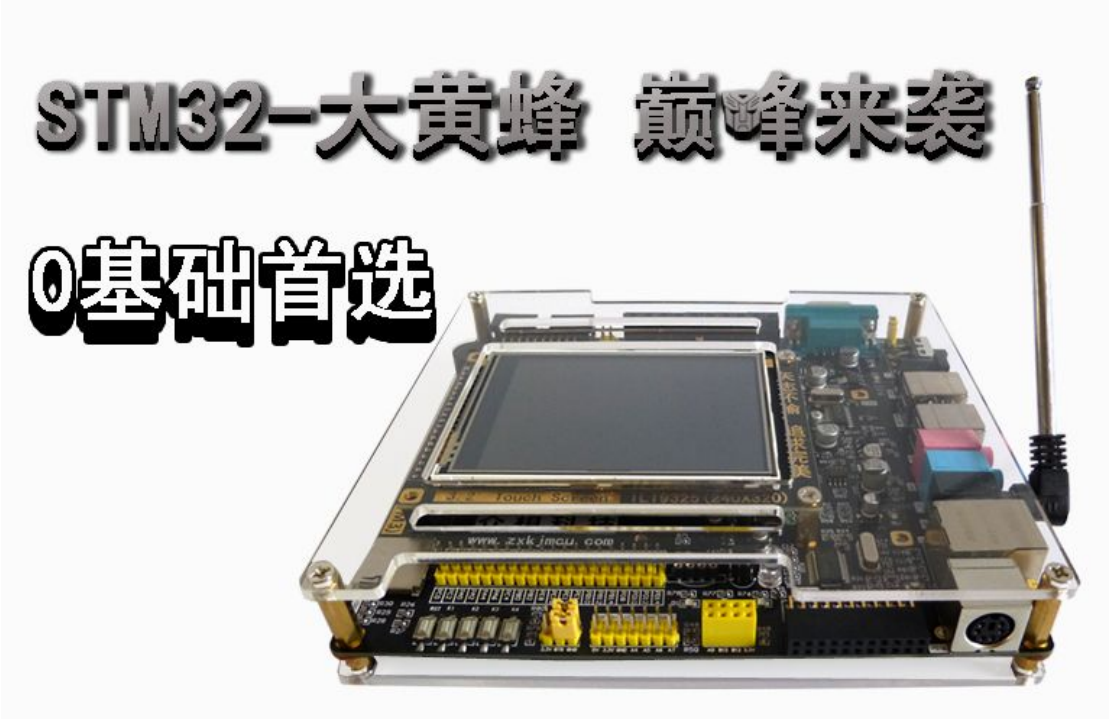
学 ARM 从 STM32 开始

STM32 开发板库函数教程--实战篇



官方网站：http://www.zxkjmcu.com
官方店铺：http://zxkjmcu.taobao.com
官方论坛：http://bbs.zxkjmcu.com
刘洋课堂：http://school.zxkjmcu.com

4.20 STM32 程序加密

4.20.1 概述

4.20.1.1 加密定义

加密，是以某种特殊的算法改变原有的信息数据，使得未授权的用户即使获得了已加密的信息，但因不知解密的方法，仍然无法了解信息的内容。

4.20.1.2 加密分类

加密建立在对信息进行数学编码和解码的基础上。加密类型分为两种，对称加密与非对称加密，对称加密双方采用共同密钥，（当然这个密钥是需要对外保密的），这里讲一下非对称加密，这种加密方式存在两个密钥，密钥-- 一种是公共密钥（正如其名，这是一个可以公开的密钥值），一种是私人密钥（对外保密）。 您发送信息给我们时，使用公共密钥加密信息。 一旦我们收到您的加密信息，我们则使用私人密钥破译信息密码（被我们的公钥加密的信息，只有我们的唯一的私钥可以解密，这样，就在技术上保证了这封信只有我们才能解读——因为别人没有我们的私钥）。 使用私人密钥加密的信息只能使用公共密钥解密（这一功能应用与数字签名领域，我的私钥加密的数据，只有我的公钥可以解读，具体内容参考数字签名的信息）反之亦然，以确保您的信息安全。

4.20.2 STM32 STM32 单片机 ID 号加密简介

4.20.2.1 STM32 ID 号是 12 个字节身份标识寄存器。

4.20.2.2 STM32 单片机产品都具备唯一身份标识寄存器(96 位)

产品唯一的身份标识非常适合：

- 用来作为序列号(例如 USB 字符序列号或者其他的终端应用)

● 用来作为密码，在编写闪存时，将此唯一标识与软件加解密算法结合使用，提高代码在闪存存储器内的安全性。

● 用来激活带安全机制的自举过程

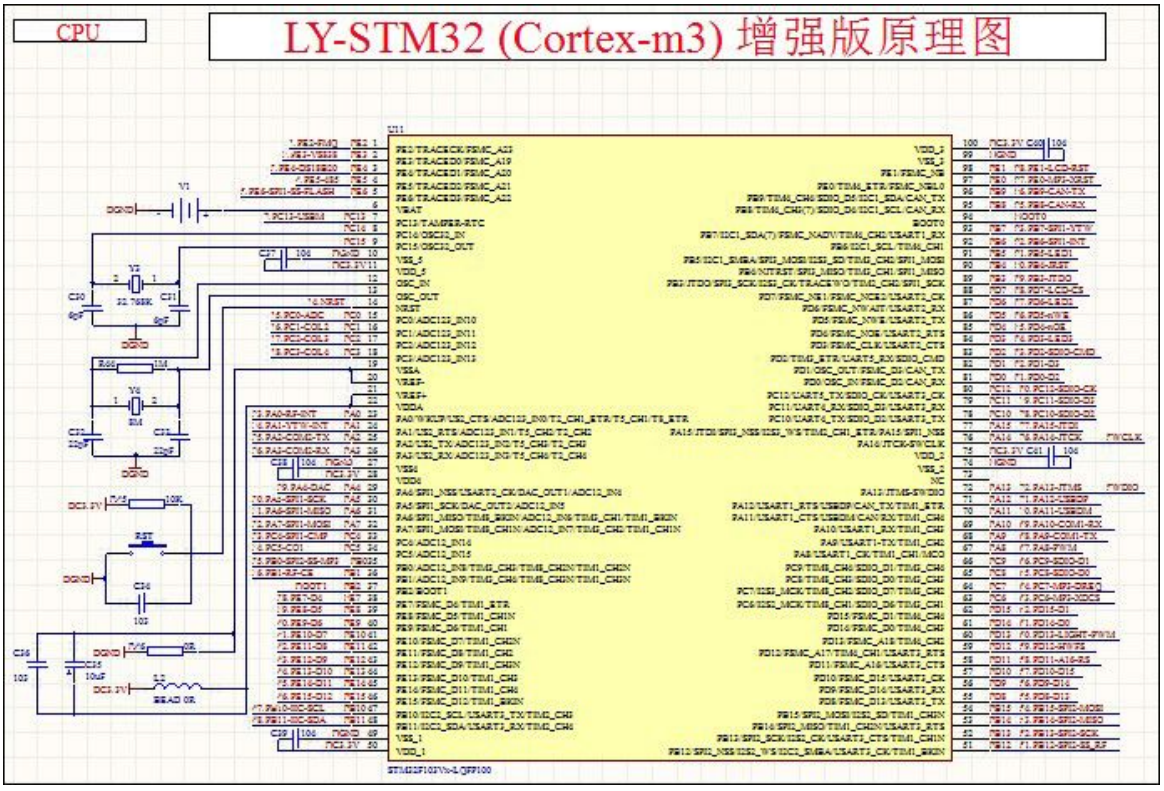
96 位的产品唯一身份标识所提供的参考号码对任意一个 STM32 微控制器，在任何情况下都是唯一的。用户在何种情况下，都不能修改这个身份标识。 这个 96 位的产品唯一身份标识，按照用户不同的用法，可以以字节(8 位)为单位读取，也可以以半字(16 位)或者全字(32 位)读取。

4.20.3 实验目的

当给实验板送点以后，程序读取 ID 号，通过 prinif 重定向程序把写入 ID 号打印到串口精灵上。如果读出来的 ID 号等于我们预先设置好的 ID 号，主程序顺序执行；如果读出来的 ID 号不等于我们预先设置好的 ID 号，程序进入死循环。

4.20.4 硬件设计

利用实验板上的 CPU 电路，可以很方便的实现加密功能。



4.20.5 软件设计

4.20.5.1 软件设计说明

这次试验是先把 ID 号读出来，并打印输出，送到显示器显示出来。设计思路是这样的：

1、读取芯片中的 ID 号，定义数组，存放读取的 ID 号；

2、打印输出到显示器；

3、和设置好的 ID 号比较；

4.20.5.2 加密方式相关函数简介

FLASH 是很重要的，在下面我们着重介绍相关应用函数，先熟悉应用函数然后编写程序。

1、函数 FLASH_ClearFlag（清除 FLASH 待处理标志位）

表 1 描述了函数 FLASH_ClearFlag

函数名称	FLASH_ClearFlag
函数原型	void FLASH_ClearFlag(u16 FLASH_Flag)

功能描述	清除 FLASH 待处理标志位
输入参数	FLASH_FLAG：待清除的标志位 参阅 Section：FLASH_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

FLASH_FLAG 为能够被函数 FLASH_ClearFlag 清除的标志位。它们列举于下表：

附表 1 FLASH_FLAG 值

FLASH_FLAG	描述
FLASH_FLAG_BSY	FLASH 忙标志位
FLASH_FLAG_EOP	FLASH 操作结束标志位
FLASH_FLAG_PGERR	FLASH 编写错误标志位
FLASH_FLAG_WRPRTERR	FLASH 页面写保护错误标志位

2、函数函数 FLASH_ErasePage（要擦出页的起始地址）

表 2 描述了函数 FLASH_ErasePage

函数名称	FLASH_ErasePage
函数原型	FLASH_Status FLASH_ErasePage(u32 Page_Address)
功能描述	擦除一个 FLASH 页面
输入参数	无
输出参数	无
返回值	擦除操作状态
先决条件	无
被调用函数	无

4.20.5.3 STM32 库函数文件

```
stm32f10x_gpio.c
stm32f10x_rcc.c
Misc.c  // 中断控制字（优先级设置）库函数
stm32f10x_exti.c // 外部中断库处理函数
stm32f10x_tim.c  // 定时器库处理函数
stm32f10x_usart.c  // 串口通讯函数
```

本节实验及以后的实验我们都是用到库文件，其中 stm32f10x_gpio.h 头文件包含了 GPIO 端口的定义。stm32f10x_rcc.h 头文件包含了系统时钟配置函数以及相关的外设时钟使能函数，所以我们要把这两个头文件对应的

stm32f10x_gpio.c 和 stm32f10x_rcc.c 加到工程中；Misc.c 库函数主要包含了中断优先级的设置，stm32f10x_exti.c 库函数主要包含了外部中断设置参数，tm32f10x_tim.c 库函数主要包含定时器设置，tm32f10x_usart.c 库函数主要包含串行通讯设置，这些函数也要添加到函数库中。以上库文件包含了本次实验所有要用到的函数使用功能。

4.20.5.4 自定义头文件

```
pbdata.h
pbdata.c
```

同时我们自己也创建了两个公共的文件，这两个文件主要存放我们自定义的公共函数和全局变量，以方便以后每个功能模块之间传递参数。

4.20.5.5 pbdata.h 文件里的内容是

```
#ifndef _pbdata_H
#define _pbdata_H

#include "stm32f10x.h"
#include "misc.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_tim.h"
#include "stm32f10x_usart.h"
#include "sttdio.h"

extern u8 dt;//定义变量

void RCC_HSE_Configuration(void); //定义函数
void delay(u32 nCount);
void delay_us(u32 nus);
void delay_ms(u16 nms);

#endif
```

语句 #ifndef、#endif 是为了防止 pbdata.h 文件被多个文件调用时出现错误提示。如果不加这两条语句，当两个文件同时调用 pbdata 文件时，会提示重复调用错误。

4.20.5.6 pbdata.c 文件里的内容是

```
#include "pbdata.h" //很重要，引用这个头文件

u8 dt=0;

void RCC_HSE_Configuration(void) //HSE 作为 PLL 时钟，PLL 作为 SYSCLK
{
   RCC_DeInit(); /*将外设 RCC 寄存器重设为缺省值 */
   RCC_HSEConfig(RCC_HSE_ON);    /*设置外部高速晶振（HSE） HSE 晶振打开(ON)*/

   if(RCC_WaitForHSEStartUp() == SUCCESS) {  /*等待 HSE 起振,  SUCCESS：HSE 晶振稳定且就绪*/

   RCC_HCLKConfig(RCC_SYSCLK_Div1);/*设置 AHB 时钟(HCLK)RCC_SYSCLK_Div1——AHB 时钟 = 系统时*/
   RCC_PCLK2Config(RCC_HCLK_Div1); /*设置高速 AHB 时钟(PCLK2)RCC_HCLK_Div1——APB2 时钟 = HCLK*/
   RCC_PCLK1Config(RCC_HCLK_Div2); /*设置低速 AHB 时钟(PCLK1)RCC_HCLK_Div2——APB1 时钟 = HCLK / 2*/

   RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);/*设置 PLL 时钟源及倍频系数*/
   RCC_PLLCmd(ENABLE);    /*使能 PLL */
   while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) ; /*检查指定的 RCC 标志位(PLL 准备好标志)设置与否*/

   RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);  /*设置系统时钟（SYSCLK） */
   while(RCC_GetSYSCLKSource() != 0x08);      /*0x08：PLL 作为系统时钟 */
 }
}
void delay(u32 nCount)
{
    for(;nCount!=0;nCount--);
}
/*******************************************************************************
* 名    称：delay_us(u32 nus)
* 功    能：微秒延时函数
* 入口参数：u32  nus
* 出口参数：无
* 说    明：
* 调用方法：无
*******************************************************************************/
void delay_us(u32 nus)
```

```
{
    u32 temp;
   SysTick->LOAD = 9*nus;
   SysTick->VAL=0X00;//清空计数器
   SysTick->CTRL=0X01;//使能，减到零是无动作，采用外部时钟源
   do
   {
    temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(!(temp&(1<<16))));//等待时间到达

    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}

/*****************************************************************************
* 名    称：delay_ms(u16 nms)
* 功    能：毫秒延时函数
* 入口参数：u16 nms
* 出口参数：无
* 说    明：
* 调用方法：无
*****************************************************************************/
void delay_ms(u16 nms)
{
   u32 temp;
    SysTick->LOAD = 9000*nms;
   SysTick->VAL=0X00;//清空计数器
   SysTick->CTRL=0X01;//使能，减到零是无动作，采用外部时钟源
   do
   {
    temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(!(temp&(1<<16))));//等待时间到达
    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}
```

4.20.6 STM32 系统时钟配置 SystemInit()

每个工程都必须在开始时配置并启动 STM32 系统时钟。

4.20.7 GPIO 引脚时钟使能

```
    SystemInit();//72m
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);//设置串口 1 时钟使
```

```
能
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//功能复用 IO 时钟使能
```

本节实验用到了 PA 端口，所以要把 PA 端口的时钟打开；串口 1 时钟源是通过 APB2 预分频器得到的，串口 1 时钟初始化；因为要与外部芯片通讯，所以要打开功能复用时钟。

4.20.8 GPIO 管脚电平控制函数

在主程序中采用 while(1)循环语句，等待串口通讯中断的到来。

```
    while(1);
```

4.20.9 stm32f10x_it.c 文件里的内容是

在中断处理 stm32f10x_it.c 文件里中仅串口 1 子函数非空，进入中断处理函数后，只有串口 1 有参数输出。

```
    #include "stm32f10x_it.h"
    #include "stm32f10x_exti.h"
    #include "stm32f10x_rcc.h"
    #include "misc.h"
    #include "pbdata.h"

    void NMI_Handler(void)
    {
    }

    void USART1_IRQHandler(void)
    {
       if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET)
       {
          USART_SendData(USART1,USART_ReceiveData(USART1));
          while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
       }
    }
```

4.20.10 main.c 文件里的内容是

大家都知道 prinif 重定向是把需要显示的数据打印到显示器上。STM32 的 ID 号一共是 12 个字节，一定要全部读取。ID 号在 FLASH 中的地址是固定的，起始地址是：0x1FFFF7E8。读取的时候直接从这个起始地址顺序读出。

然后通过 prinif 重定向打印到串口来查看 ID 号是多少。

详细请参考程序注释。

```
#include "pbdata.h"

void RCC_Configuration(void);
void GPIO_Configuration(void);
void NVIC_Configuration(void);
void USART_Configuration(void);

int fputc(int ch,FILE *f)//打印输出到显示器上
{
    USART_SendData(USART1,(u8)ch);
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    return ch;
}

int main(void)
{
    u8  Sys_ID[12],i;//定义一个数组 Sys_ID[]，存放 ID 号

    RCC_Configuration(); //系统时钟初始化
    GPIO_Configuration();//端口初始化
    USART_Configuration();
    NVIC_Configuration();

    for(i=0;i<12;i++)//读出 12 组数据（ID 号）
    {
        Sys_ID[i]=*(u8*)(0x1FFFF7E8+i);//读 FLASH 中的 ID 号，起始地址是
0x1FFFF7E8
        printf(" %0.2X",Sys_ID[i]);//把 12 组 ID 号打印输出
    }

    if(Sys_ID[0]==0x36 && Sys_ID[1]==0xFF && Sys_ID[2]==0xD7 &&
       Sys_ID[3]==0x05 && Sys_ID[4]==0x34 && Sys_ID[5]==0x52 &&
       Sys_ID[6]==0x32 && Sys_ID[7]==0x31 && Sys_ID[8]==0x15 &&
       Sys_ID[9]==0x83 && Sys_ID[10]==0x11 && Sys_ID[11]==0x43)//判断数组，和
设置好的 ID 号作比较
    {
       printf("\r\n 通过\r\n");
    }
    else
    {
```

```
        printf("\r\n 失败\r\n");
        //while(1);//ID 号不对，程序死循环，不工作。
    }
    while(1); //ID 号对，程序顺序执行
}

void RCC_Configuration(void)
{
    SystemInit();//72m
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
}

void GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    //LED
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;//TX
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA,&GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;//RX
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA,&GPIO_InitStructure);
}

void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

void USART_Configuration(void)
{
    USART_InitTypeDef  USART_InitStructure;
```

```
    USART_InitStructure.USART_BaudRate=9600;
    USART_InitStructure.USART_WordLength=USART_WordLength_8b;
    USART_InitStructure.USART_StopBits=USART_StopBits_1;
    USART_InitStructure.USART_Parity=USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_
None;
    USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;

    USART_Init(USART1,&USART_InitStructure);
    USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
    USART_Cmd(USART1,ENABLE);
    USART_ClearFlag(USART1,USART_FLAG_TC);
}
```

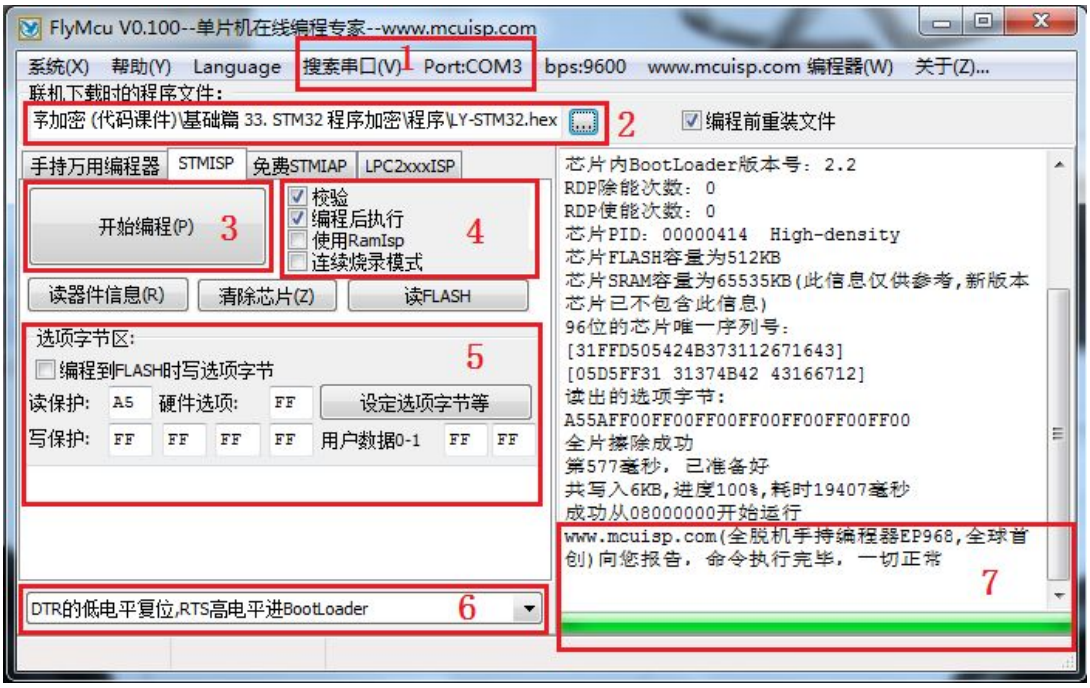
在 main(void)程序体中代码比较多，大部分都是前期课程的延伸，新的知识点不多，要注意的是第一次读取单片机中 ID 号时肯定是错误的，因为我们不知道这片单片机中真正的 ID 号是多少；只有通过第一次读取，我们才能得到正确的 ID 号码。

4.20.11 在串口窗口中设置写保护

4.20.12 程序下载

请根据下图所指向的 7 个重点区域配置。其中（1）号区域根据自己机器的实际情况选择，我的机器虚拟出来的串口号是 COM2。（2）号区域请自己选择程序所在的文件夹。（7）号区域当程序下载完后，进度条会到达最右边，并且提示一切正常。（4、5、6）号区域一定要按照上图显示的设置。当都

设置好以后就可以直接点击（3）号区域的开始编程按钮下传程序了。



本节实验的源代码在光盘中：（LY-STM32 光盘资料\1.课程\1,基础篇\基础篇 33. STM32 程序加密\程序）

4.20.13 实验效果图

初始化程序写入实验板后，就等待中断的到来。使用公司开发的多功能监视系统。初始画面如下图所示。

点击左上角的“串口调试”按钮，在串口调试界面中的接收区就会接收到 FLASH 中读出的 ID 号数据。请大家注意：由于我们的主程序仅仅读取一次后就打印到串口调试界面中。如果还想重复打印输出 ID 号，就必须按“复位”键，让程序重新执行上述过程。

下面的实验画面中的 5 组数据是复位了 5 次读出的 ID 号。