判断

{

if(year%4==0) //必须能被4整除

{

if(year%100==0)

{

if(year%400==0)return 1;//如果以00结尾,还要能被400整除

else return 0;

}else return 1;

}else return 0;

}

③u8 RTC\_Alarm\_Set(u16 syear,u8 smon,u8 sday,u8 hour,u8 min,u8 sec);

④u8 RTC\_Get(void); //更新时间

⑤u8 RTC\_Get\_Week(u16 year,u8 month,u8 day);

⑥u8 RTC\_Set(u16 syear,u8 smon,u8 sday,u8 hour,u8 min,u8 sec);//设置时间

实验15 待机唤醒实验

1.函数操作来源

①stm32f10x\_rcc.h

I. void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);（操作RCC\_APB2ENR寄存器）

II.void RCC\_APB1PeriphClockCmd(uint32\_t RCC\_APB1Periph, FunctionalState NewState);（操作RCC\_APB1ENR寄存器）

III. void RCC\_APB1PeriphResetCmd(uint32\_t RCC\_APB1Periph, FunctionalState NewState);（操作RCC\_APB2RSTR寄存器）

②stm32f10x\_pwr.h

I. void PWR\_WakeUpPinCmd(FunctionalState NewState);（操作PWR\_CSR寄存器）

II. void PWR\_EnterSTANDBYMode(void);（操作PWR\_CR/SCB\_SCR寄存器）

③stm32f10x\_gpio.h

I. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);(操作GPIO\_CRL/CRH寄存器)

II. void GPIO\_EXTILineConfig(uint8\_t GPIO\_PortSource, uint8\_t GPIO\_PinSource);(操作EXTI\_IMR寄存器)

④stm32f10x\_exti.h

I. void EXTI\_Init(EXTI\_InitTypeDef\* EXTI\_InitStruct);（操作EXTI\_RTSR/FTSR寄存器）

II. ITStatus EXTI\_GetITStatus(uint32\_t EXTI\_Line);(读取EXTI\_SWIER寄存器)

III. void EXTI\_ClearITPendingBit(uint32\_t EXTI\_Line);(操作EXTI\_PR寄存器)

⑤misc.h

I. void NVIC\_Init(NVIC\_InitTypeDef\* NVIC\_InitStruct);（中断初始化）

II. void NVIC\_PriorityGroupConfig(uint32\_t NVIC\_PriorityGroup);(分组)

2.待机唤醒配置时钟

①使能电源时钟（因为要配置电源控制寄存器）

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_PWR, ENABLE);

②设置WK\_UP引脚作为唤醒源

PWR\_WakeUpPinCmd(ENABLE);

③设置SLEEPDEEP位，设置PDDS位，执行WFI指令，进入待机

PWR\_EnterSTANDBYMode();

3.待机唤醒函数

①void Sys\_Standby(void); //进入待机模式

②void Sys\_Enter\_Standby(void); //进入系统待机模式

③u8 Check\_WKUP(void); //检测WKUP引脚信号

④void WKUP\_Init(void); //PA0 WKUO 唤醒引脚初始化

4.精彩源码

①u8 Check\_WKUP(void)

{

u8 t=0; //记录按下的时间

LED0=0; //亮灯 DS0

while(1)

{

if(WKUP\_KD)

{

t++; //已经按下了

delay\_ms(30);

if(t>=100) //按下超过3秒钟

{

LED0=0; //点亮DS0

return 1; //按下3s以上了

}

}else

{

LED0=1;

return 0; //按下不足3秒

}

}

}

实验16 ADC实验

1.函数操作来源

①stm32f10x\_rcc.h

I. void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);（操作RCC\_APB2ENR寄存器）

II. void RCC\_ADCCLKConfig(uint32\_t RCC\_PCLK2);（操作RCC\_CFGR寄存器）

②stm32f20x\_gpio.h

I. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);（操作GPIO\_CRL/CRH寄存器）

③stm32f10x\_adc.h

I. void ADC\_DeInit(ADC\_TypeDef\* ADCx);（操作RCC\_APB2RSTR寄存器）

II. void ADC\_Init(ADC\_TypeDef\* ADCx, ADC\_InitTypeDef\* ADC\_InitStruct);（操作ADC\_CR1/CR2寄存器）

III. void ADC\_TempSensorVrefintCmd(FunctionalState NewState);（操作ADC\_CR2寄存器）

IV. void ADC\_ResetCalibration(ADC\_TypeDef\* ADCx);（操作ADC\_CR2寄存器）

V. FlagStatus ADC\_GetResetCalibrationStatus(ADC\_TypeDef\* ADCx);（读取ADC\_CR2寄存器）

VI. void ADC\_StartCalibration(ADC\_TypeDef\* ADCx);（操作ADC\_CR2寄存器）

VII. FlagStatus ADC\_GetCalibrationStatus(ADC\_TypeDef\* ADCx);（读取ADC\_CR2寄存器）

VIII. void ADC\_RegularChannelConfig(ADC\_TypeDef\* ADCx, uint8\_t ADC\_Channel, uint8\_t Rank, uint8\_t ADC\_SampleTime);（操作ADC\_SMPR1/SMPR2寄存器）

IX. void ADC\_SoftwareStartConvCmd(ADC\_TypeDef\* ADCx, FunctionalState NewState);（操作ADC\_CR2寄存器）

X. FlagStatus ADC\_GetFlagStatus(ADC\_TypeDef\* ADCx, uint8\_t ADC\_FLAG);（读取ADC\_SR寄存器）

XI. uint16\_t ADC\_GetConversionValue(ADC\_TypeDef\* ADCx);（读取ADC\_DR寄存器）

2.实验操作步骤

①void Adc\_Init(void)

I.开启端口A时钟，ADC通道时钟，设置模拟输入

II.设置ADC分频因子，最大频率不得超于14Mhz

III.复位ADC1

IV.ADC1初始化，独立单通道单次转换模式，软件触发，右对齐转换数目为1

V.使能指定的ADC1

VI.使能复位校准并等待

VII.使能AD校准并等待

②u16 Get\_Adc(u8 ch)

I.设置指定的ADC通道，一个序列，采样时间

II.开启软件转换启动功能

III.等待转换结束

IV.返回最近一次ADC1规则组的转换结果

③u16 Get\_Adc\_Average(u8 ch,u8 times) //取平均，提高准确度

{

u32 temp\_val=0;

u8 t;

for(t=0;t<times;t++)

{

temp\_val+=Get\_Adc(ch);

delay\_ms(5);

}

return temp\_val/times;

}

实验17 内部温度传感器实验

1.函数操作来源

①stm32f10x\_rcc.h

I. void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);（操作RCC\_APB2ENR寄存器）

II. void RCC\_ADCCLKConfig(uint32\_t RCC\_PCLK2);（操作RCC\_CFGR寄存器）

②stm32f20x\_gpio.h

I. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);（操作GPIO\_CRL/CRH寄存器）

③stm32f10x\_adc.h

I. void ADC\_DeInit(ADC\_TypeDef\* ADCx);（操作RCC\_APB2RSTR寄存器）

II. void ADC\_Init(ADC\_TypeDef\* ADCx, ADC\_InitTypeDef\* ADC\_InitStruct);（操作ADC\_CR1/CR2寄存器）

III. void ADC\_TempSensorVrefintCmd(FunctionalState NewState);（操作ADC\_CR2寄存器）

IV. void ADC\_Cmd(ADC\_TypeDef\* ADCx, FunctionalState NewState);(操作ADC\_CR2寄存器)

V. void ADC\_ResetCalibration(ADC\_TypeDef\* ADCx);（操作ADC\_CR2寄存器）

VI. FlagStatus ADC\_GetResetCalibrationStatus(ADC\_TypeDef\* ADCx);（读取ADC\_CR2寄存器）

VII. void ADC\_StartCalibration(ADC\_TypeDef\* ADCx);（操作ADC\_CR2寄存器）

VIII. FlagStatus ADC\_GetCalibrationStatus(ADC\_TypeDef\* ADCx);（读取ADC\_CR2寄存器）

IX. void ADC\_RegularChannelConfig(ADC\_TypeDef\* ADCx, uint8\_t ADC\_Channel, uint8\_t Rank, uint8\_t ADC\_SampleTime);（操作ADC\_SMPR1/SMPR2寄存器）

X. void ADC\_SoftwareStartConvCmd(ADC\_TypeDef\* ADCx, FunctionalState NewState);（操作ADC\_CR2寄存器）

XI. FlagStatus ADC\_GetFlagStatus(ADC\_TypeDef\* ADCx, uint8\_t ADC\_FLAG);（读取ADC\_SR寄存器）

XII. uint16\_t ADC\_GetConversionValue(ADC\_TypeDef\* ADCx);（读取ADC\_DR寄存器）

2.实验操作步骤

①void T\_Adc\_Init (void)

I.开启端口A时钟，ADC通道时钟，设置模拟输入

II.设置ADC分频因子，最大频率不得超于14Mhz

III.复位ADC1

IV.ADC1初始化，独立单通道单次转换模式，软件触发，右对齐转换数目为1

V.开启内部温度传感器

VI.使能指定的ADC1

VII.使能复位校准并等待

VIII.使能AD校准并等待

②u16 T\_Get\_Adc(u8 ch)

I.设置指定的ADC通道，一个序列，采样时间

II.开启软件转换启动功能

III.等待转换结束

IV.返回最近一次ADC1规则组的转换结果

③u16 Get\_Adc\_Average(u8 ch,u8 times) //取平均，提高准确度

{

u32 temp\_val=0;

u8 t;

for(t=0;t<times;t++)

{

temp\_val+=Get\_Adc(ch);

delay\_ms(5);

}

return temp\_val/times;

}

④short Get\_Temprate(void) //获取内部温度传感器温度值

{

u32 adcx;

short result;

double temperate;

adcx=T\_Get\_Adc\_Average(ADC\_Channel\_16,20); //读取通道16,20次取平均

temperate=(float)adcx\*(3.3/4096); //电压值

temperate=(1.43-temperate)/0.0043+25; //转换为温度值

result=temperate\*=100; //扩大100倍

return result;

}

实验18光敏传感器实验

1.函数操作来源

①stm32f10x\_rcc.h

I. void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);（操作RCC\_APB2ENR寄存器）

II. void RCC\_ADCCLKConfig(uint32\_t RCC\_PCLK2);（操作RCC\_CFGR寄存器）

②stm32f20x\_gpio.h

I. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);（操作GPIO\_CRL/CRH寄存器）

③stm32f10x\_adc.h

I. void ADC\_DeInit(ADC\_TypeDef\* ADCx);（操作RCC\_APB2RSTR寄存器）

II. void ADC\_Init(ADC\_TypeDef\* ADCx, ADC\_InitTypeDef\* ADC\_InitStruct);（操作ADC\_CR1/CR2寄存器）

III. void ADC\_TempSensorVrefintCmd(FunctionalState NewState);（操作ADC\_CR2寄存器）

IV. void ADC\_Cmd(ADC\_TypeDef\* ADCx, FunctionalState NewState);(操作ADC\_CR2寄存器)

V. void ADC\_ResetCalibration(ADC\_TypeDef\* ADCx);（操作ADC\_CR2寄存器）

VI. FlagStatus ADC\_GetResetCalibrationStatus(ADC\_TypeDef\* ADCx);（读取ADC\_CR2寄存器）

VII. void ADC\_StartCalibration(ADC\_TypeDef\* ADCx);（操作ADC\_CR2寄存器）

VIII. FlagStatus ADC\_GetCalibrationStatus(ADC\_TypeDef\* ADCx);（读取ADC\_CR2寄存器）

IX. void ADC\_RegularChannelConfig(ADC\_TypeDef\* ADCx, uint8\_t ADC\_Channel, uint8\_t Rank, uint8\_t ADC\_SampleTime);（操作ADC\_SMPR1/SMPR2寄存器）

X. void ADC\_SoftwareStartConvCmd(ADC\_TypeDef\* ADCx, FunctionalState NewState);（操作ADC\_CR2寄存器）

XI. FlagStatus ADC\_GetFlagStatus(ADC\_TypeDef\* ADCx, uint8\_t ADC\_FLAG);（读取ADC\_SR寄存器）

XII. uint16\_t ADC\_GetConversionValue(ADC\_TypeDef\* ADCx);（读取ADC\_DR寄存器）

2.实验操作步骤

①void Lsens\_Init(void)

I.开启端口A时钟，ADC通道时钟，设置PF8模拟输入

II.设置ADC分频因子，最大频率不得超于14Mhz

III.复位ADC3

IV.ADC8初始化，独立单通道单次转换模式，软件触发，右对齐转换数目为1

V.开启内部温度传感器

VI.使能指定的ADC3

VII.使能复位校准并等待

VIII.使能AD校准并等待

②u16 T\_Get\_Adc3(u8 ch)

I.设置指定的ADC通道，一个序列，采样时间

II.开启软件转换启动功能

III.等待转换结束

IV.返回最近一次ADC1规则组的转换结果

③u8 Lsens\_Get\_Val(void)

{

u32 temp\_val=0;

u8 t;

for(t=0;t<LSENS\_READ\_TIMES;t++)

{

temp\_val+=Get\_Adc3(LSENS\_ADC\_CHX); //读取ADC值

delay\_ms(5);

}

temp\_val/=LSENS\_READ\_TIMES;//得到平均值

if(temp\_val>4000)temp\_val=4000;

return (u8)(100-(temp\_val/40));

}

实验19 DAC数模转换

1.函数操作来源

①stm32f10x\_rcc.h

I. void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);（操作RCC\_APB2ENR寄存器）

II.void RCC\_APB1PeriphClockCmd(uint32\_t RCC\_APB1Periph, FunctionalState NewState);（操作RCC\_APB1ENR寄存器）

②stm32f10x\_gpio.h

I. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);（操作GPIO\_CRL/CRH寄存器）

II. void GPIO\_SetBits(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin);（操作GPIO\_ODR寄存器）

③stm32f10x\_dac.h

I. void DAC\_Init(uint32\_t DAC\_Channel, DAC\_InitTypeDef\* DAC\_InitStruct);（操作DAC\_CR寄存器）

II. void DAC\_Cmd(uint32\_t DAC\_Channel, FunctionalState NewState);（操作DAC\_CR寄存器）

III.void DAC\_SetChannel1Data(uint32\_t DAC\_Align, uint16\_t Data);（操作DAC\_DHR寄存器）

2.实验操作步骤

①void Dac1\_Init(void)

I.开启端口A时钟和DAC时钟

II.设置PA4位模拟输入

III.初始化DAC，设置DAC的工作模式

IV.开启DAC转换通道

V.设置DAC的输出值为0

②void Dac1\_Set\_Vol(u16 vol)

{

float temp=vol;

temp/=1000;

temp=temp\*4096/3.3;

DAC\_SetChannel1Data(DAC\_Align\_12b\_R,temp);//12位右对齐数据格式设置DAC值

}

实验20 PWM DAC的输出

1.函数操作来源

①stm32f10x\_rcc.h

I. void RCC\_APB1PeriphClockCmd(uint32\_t RCC\_APB1Periph, FunctionalState NewState);( 操作RCC\_APB1ENR寄存器)

II. void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState);(操作RCC\_APB2ENR寄存器)

②stm32f10x\_gpio.h

I. void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct);（操作GPIO\_CRL/CRH寄存器）

II. void GPIO\_PinRemapConfig(uint32\_t GPIO\_Remap, FunctionalState NewState);（操作AFIO\_MAPR寄存器）

③stm32f10x\_tim.h

I. void TIM\_TimeBaseInit(TIM\_TypeDef\* TIMx, TIM\_TimeBaseInitTypeDef\* TIM\_TimeBaseInitStruct);（操作TIMx\_ARR/PSC寄存器）

II. void TIM\_OC2Init(TIM\_TypeDef\* TIMx, TIM\_OCInitTypeDef\* TIM\_OCInitStruct);(操作TIMx\_CCMR1寄存器)

III. void TIM\_OC2PreloadConfig(TIM\_TypeDef\* TIMx, uint16\_t TIM\_OCPreload);（操作TIMx\_CCMR1寄存器）

IV.void TIM\_CtrlPWMOutputs(TIM\_TypeDef\* TIMx, FunctionalState NewState);（操作TIM\_BDTR寄存器）

V. void TIM\_Cmd(TIM\_TypeDef\* TIMx, FunctionalState NewState); （操作TIMx\_CR1寄存器）

2.实验操作步骤

① void TIM1\_PWM\_Init(u16 arr,u16 psc)

I.使能端口时钟， TIM时钟（APE2ENR）

II.配置GPIO复用推挽输出(GPIO\_CRL/CRH)

III.设置重装载值和分频系数(TIMx\_ARR/PSC)

IV.设置PWM模式(TIMx\_CCMR1)

V.预装载使能(TIMx\_CCMR1)

VI.MOE 主输出使能,高级定时器必须开启这个（TIM\_BDTR）

VII.TIM计数器和输出使能(TIMx\_CR1/CCER)

VIII.修改占空比(TIMx\_CCR2)

实验21 DMA实验

1.函数操作来源

①stm32f10x\_rcc.h

I. void RCC\_AHBPeriphClockCmd(uint32\_t RCC\_AHBPeriph, FunctionalState NewState);(操作RCC\_AHBENR寄存器)

②stm32f10x\_dma.h

I. void DMA\_DeInit(DMA\_Channel\_TypeDef\* DMAy\_Channelx);()

II. void DMA\_Init(DMA\_Channel\_TypeDef\* DMAy\_Channelx, DMA\_InitTypeDef\* DMA\_InitStruct);(操作DMA\_CPARx/CMARx /CCRx)

III. void DMA\_Cmd(DMA\_Channel\_TypeDef\* DMAy\_Channelx, FunctionalState NewState);(操作DMA\_CCRx寄存器)

IV. void DMA\_SetCurrDataCounter(DMA\_Channel\_TypeDef\* DMAy\_Channelx, uint16\_t DataNumber);（操作DMA\_ CNDTRx寄存器）

③stm32f10\_usart.h

I. void USART\_DMACmd(USART\_TypeDef\* USARTx, uint16\_t USART\_DMAReq, FunctionalState NewState);(操作USART\_CR3寄存器)

2.实验操作步骤

① void MYDMA\_Config(DMA\_Channel\_TypeDef\* DMA\_CHx,u32 cpar,u32 cmar,u16 cndtr)

I.使能DMA传输

II.复位DMA通道值

III.DMA设置初始化（外设地址，内存地址，传输方向，数据大小，数据宽度，工作模式，通道优先级，地址递增，内存到内存传输）

②. void MYDMA\_Enable(DMA\_Channel\_TypeDef\*DMA\_CHx)

I.关闭DMA通道

II.设置通道缓存大小

III.开启DMA通道

3.优秀源码

① 在串口中自动回车发送

const u8 TEXT\_TO\_SEND[]={"ALIENTEK WarShip STM32F1 DMA 串口实验"};

j=sizeof(TEXT\_TO\_SEND);

for(i=0;i<SEND\_BUF\_SIZE;i++) //填充数据到SendBuff

{

if(t>=j) //加入换行符

{

if(mask)

{

SendBuff[i]=0x0a;

t=0;

}else

{

SendBuff[i]=0x0d;

mask++;

}

}else //复制TEXT\_TO\_SEND语句

{

mask=0;

SendBuff[i]=TEXT\_TO\_SEND[t];

t++;

}

}

②显示剩余数据百分比

pro=DMA\_GetCurrDataCounter(DMA1\_Channel4); //得到当前还剩余多少个数据

pro=1-pro/SEND\_BUF\_SIZE; //得到百分比

pro\*=100; //扩大100倍

实验22 IIC通信实验

1.IIC协议

①空闲状态

IIC总线的SDA和SCL两条信号线同时处于高电平时，规定为总线的空闲状态，即

初始化时要设置SDA和SCL为高电平。

②起始信号

当SCL为高期间，SDA由高到低的跳变；启动信号是一种电平跳变时序信号，而不

是一个电平信号。

③停止信号

当SCL为高期间，SDA由低到高的跳变；停止信号也是一种电平跳变时序信号，而 不是一个电平信号。

④应答信号

对于反馈有效应答位ACK的要求是，接收器在第9个时钟脉冲之前的低电平期间将 SDA线拉低，并且确保在该时钟的高电平期间为稳定的低电平。

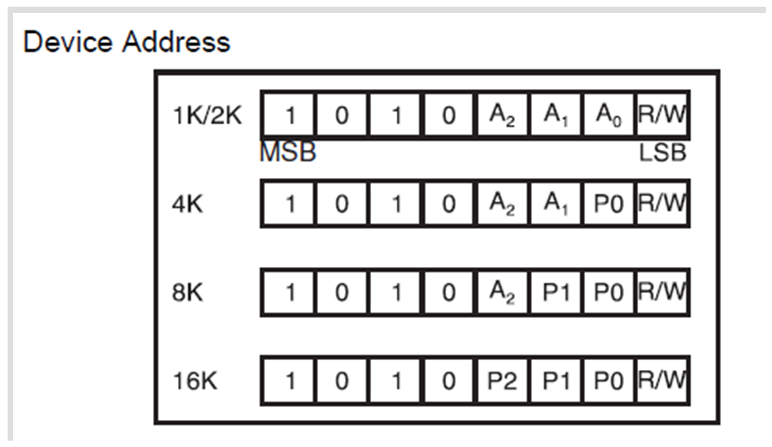
⑤数据有效性

数据在SCL的上升沿到来之前就需准备好。并在在下降沿到来之前必须稳定。

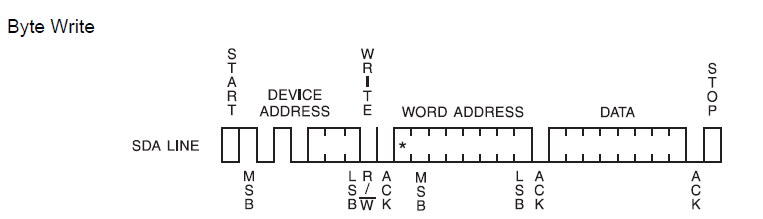
⑥数据的传送

在SCL串行时钟的配合下，在SDA上逐位地串行传送每一位数据。数据位的传输是 边沿触发。

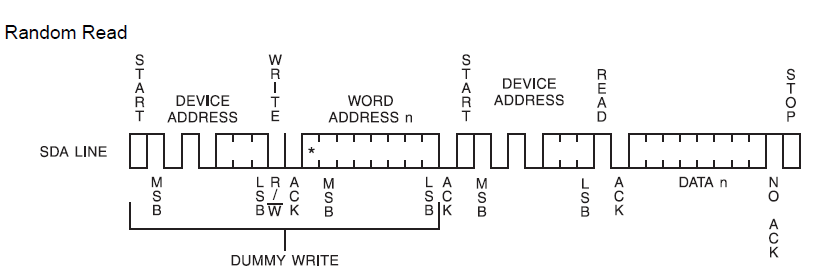
2.EEPROM(24C02)

①操作地址：

②24C02字节写时序：



③24C02字节读时序：



3.实验操作步骤（IIC.h部分）

① void IIC\_Init(void)

I.使能GPIO时钟，并设置为推挽输出

II.设置输出电平为高，使得IIC处于闲置状态

② void IIC\_Start(void)

I. 当SCL为高期间，SDA由高到低的跳变，延时4us左右

II.注意在SCL变化之前，SDA就已经要保持稳定

③ void IIC\_Stop(void)

I. 当SCL为高期间，SDA由低到高的跳变，延时4us左右

II.同样需要注意SCL变化之前，SDA就要保持稳定

④ u8 IIC\_Wait\_Ack(void)

I.SDA设为输入，等待SDA拉低，才表示此时接收应答成功

II.接收应答成功return 0，接收应答失败return 1

⑤ void IIC\_Ack(void)

I.IIC作为从机，发出应答信号

II.SDA始终为0

⑥ void IIC\_NAck(void)

I.IIC作为从机，发出非应答信号

II.SDA始终为1

⑦ void IIC\_Send\_Byte(u8 txd)

I. for(t=0;t<8;t++)

{

//IIC\_SDA=(txd&0x80)>>7;

if((txd&0x80)>>7)

IIC\_SDA=1;

else

IIC\_SDA=0;

}

注意：先发送高字节

⑧ u8 IIC\_Read\_Byte(unsigned char ack)

I.SDA设为输入模式

II. for(i=0;i<8;i++ )

{

IIC\_SCL=0;

delay\_us(2);

IIC\_SCL=1;

receive<<=1;

if(READ\_SDA)receive++;

delay\_us(1);

}

4.实验操作步骤（24CXX.h）

①初始化IIC口void AT24CXX\_Init(void)

void AT24CXX\_Init(void)

{

IIC\_Init();

}

② u8 AT24CXX\_ReadOneByte(u16 ReadAddr)

③ void AT24CXX\_WriteOneByte(u16 WriteAddr,u8 DataToWrite)

④ void AT24CXX\_WriteLenByte(u16 WriteAddr,u32 DataToWrite,u8 Len)

⑤ u32 AT24CXX\_ReadLenByte(u16 ReadAddr,u8 Len)

⑥ u8 AT24CXX\_Check(void)

⑦ void AT24CXX\_Read(u16 ReadAddr,u8 \*pBuffer,u16 NumToRead)

⑧ void AT24CXX\_Write(u16 WriteAddr,u8 \*pBuffer,u16 NumToWrite)