PVD //可编程电压检测器

1. Lib中加入stm32f10x\_pwr.c和bkp.c文件。
2. 去掉conf.h中的两个注释
3. 开启时钟APB1的BKP和PWR
4. PVD的中断时位于EXTI16上的外部中断
5. 具体步骤：缺省值设置，清中断标志位，选择中断线路，模式，触发方式，线路使能，初始化结构体，后备寄存器操作使能，设置电压的阀值，使能PVD
6. 中断的优先级推荐选择抢占最高级
7. PVD中断中保存的数据量和供电的电源上的电容大小有绝对关系。

**SPI //串行外设接口**

1. 加入spi.c文件,去掉conf.h中的两个注释
2. 开启时钟APB2(SPI1)，开启端口复用
3. 配置SPI的MISO,MOSI,Clock,NSS(复用推挽输出，浮空输入，NSS选择软件模式不需要配置，但是模块的NSS还是需要设置的)
4. 用开发板的话，需要失能其他不用的SPI模块，避免出现问题。
5. 结构体数组初始化，模式，极性，相位，校验，等等。

6. SPI的发送和接收，SPI读数据需要发送任意数据才能读。(原因是主设备提供时钟)

7. SPI的NSS引脚在硬件模式下可输入可输出，设置主模式输出模式的时候需要SPI\_SSOutputCmd(SPI1, ENABLE);

8. (16.6.14)

SPI不用中断的发送接收函数测试：

SPI是全双工，读写分别有数据线，共用SCK时钟，写，读是同步进行，不过操作具体芯片的时候，返回值要根据芯片的datasheet操作。所以一般读的时候都需要写数据启动时钟。

9. (16.7.27)

用IO口模拟时序的时候，MOSI和MISO的初始化引脚置高

2016/3/1

1. B码程序MAX485在无输入信号的情况下,输出保持高电平。
2. 定时器输入捕获的时候,捕获上升沿时,捕获引脚设置为下拉输入。
3. 上拉输入：外部无信号输入时，引脚电平保持高电平（比如I2C）
4. 在定时器输入捕获时，捕获引脚设置成浮空输入的话，引脚电平状态是不稳定的，有波动

2016/3/30 I2C

1. STM32的硬件I2C模块不稳定，使用软件模拟时序的方式使用I2C模块，硬件I2C存在Bug.
2. FATFS32的MP3程序在支持中文文件名之后存在目录读取英文文件名失败的现象，此问题未解决(已解决，指针地址没有回送)。

2016/4/5 CAN

1. 在设置一个寄存器之前，相应的位先清空，再设置。
2. CAN总线告一段落，设置CAN总线CAN\_InitTypeDef （设置工作模式，波特率）和 CAN\_FilterInitTypeDef （设置屏蔽滤波器）。
3. 设置发送报文的ID格式，IDE，RTR，数据等参数CanTxMsg 和 CanRxMsg。
4. 相应的中断。屏蔽过滤寄存器关联FIFO，FIFO0时是USB\_LP\_CAN1\_RX0\_IRQHandler。FIFO1时是CAN1\_RX1\_IRQHandler。

2016/4/5 中断

1. Stm32f10x\_it.c中的中断名称 是在startup\_stm32f10x\_hd.s中查找添加。
2. NVIC组管理中的中断通道名称 是在 stm32f10x\_it.h中查找添加。

2016/4/13 Systick 和 TFT触摸屏

1. systick在设置成1us中断时，容易出现问题，具体的等液晶触摸屏功能实现后再测试。

2. TFT触摸屏使用SPI通信时，SPI的频率选择速度太快（8分频）时，高电平的时间达不到200ns，会丢失数据。

3. 等差数列中位数（a0+an）/2

4. 触摸屏使用中断法画点的时候，出现描点非常慢的现象。

解决：因为串口不停的在打印消息占用中断资源，在描点的时候不要打印

5. 液晶取模软件的使用，字模3和 LCD2002完美版，取模的方式不同，字模3是整体取模，但是LCD是每个字单独取模，所以编程上不同。2016.8.4

2016/4/23 定时器延迟函数

1. 定时器更新标志位放在结构体初始化前清空标志位

2016/4/25 Modbus 之CRC校验

1. 串口发送数据时或者检测定时器更新事件时，通过读SR的相应标志位保证发送或者事件的发生。

2. Modbus的CRC16校验在发送时是 低8位在前，高8位在后。

3. 注意运算符的优先级问题。

4. 对于memcpy(目标数组，源数组，个数) 。

5. A? b:c 若A为真，则表达式b；若B为假，则表达式C

2016/4/27 SD response ，数据地址

1. Some SD’s command have response, their saved in SDIO\_RESPx register.

2. When write the address’s data, if address=0, Data width is 4. So when you write in the address 3, you should be 3\*4=12 , Use 12 replace 3.

Example: Address 0—use 0 1 2 3 , Address1—use 4 5 6 7 , Address2—head address= 2\*4=8 9 10 11.

Example: Sector \* block size

上面是应用在字节一一对应

3. normal

u16 a[]={1,2,3};

a 的地址如果是m ,那么a[1]的地址就是m+2

u32的数组就是m+4

u8 的数组就是 m+1

2016/4/28 SPI

1. SPI的接收寄存器

SPI\_InitStructure.SPI\_CPOL = SPI\_CPOL\_Low; //没有数据传输的时候，时钟保持低电平

SPI\_InitStructure.SPI\_CPHA = SPI\_CPHA\_1Edge; //在第一个时钟边沿处理数据

2016/4/29

1. 外部中断模式的选择，选择是中断模式还是事件模式

2016/5/5

1.平年闰年判断

if((timesVal[5]%400==0)||((timesVal[5]%4==0)&&(timesVal[5]%100!=0))) //如果为闰年

DaysOfMouth[1]=29;

else

DaysOfMouth[1]=28;

2016/5/13 中断

一. 电平同时变化

1. 中断上下边沿的时间间隔1us时。(即高电平保持1us) 结论：丢失

STM32外部中断比如，EXTI9~5的中断，选择PB6，PB7作为外部中断输入，如果外部中断同时来的话，比如：先下降后上升，第一个边沿都能进中断函数，并且按照函数顺序处理(只进一次中断)，但是，当第二个边沿来时会丢失函数顺序靠后的中断线函数。

2. 中断上下边沿的时间间隔1ms时(即高电平保持1ms) 结论：不丢失

STM32外部中断线都能很好的进入相应的中断处理函数，比如Line6与Line7 都可以相应的判断处理。

前提是中断处理函数中的程序小于1ms

3.一般在编程的时候中断中函数用systick计算时间。Systick的us级中断使用时，在systick中断函数中的程序就不能超过1us否则死中断了。

4.外部中断设置上下降沿双触发时，触发上升沿后再触发下降沿时，STM32内核可能需要时间设置，所以如果脉冲边沿间隔时间太短容易丢失，1us会丢失。

二. 电平单独线路中断

1.单个中断线的中断，当中断处理函数执行时间t大于电平跳变时间t2时，如果设置捕捉上升下降跳变中断t2的下降(或上升)沿中断会丢失。

三. STM32的中断与事件

产生中断一定产生事件，事件不可屏蔽，中断可以屏蔽

2016/5/18 串口不用microLib 的打印

1. 不用microLib的printf函数，自己的版本需要改写。原子的源码和自己写的串口初始化程序有点问题，fput需要改动待测的标志位。

2016/7/2 LWIP源码无法PING通

对应GPIO口需要改写，在初始化的时候，尤其是复位口，还有片选口，一定需要给确定的状态，在移植时，源码的片选操作端口也要对应的改

2016/7/4 1. LWIP源码网页无法打开 2. LWIP网页无法刷新

1. 数据包发送函数调用的写buff函数时序逻辑错误和datasheet不相符合(多加了一句空指令)。

2. 发送函数的计数初值类型u8 过小，实际为u16 ，网页实质已经刷新，但是前256个数据一样，所以造成没有刷新的假象。

2016/7/14—2016/7/29 nrf24l01调试经验

1. 串口调试助手，在选HEX发送时，发0x01时，只要输入 01 发送就是0x01

2. 在调试的时候，尤其是调试开发板的时候，一定需要检查板子上一些模块可能共用的总线有干扰

3. STM32 在上电后只要是没有初始化的端口，GPIO的属性都是低电平

4. 51单片机上电后只要是没有给初值的端口，IO口都是高电平

5. NRF24L01工作在增强型shockburst 模式时，发送和接收模式都需要设置自动重发功能，增强型比普通型更加可靠，有应答和自动重发的功能。

6. 操作新的模块时一定需要把工作模式和寄存器的配置弄清楚

7. 在头文件里定义数组的时候，需要加static 否则，在放在STM32f10x.h一起包含的时候，会出现重复定义。函数声明放在.h中就不会出现。

8. 在配置无线模块地址时，地址宽度P0，P1的地址最长5位，模块先写地址低字节，P1-P5的地址只有最低位可以设置。

9. 自动应答模式和自动重发关闭时，实测速率可达64KB/s，自动应答模式下速率40KB/s.

10. 只有P0地址可以接受自动应答，而且发送端发送地址等于接收地址。

STM32下载完有变化的程序后，如果出现没有变化，那么断电再开。前一次和后一次外设时钟关闭有可能没关掉，得重启

2016/8/31

STM32的I2C

**一. C语言编程函数**

1. Strstr（str1，str2）

strstr(str1,str2) 函数用于判断字符串str2是否是str1的子串。如果是，则该函数返回str2在str1中首次出现的地址；否则，返回NULL。

char \*strcpy(char\* dest, const char \*src);

说明：src和dest所指内存区域不可以重叠且dest必须有足够的空间来容纳src的字符串。

返回指向dest的指针。

2. abs(int i)

求绝对值

3. sqrt(a) , a>=0

开平方

4. int sprintf( char \*buffer, const char \*format, [ argument] … );

将 " String: %s\n" 强制转换写入到buffer[200]的数组并且 %s处 已经用s[]替换成”computer”

char buffer[200], s[] = "computer" ;

sprintf( buffer, " String: %s\n", s );

返回值是写入buffer 的字符数。

5. 高级宏应用

1. #define myprintf(...) printf("[lch]:File:%s, Line:%d, Function:%s, "

\_\_VA\_ARGS\_\_, \_\_FILE\_\_, \_\_LINE\_\_ , \_\_FUNCTION\_\_);

宏定义myprintf(…) 中的省略号会原样的替换在\_\_VA\_ARGS\_\_处。

并且\_\_VA\_ARGS\_\_只能等于常量。

2. #，##

#代表与一个字符串连接 // #n 就是以 "n" 的字符串

##代表与符号连接，可以是宏符号，或者是变量

//##n 就是和 ##左边的参数直接连接比如 a##5 就是a5

例

#define n 8+9

#define nn 123

#define mm(n) printf( "这是一个测试" #n "测试 " "%d" , ##nn)

3. #define PDEBUG(fmt, args...) printk( KERN\_DEBUG "scull: " fmt, ## args)

宏定义中的省略号表示一串可变的参数, 这一串参数可用args加以引用.

例

PDEBUG("a=%d, b=%d", a, b); 宏展开 printk( KERN\_DEBUG "scull: " "a=%d, b=%d", a, b);

4. 字节内存填充的问题 \_packed 关键词是编译器语言

typedef \_\_packed struct{…}

\_\_packed表示此结构体成员变量不进行内存填充，优点是方便指针读数据，缺点是不方便CPU操作

比如：  
typedef struct typedef \_\_packed struct   
{ {  
  char x; char x;  
  int y;  int y;   
}struct1; }struct2;

在32位的ARM SDT编译器中，编译  
sizeof(struct1)==8，  char x; 会在x后面填充3个字节  
sizeof(struct2)==5

5. assert(n!=0); 或者 ASSERT(n!=0) 参数检查，程序判断n不能等于0 ，否则报错

System("pause"); 系统暂停程序，会打印“按任意键继续”

6. 1 ul

定义1是unsigned long型数据，同理1u 代表定义为unsigned int 型

7. sizeof(数组名)

求占用空间长度

8. int strncmp(str1, str2, n)

比较前n个str1和str2的ASCII码值，相等返回0, 若s1>s2,返回大于0 的值，s1<s2,返回小于0的值。

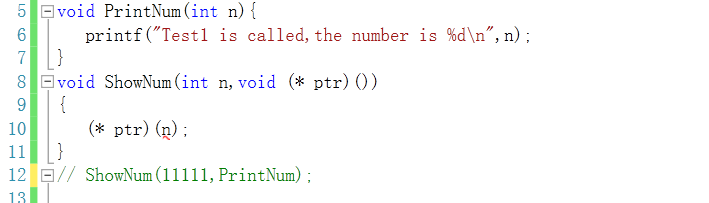
Int strcmp(const char\*s1, const char \*s2)

当s1<s2时，返回为负数；

当s1=s2时，返回值= 0；

当s1>s2时，返回正数。

9. 函数指针



Void showNum(int n, void (\*ptr)() );

Void (\*ptr)()这里是声明了一个void类型的函数指针变量ptr 。

这里的ptr需要用括号括起来void代表无返回值，但是去掉括号就会有歧义混乱。所以函数指针一定要用括号括起来。

10. 联合体：

————几个不同的变量共同占用一段内存的结构。

单片机中经常会遇见分离高低字节的操作，比如进行计时中断复位操作时往往会进行

(65535-200)/256,(65535-200)%256这样的操作，而一个除法消耗四个机器周期，取余也需要进行一些列复杂的运算，如果在短时间内需要进行很多次这样的运算无疑会给程序带来巨大的负担。其实进行这些操作的时候我们需要的仅仅是高低字节的数据分离而已，这样利用联合体我们很容易降低这部分开销。

代码：

union chufa{

int n; //n中存放要进行分离高低字节的数据

char a[2]; //在keil c中一个整形占两个字节，char占一个字节，所以n与数组a占的字节数相同

}test;

test.n=65535-200; //进行完这句后就一切ok了，下面通过访问test中数组a的数据来取出高低字节的数据

TH1=test.a[0]; //test.a[0]中存储的是高位数据，这是由于keil的特性，如果其他编译器还请自测

TL1=test.a[1]; //test.a[1]中储存了test.n的低位数据

!that's good,仅仅用了一条减法指令就达到了除法、取余的操作，在进行高频率定时时尤为有用。

11. C语言中四种存储类别

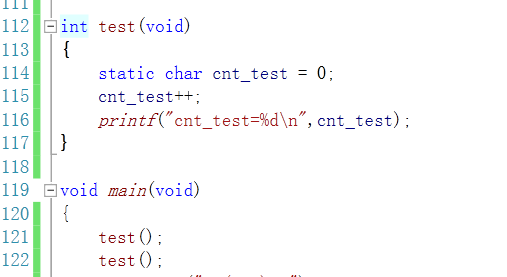
C语言中变量的声明/定义格式如下：  
存储类型 类型修饰符 数据类型 变量名;

Static… long、const… int、char… aa…

Static变量（静态）

（1）限制作用域。利用这一特性可以在不同的文件中定义同名函数和同名变量，而不必担心命名冲突。

（2）静态存储区，类似全局变量，变量的内存只分配一次，但是仍然有（1）中的作用。

（3）static和全局变量一样初始化默认值为0。

（4）static申明局部变量，当第二次进入子函数时，会将上次变量结果值赋给变量

Extern （静态）

Auto （动态）

即局部变量，非全局变量的缺省值假定为auto。故基本不用。

Register（动态）

把变量优先放在寄存器中操作，提高速度，不常用。

const 常量，数值不可以更改

volatile 每次取值必须读内存

code在STM32中是把内容放在片上flash

12. 负数 = 正数取反+1

13. C语言中程序运行时具有以下几个存储空间 （大体上分成数据区和代码区）

静态存储区：静态数据、全局变量和常量

栈区：局部变量（结束自动释放）

堆区：动态分配内存（malloc，calloc,free） \_msize(void\* p) 得到malloc分配的内存大小

代码区：存放代码

文字常量区：常量定义的字符串

函数指针指向：代码区

数据指针指向：堆，栈，数据存储区

14. Program Size: Code=8044 RO-data=52 RW-data=56 ZI-data=5184

Code指存储到flash【Rom】中的程序代码，代码被系统添加了一部分初始化ZI的代码，编程者看不到这部分。

 ZI英语是zero initial，就是程序中用到的，并且被系统初始化为0的变量的字节数，keil编译器默认是把你没有初始化的变量都赋值一个0，这些变量在程序运行时是保存在RAM中的

RW是可读可写变量，就是初始化时候就已经赋值了的，RW + ZI就是你的程序总共使用的RAM字节数。

RO，这个是初始化的常量，但是这些值是被保存到Rom中的，就放在主程序后面，这样所写的程序占用的rom的字节总数就是Code + RO + RW。为什么Rom中还要存RW，还有ZI函数呢，因为掉电后RAM中所有数据都丢失了，每次上电RAM中的数据是被重新赋值的，每次这些固定的值就是存储在Rom中的。

15. 回调函数一般形式

Typedef void (\*cbShow)(int n);

void ShowNum(cbShow pFun， int n)

{ pFun(n); } //showNum就是一个回调函数

void PrintNum(int n){

printf("Test1 is called,the number is %d\n"，n);

}

主函数中

ShowNum( PrintNum，11111)；

16. void \*memset(void \*s, int ch, size\_t n);

函数解释：将s中前n个字节 （typedef unsigned int size\_t ）用 ch 替换并返回指向 s 的指针。

memset：作用是在一段内存块中填充某个给定的值，它是对较大的结构体或数组进行清零操作的一种最快方法。

17. htonl()函数

字节顺序转换为网络字节顺序，即改变大小尾格式

18. strlen()

求字符串长度不包含 \0

19. 处理字符串尾部要加0

20. 对于短小函数的实现， C中是宏定义

C++中是内联函数

21. 内存中的地址都是字节

unsigned long \*p2;

p2=(unsigned long \*)0x810000;

p2+5 = 0x810014

22. sizeof如用于数组，只能测出静态数组的大小，无法检测动态分配的或外部数组大小。

23. 指针

Char const \* p = const char\* p 读出数据是常量,函数指针能够对任意地址进行只读操作

Char\* const p 函数指针是常量， 只能对固定的位置进行读写操作

int (\*p)[n]

// 数组指针，初始化例子： int a[x][n]; (\*p)[n]=a; 定义一个行指针,并指向二维数组的第0行

int \* p[n] // 指针数组

24. 无限循环

u8 i;

for(i=0;i<=255;i++)

当i执行到255时，i会255+1=0，则无限循环

25. 系统暂停

引入头文件”stdlib.h” System("pause"); //按任意键继续

26. 使用循环一定要注意再次循环时的相关参数是否需要复位

27.随机数

srand((u16)time(NULL)); 下种子

rand()%N 生成 0—N-1 的随机数

28. 字符的表示

对于无符号数来说，字符”-1” = 0xff = 0-1= -1

29. C语言数组从0开始，循环处要格外小心

30. 柔性数组，静态链表中使用了柔性数组

31.

Char a[]=’1’;

Void p(char \* m);

函数调用P(a)

sizeof(a) = 4； //这里子函数求得是 指针的大小

常用算法 具体相关算法实现练习见VS2010的test项目tat.c练习

1. 4字节对齐算法 define MEM\_ALIGNMENT 4

( ((size) + MEM\_ALIGNMENT- 1) & ~(MEM\_ALIGNMENT-1) )

2. 冒泡法排序

3. 两分法定位

一般都是先排序，然后用两分法排序

HTML调试经验

1. 提倡全部对属性值加双引号（参数值为数字除外）。（颜色用rgb时也不例外）

例： <font color="red" face="宋体" size="7">被设置的内容</font>

2. <hr/> 下划线

如果是标签中需要设置属性的话，那 / 放在最末尾

3. URL(Uniform Resource Locator)中文名字为“统一资源定位器”。

4. 表单，是HTML页面与浏览器端实现交互的重要手段。

PQM09调试

1. DSP下载程序(XP系统)，选型号(初次)，选程序，有的需要解锁(密码)。

具体：连接好后，先设备上电，然后再开软件连接。

2. 设备接线(电压电流调试线)，三相台参数设置，台式机软件启动配置参数，仪表，下载增益后，设备上电

3. 调节无功功率需要，发送电流每相加90度

4. 调节有功和正向无功，P>0时需要0.5L（感性电阻电压超前于电流），然后0. 5C（容性电阻，电压滞后于电流），最后调节相角选1.0，然后电压夹角，最后看设备电压不平衡度要<%1

ERP流程

1. 车间生产一概下生产订单，生产需要出库需要生产订单。研发除外。