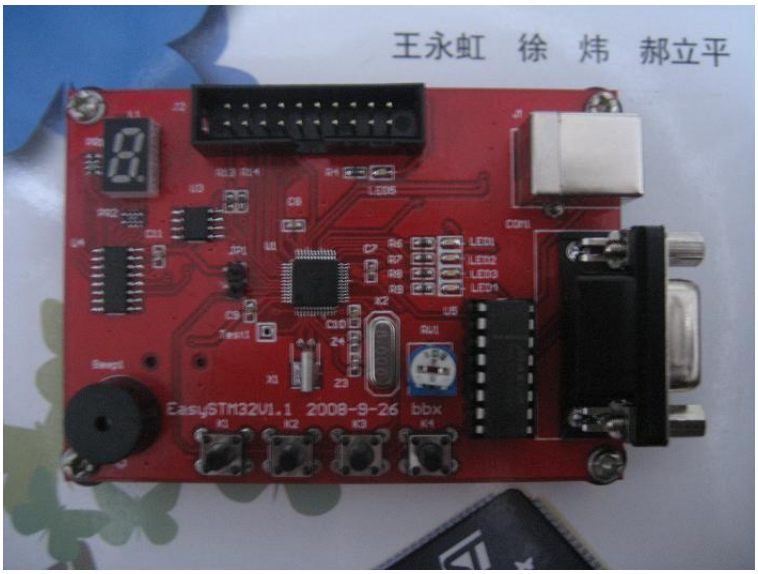


EasySTM32 开发板
说明书

bbx

2008-9-24

http://auction1.taobao.com/auction/item_detail-0db1-b8d83af207f1916f6d6bfc886bd9f586.jhtml

目录

一 前言

当今的世界电子产业发展迅速，各种新芯片陈出不穷。回想 5 年前刚毕业时候还只知 51，后来接触 AVR、PIC 等单片机，再后来这 2 年 ARM7、ARM9 满地都是，现在 ARM 公司又最新推出了 CortextM3 核。需要学习的东西实在是太多了。当然其实各种 MCU 都是相通的，掌握了一种，其他掌握也不困难。CortextM3 核被誉为是 ARM7 的革命版，对比 ARM7 性价比实在是强上太多，我用过一次后就再也不想用 ARM7 了。STM32 是 ST 公司最新推出的使用 CortextM3 核的系列芯片，在先进的 CortextM3 核的基础上增加了众多外设，功能非常强大，价格却很便宜。我个人认为可以把目前各种中档以上 MCU、包括各种集成 Flash 的 ARM7 一网打尽，优势实在太大，实在是不能不学，不能不用！

EasySTM32 学习板就是我在工作之余制作的一块简单的 STM32 学习板。该板可以不用仿真器，直接下载程序，方便初学者。同时也提供 JTAG 标准接口，如果你有仿真器也可以直接使用。该板虽然简单，但是也可以完成包括 USB 虚拟串口等众多实验，让大家初步领略 STM32 的外设使用、开发过程。

有关 STM32 的最详细、最权威的资料都在 ST 的网站上（www.st.com），包括数据手册、各种例程，大家可以自行查找。本实验板只是起到入门作用，想更深入了解 STM32 需要大家自己的不懈努力了！

免责声明：

对于在本实验板以外的硬件上，应用本实验板提供的例程或部分程序，或者应用本实验板电路所造成的后果不承担任何责任。

二 实验板使用说明

使用本实验板，大家需要准备一条 USB 方头线（给实验板供电、USB 实验使用），一条串口延长线（公母头，下载程序、串口实验使用），一台有串口的 PC（没 PC 本文档也看不到了）。

收到实验板后，断开 JP1 跳线，插入 USB 方头线，板上 Led1~Led5 全亮、数码管全亮、同时蜂鸣器开始唱歌，这就表示实验板工作正常，否则请及时和我联系。

三 EasySTM32 原理图

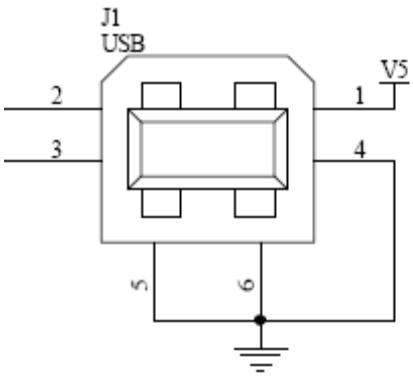
附带文档中包含了实验板的原理图。下面对原理图做一些简要说明。

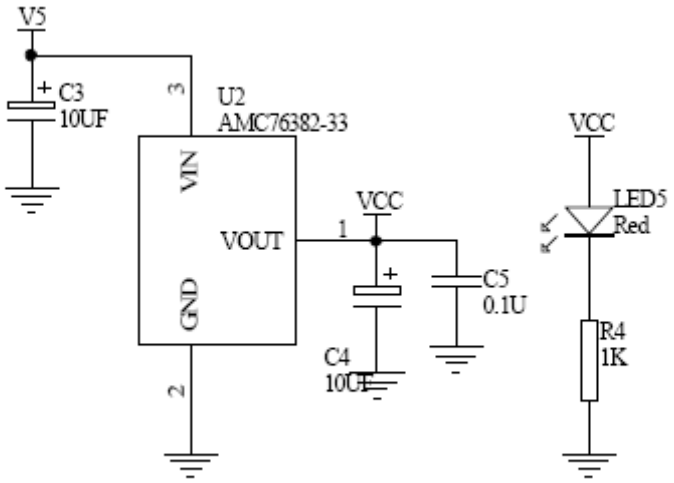
1 STM32 最小系统

1）电源

实验板使用 USB 的 5V 供电，通过 LDO 转换为 STM32 需要的 3.3V。LED5 为电源指示，只要实验板带电，LED5 就应该会一直亮着。

（注：STM32 为单一 3.3V 供电芯片）

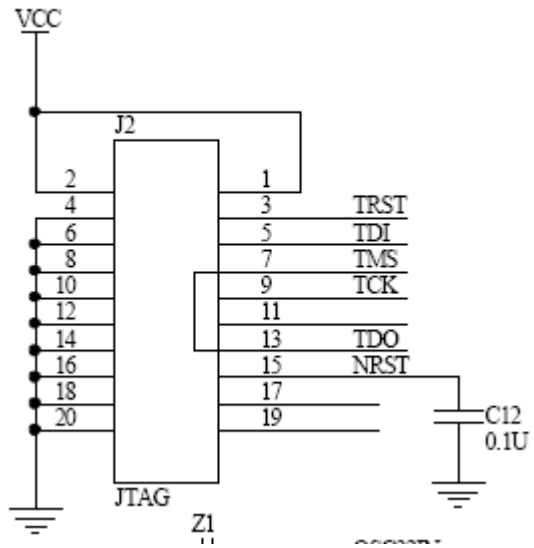




2）JTAG

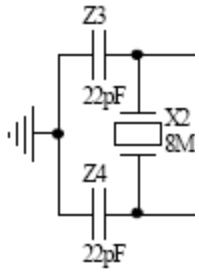
使用标准 20 芯 JTAG 电路，可以直接使用各种 STM32 的仿真器。

建议大家使用仿真器来调试程序，这样更加直观、快捷。



3）外部晶振

使用 8M 外部晶振，和 STM32 官方开发板上一致。

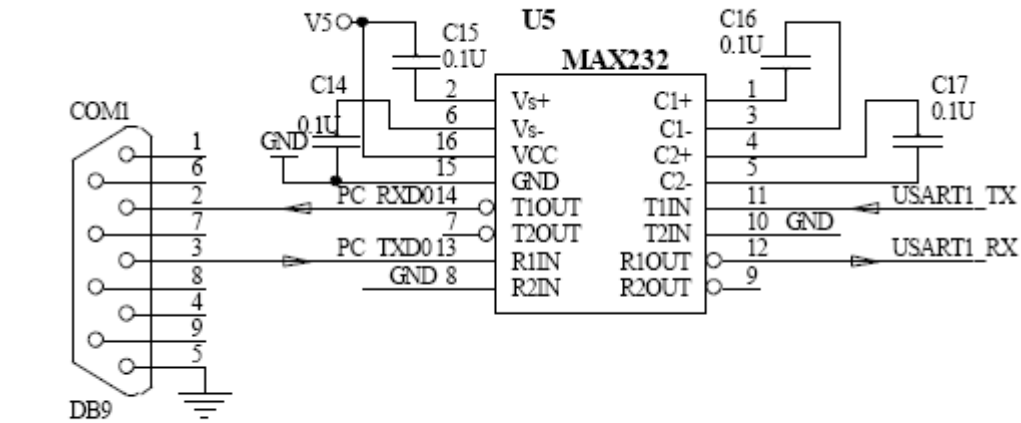


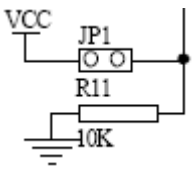
4）ISP

STM32 可以直接通过串口 **1** 做 **ISP**，直接下载编译生成的**.hex**，**.bin** 文件。

JP1 是进入 **ISP** 的选择跳线。

具体 **ISP** 的使用下面有介绍。

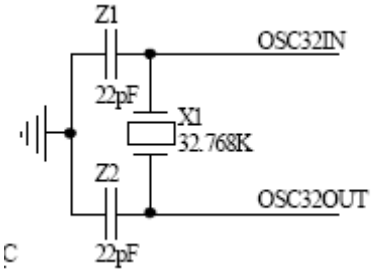




5）RTC

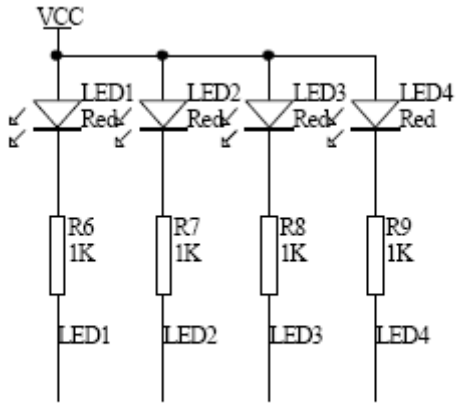
STM32 内带 **RTC** 电路，外面只需要一个 **32.768K** 的晶振、以及一个电源。

要注意 **32.768** 晶振的负载电容，下图的 **Z1**、**Z2** 是和焊接上的 **32.768K** 晶振相关的。



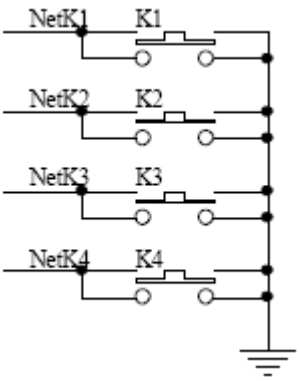
2 LED 实验电路

STM32 的 IO 可以直接驱动 LED。



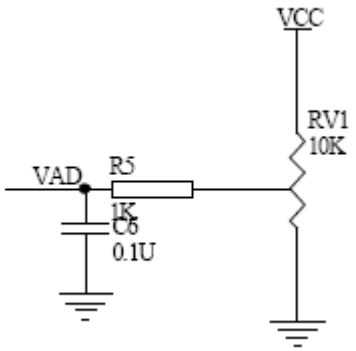
3 按键实验电路

STM32 的 IO 可以设置为上拉输入，所以按键直接接 IO 口。



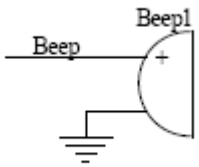
4 AD 实验电路

通过一个电位器调节分压，得到不同的电压。



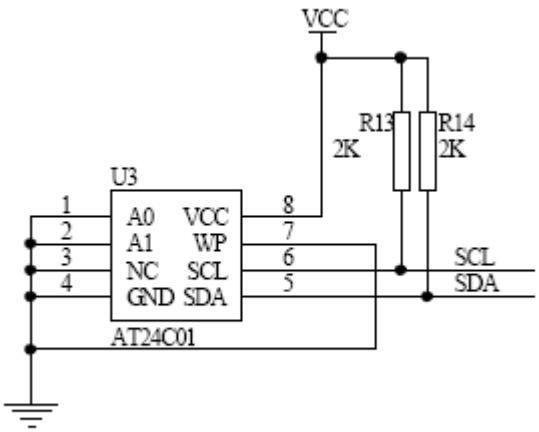
5 PWM 实验电路

PWM 管脚直接接无源蜂鸣器，可以通过改变 PWM 的频率改变蜂鸣器的音调。



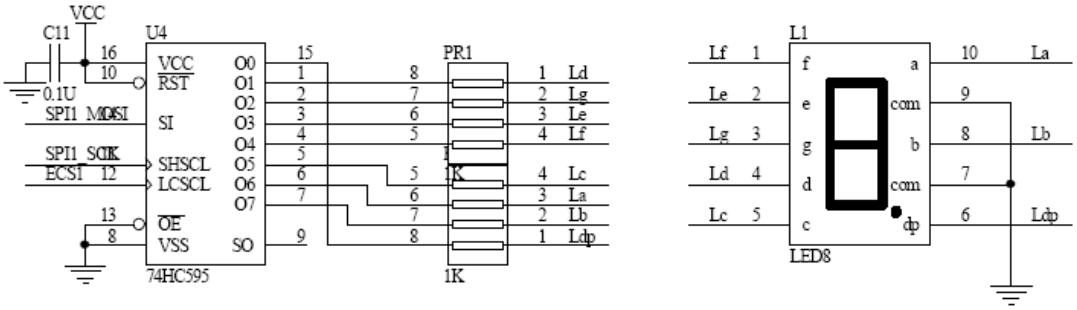
6 I2C 实验电路

使用最常用的 I2C 接口 E2PROM AT24C01 来做 I2C 实验。



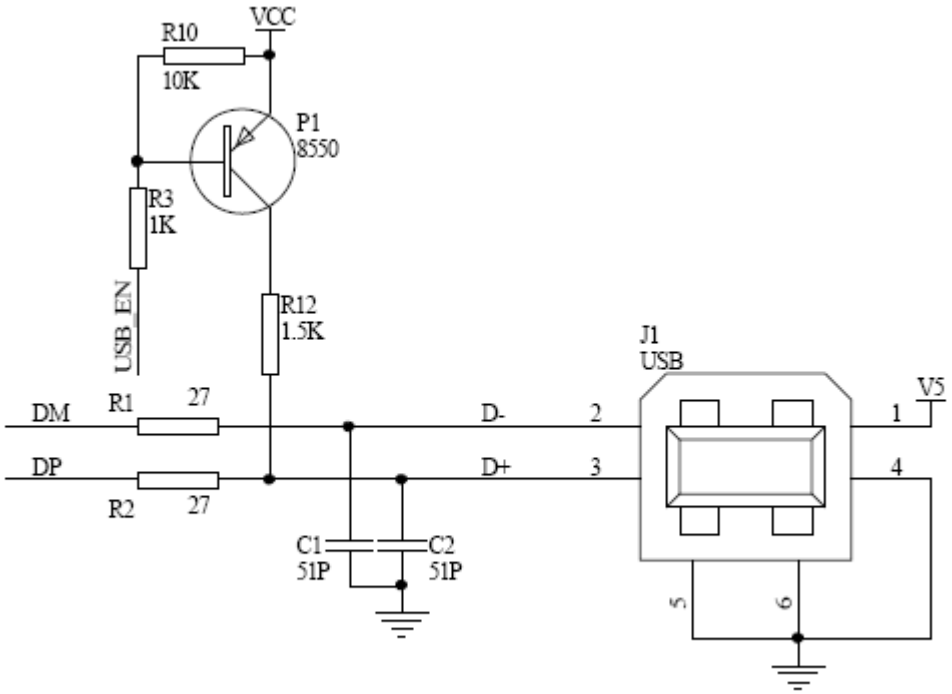
7 SPI 实验电路

使用 **SPI** 控制 **74HC595**，驱动一个数码管做显示。



8 USB 实验电路

STM32 内含 **USB** 接口，只需要一点点元件就可以实现 **USB** 功能。



三 开发工具简介

目前 **STM32** 最主要有 **Keil**、**IAR**，**2** 种开发环境。各有各的特点，相互之间转换也很方便，大家可以任意选用。本实验板的例程都采用 **Keil** 做开发环境。有关开发环境使用大家

可以网上查找相关资料。

STM32 的仿真器也有很多种，**ULINK2** 在 **Keil** 下使用，**JLINK** 可以在 **IAR**、最新的 **Keil** 下使用。此外现在 **HJTAG** 也可以调试 **STM32.**还有专门开发 **STM32** 的 **ST-LINKII**，价格低廉。大家可以自行购买。

开发 **STM32** 也可以不使用仿真器，直接使用芯片的 **ISP** 功能来把编译好的程序下载调试。这对初学者是最廉价的方式。

四 基础实验

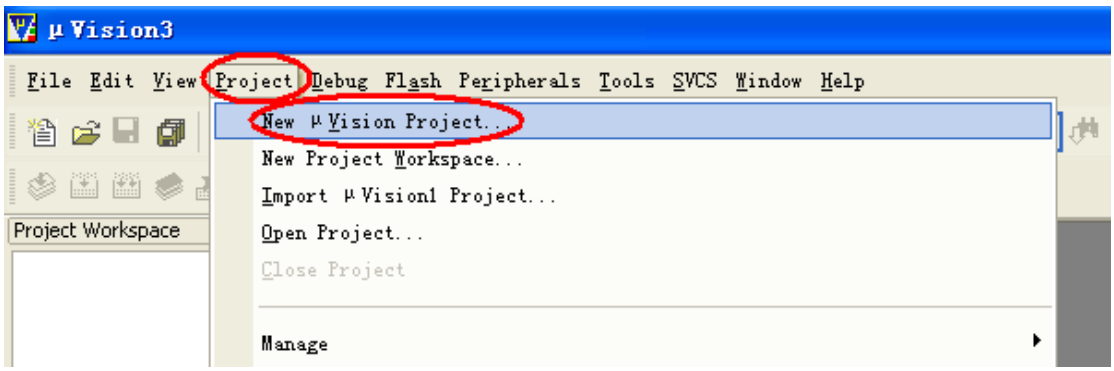
1 Keil 中新建工程实验

本实验板上所有例程都在 **Keil** 下开发。

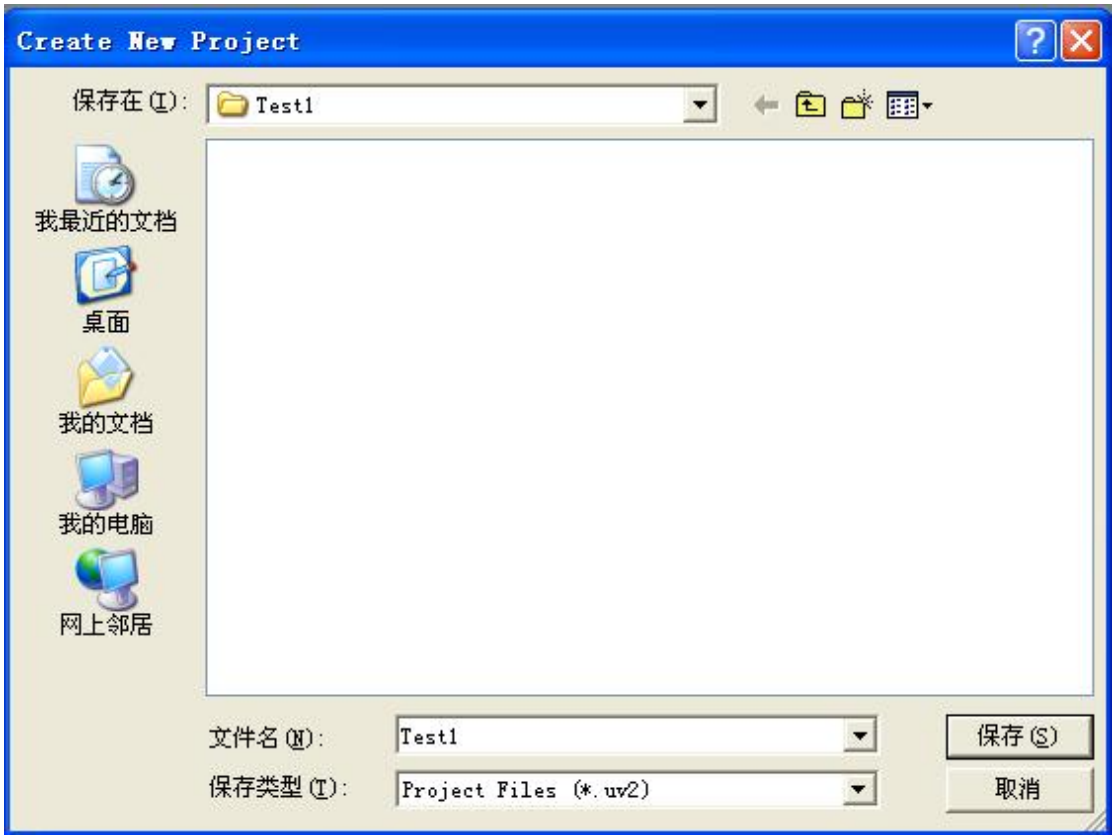
1）首先下载并安装 **Keil**，大家可以在网上下载安装

2）打开 **Keil**

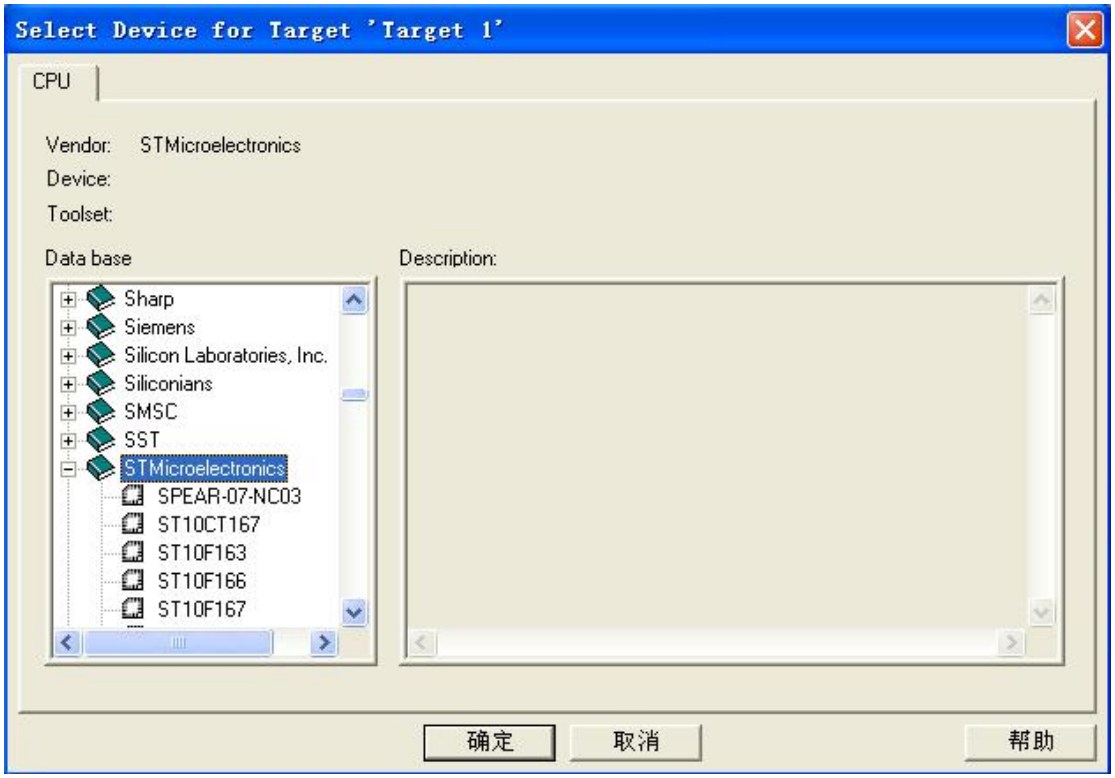
选择新建项目



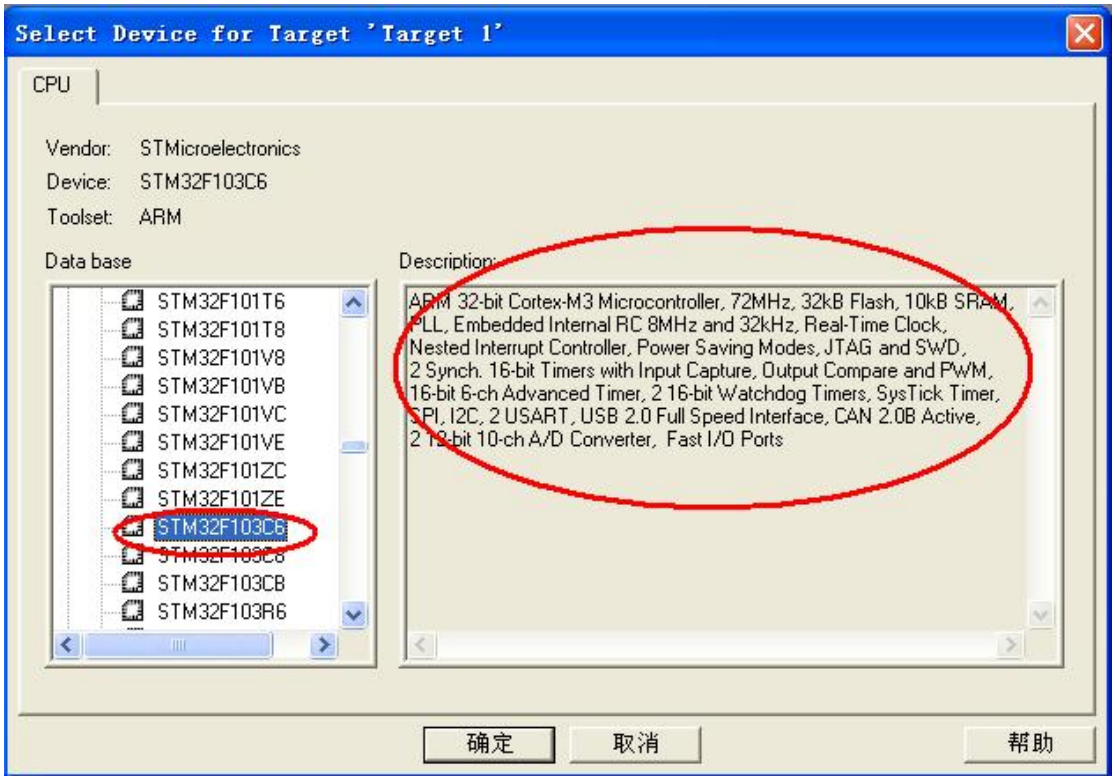
确认项目的名称、存放路径，这里我们给项目命名为 **Test1**



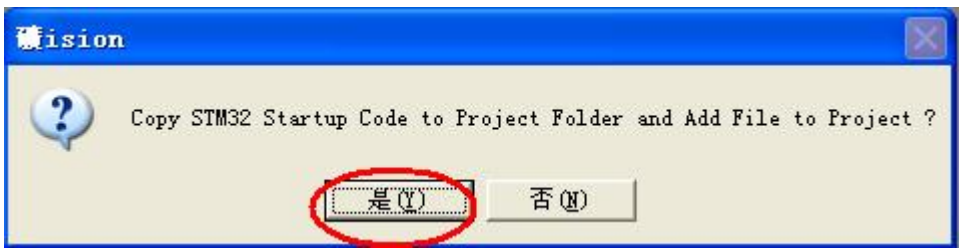
接下来选择芯片型号，我们选择 **EasySTM32** 上实际使用的芯片：**ST** 公司的 **STM32F103C6T6**



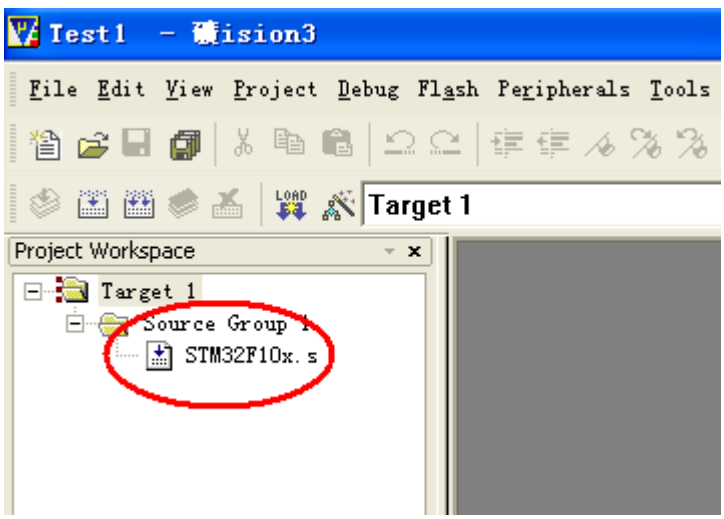
右边有 **STM32F103C6** 芯片的说明



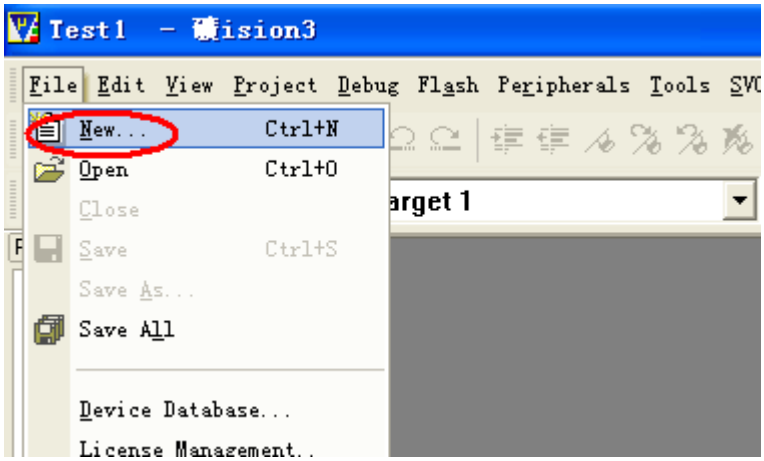
接下来会问是否 **Copy STM32** 的启动文件，当然选择 **OK** 了



可以看到工程已经建立，**STM32F10x.s** 是刚才加入的 **STM32** 启动文件



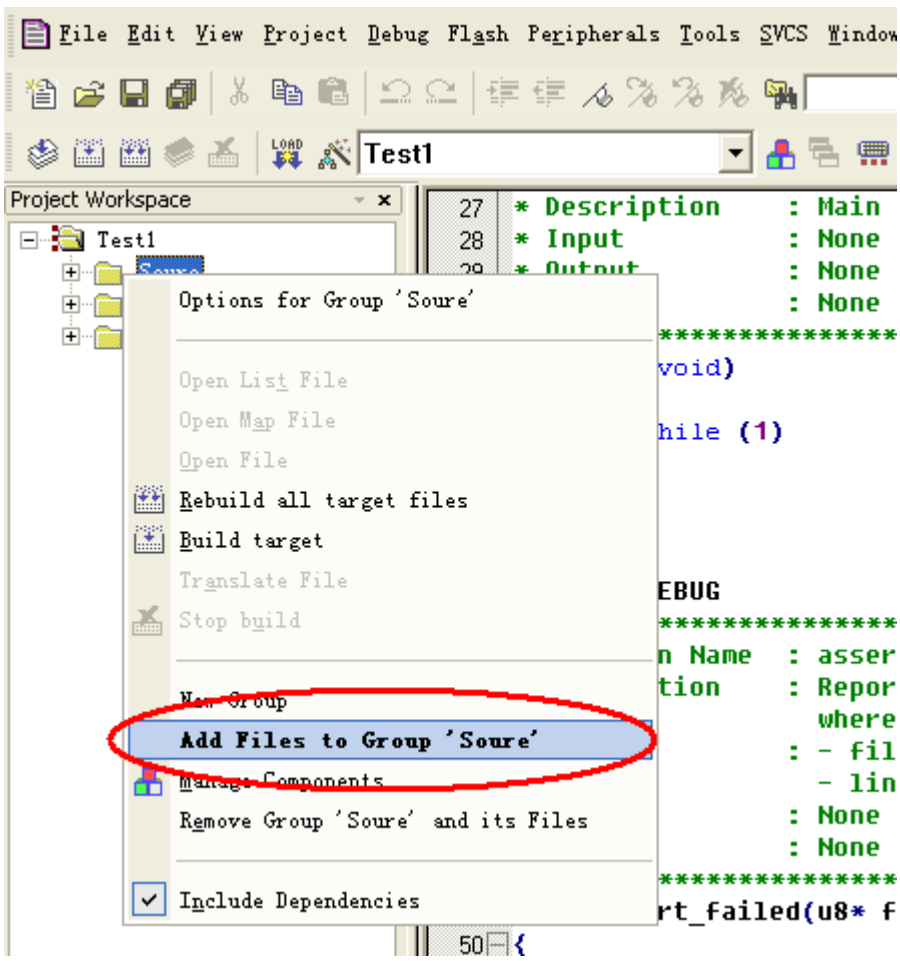
3 建立新文件



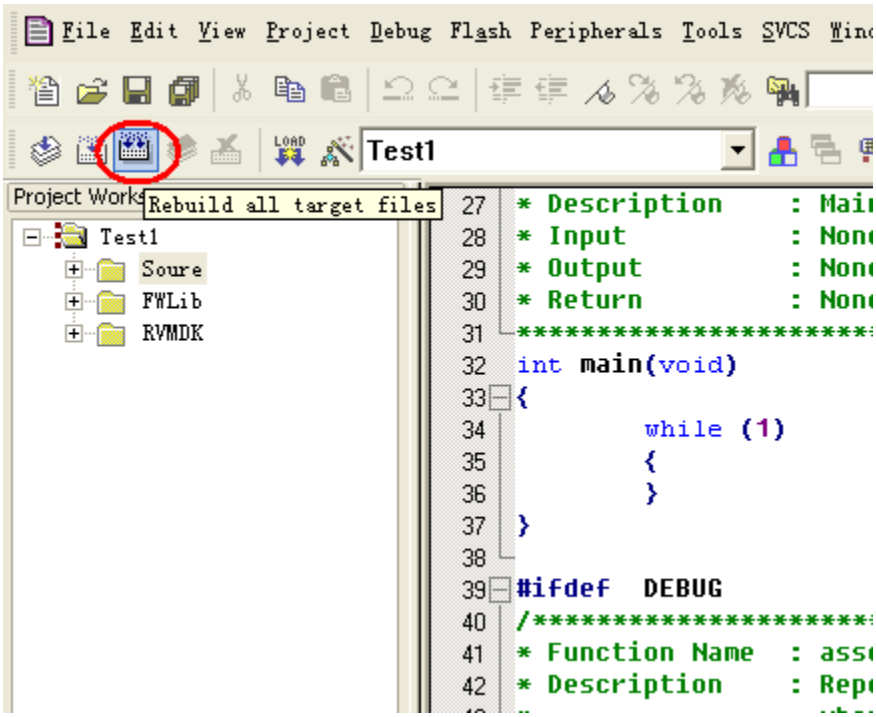
编写代码

```
int main(void)
{
        while (1)
        {
        }
}
```

4 工程中加入新建的文件



5 编译



STM32 芯片有一个最最重要的优点就是提供了外设的固件库，这样对外设的操作就十分简单，不需要了解寄存器。固件库和固件库的使用说明在 **ST** 网站上可以直接下载。

建议大家开发多使用外设库，这样可以保证编程的效率和正确性。只有在对时间或者效率要求非常高的地方才自行操作外设。

2 ISP 下载程序实验

没有仿真器下载仿真程序的情况下可以通过 **ISP** 来下载程序。

1 安装 **ST** 网站上的 **ISP** 下载的 **PC** 端程序（**13916** 是使用说明）

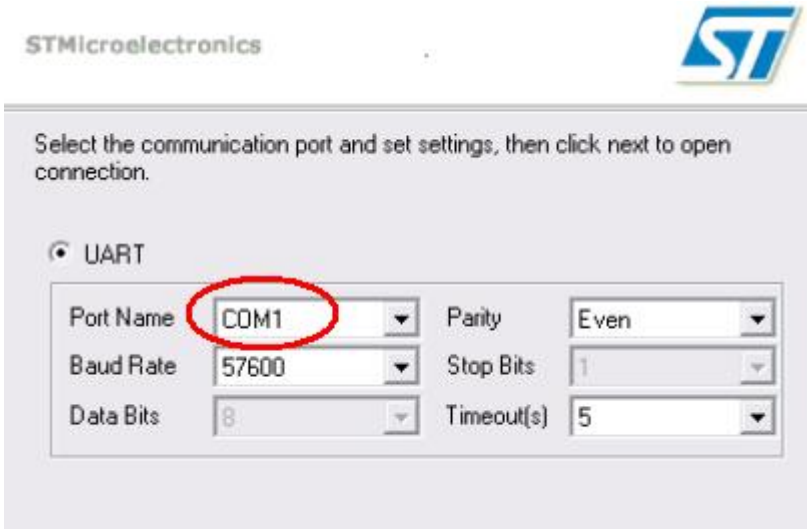
2 通过串口延长线连接好 **EasySTM32** 和 **PC**

3 连接好 **JP1**

4 拔下 **USB** 线，给 **EasySTM32** 断电

5 插入 **USB** 线，给 **EasySTM32** 上电，此时 **STM32** 即进入 **ISP** 模式

6 **PC** 上运行下载程序，选择好连接的串口



7 下载编译好的程序

8 还可以设置加密

要注意设置过读加密后，仿真器就不能仿真了，必须通过 **ISP** 程序清除加密才能仿真。

9 重新插拔 **USB** 上电，**STM32** 就开始运行刚才下载的程序。

3 跑马灯实验

跑马灯实验是我们在 **EasySTM32** 上第一个实验，也是我以前学习 **51** 做的第一个实验。虽然很简单，但是很有意思，看到 **4** 个 **Led** 能跑起来，我想大家也会觉得很有意思。通过跑马灯实验我们可以掌握 **STM32** 的 **GPIO** 做输出的应用，完整项目见附带项目文件 **Test3**。

这里要注意的 **STM32** 有许多外设，这些外设使用必须先使能。

RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);//使能外设时钟

还有一般都会对 **STM32** 的时钟做初始化，确定各部分的工作频率。

```
void RCC_Configuration(void)
{
	ErrorStatus HSEStartUpStatus;
	// RCC system reset(for debug purpose)
	RCC_DeInit();

	// Enable HSE
	RCC_HSEConfig(RCC_HSE_ON);

	// Wait till HSE is ready
	HSEStartUpStatus = RCC_WaitForHSEStartUp();

	if(HSEStartUpStatus == SUCCESS)
	{
		// HCLK = SYSCLK
		RCC_HCLKConfig(RCC_SYSCLK_Div1);

		// PCLK2 = HCLK
		RCC_PCLK2Config(RCC_HCLK_Div1);
```

```
// PCLK1 = HCLK/2
RCC_PCLK1Config(RCC_HCLK_Div2);

// Flash 2 wait state
//FLASH_SetLatency(FLASH_Latency_2);
FLASH_SetLatency(FLASH_Latency_0);
//FLASH_SetLatency(FLASH_Latency_1);


// Enable Prefetch Buffer
FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);

// PLLCLK = 8MHz * 2 = 16 MHz
RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_2);

// Enable PLL
RCC_PLLCmd(ENABLE);

// Wait till PLL is ready
while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET)
{
}

// Select PLL as system clock source
RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);

// Wait till PLL is used as system clock source
while(RCC_GetSYSCLKSource() != 0x08)
{
}
```

```
        }

        //              Enable              peripheral              clocks
------------------------------------------------
 }
```

4 UART 实验

UART 是最常用的外设之一了，这里我们直接使用外设库函数对 UART 操作。直接采用查询方式没有使用中断。实际使用中使用中断会有更高的 CPU 使用率。

```
#include "Include.h"

void UART1Init(void)
{
        GPIO_InitTypeDef GPIO_InitStructure;
        USART_InitTypeDef USART_InitStructure;

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
        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;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```
        USART_InitStructure.USART_BaudRate = 115200;
        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_Tx | USART_Mode_Rx;
        USART_Init(USART1, &USART_InitStructure);

        USART_Cmd(USART1, ENABLE);
}

void UART1SendByte(unsigned char SendData)
{
        USART_SendData(USART1,SendData);
        while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
}

unsigned char UART1GetByte(unsigned char* GetData)
{
        if(USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == RESET)
        {
                return 0;//没有收到数据
        }

        *GetData = USART_ReceiveData(USART1);

        return 1;//收到数据
}
```

```
void UART1Test(void)
{
	unsigned char i = 0;

	while(0)
	{
		while(UART1GetByte(&i))
		{
			UART1SendByte(i);
		}
	}

	while(1)
	{
		//UART1SendByte(i++);
		UART1SendByte(0x55);
		DelayNmS(200);
	}
}
```

5 按键实验

STM32 的 GPIO 可以设置为输入、输出、上拉、下拉等各种状态，这里我们设置成内部上拉输入，没有按键按下为 1，有按键按下为 0，通过读取 IO 值来判断按键。

```
#include "Include.h"

void KeyInit(void)
{
	GPIO_InitTypeDef GPIO_InitStructure;
```

```
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);//使能外设时钟
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);//使能外设时钟

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
	GPIO_Init(GPIOA, &GPIO_InitStructure);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
	GPIO_Init(GPIOB, &GPIO_InitStructure);
}

unsigned char GetKey(void)
{
	if(Bit_RESET == GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_2))
	{
		DelayNmS(10);//去抖动
		if(Bit_RESET == GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_2))
		{
			while(Bit_RESET     ==     GPIO_ReadInputDataBit(GPIOA,
GPIO_Pin_2))//等待按键释放
			{
			}
			return '1';//返回按键值
		}
	}

	if(Bit_RESET == GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_3))
```

```
	{
		DelayNmS(10);//去抖动
		if(Bit_RESET == GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_3))
		{
			while(Bit_RESET	==	GPIO_ReadInputDataBit(GPIOA,
GPIO_Pin_3))//等待按键释放
			{
			}
			return '2';//返回按键值
		}
	}

	if(Bit_RESET == GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_4))
	{
		DelayNmS(10);//去抖动
		if(Bit_RESET == GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_4))
		{
			while(Bit_RESET	==	GPIO_ReadInputDataBit(GPIOA,
GPIO_Pin_4))//等待按键释放
			{
			}
			return '3';//返回按键值
		}
	}

	if(Bit_RESET == GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_8))
	{
		DelayNmS(10);//去抖动
		if(Bit_RESET == GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_8))
		{
```

```
                        while(Bit_RESET     ==     GPIO_ReadInputDataBit(GPIOB,
GPIO_Pin_8))//等待按键释放
                        {
                        }
                        return '4';//返回按键值
                }
        }

        return NO_KEY;
}

void KeyTest(void)
{
        unsigned char i = 0;

        while(1)
        {
                i = GetKey();//按键扫描
                if(NO_KEY != i)
                {
                        LedOutput(0,1);//亮 LED 指示
                        UART1SendByte(i);//发送按键值
                        DelayNmS(100);
                        LedOutput(0,0);
                }
        }
}
```

6 SysTick 实验

以往的 MCU 都使用 Timer 来实现定时功能，而在 STM32 中 Timer 功能非常强大，除了定时外还有其他各种功能。STM32 中还有一个 SysTick 专门来做定时（这个时钟也是专门为操作系统准备的）。使用 SysTick 的中断来实现系统的计时功能。

STM32 中要使用中断非常简单：

1 对中断控制器初始化

```
void NVIC_Configuration(void)
{
        //NVIC_InitTypeDef NVIC_InitStructure;

#ifdef  VECT_TAB_RAM
        /* Set the Vector Table base location at 0x20000000 */
        NVIC_SetVectorTable(NVIC_VectTab_RAM, 0x0);
#else  /* VECT_TAB_FLASH  */
        /* Set the Vector Table base location at 0x08000000 */
        NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
#endif
}
```

2 正确设置外设的中断控制

```
void SysTInit(void)
{
        SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);//2M
        SysTick_SetReload(200000);//100ms

        SysTick_ITConfig(ENABLE);

        NVIC_SystemHandlerPriorityConfig(SystemHandler_SysTick, 2, 0);
```

```
        SysTStop();
}
```

3 添加中断程序

可以直接修改 stm32f10x_it.c 文件

4 打开中断，如果有触发事件就可以进入触发中断

该实验中 SysTick 中断会改变 Led1 的状态，例程中中断较快，肉眼看不清 Led 变化。通过修改程序可以改变 Led 闪烁的频率。

7 PWM 实验

STM32 的 Timer 有一个功能就是 PWM 输出，这里我们通过 PWM 直接驱动无源蜂鸣器，让蜂鸣器发出不同音调的声音。

```
#include "Include.h"

void Timer2Init(void)
{
        TIM_OCInitTypeDef TIM_OCInitStructure;
        TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;

        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

        TIM_TimeBaseInitStructure.TIM_Period = 10000;
        TIM_TimeBaseInitStructure.TIM_Prescaler = 0;
        TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
        TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
        TIM_TimeBaseInit(TIM2,&TIM_TimeBaseInitStructure);

        TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
        TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
        TIM_OCInitStructure.TIM_Pulse = 5000;
```

```
        TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
        TIM_OC2Init(TIM2, &TIM_OCInitStructure);

}

void Timer2OutEnable(void)
{
        GPIO_InitTypeDef GPIO_InitStructure;

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_Init(GPIOA, &GPIO_InitStructure);
}

void Timer2OutDisable(void)
{
        GPIO_InitTypeDef GPIO_InitStructure;

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
        GPIO_Init(GPIOA, &GPIO_InitStructure);
}

void Timer2Start(void)
```

```
{
        TIM_Cmd(TIM2, ENABLE);
}

void Timer2Stop(void)
{
        TIM_Cmd(TIM2, DISABLE);
}

void Timer2Test(void)
{
        Timer2Start();

        while(1)
        {
                Timer2OutEnable();
                DelayNmS(200);
                Timer2OutDisable();
                DelayNmS(200);
        }
}
```

8 音乐播放实验

在上个实验的基础上我们可以通过各种音调的组合播放简单的歌曲。

```
//音调表
//C 523
//D 587
//E 659
//F 698
```

```
//G 784
//A 880
//B 988

const unsigned int MusicData[7] = {30592,27257,24297,22922,20408,18182,16194};
void Timer2Test(void)
{
        unsigned char i = 0;

        Timer2Start();

        //一句简单的歌曲
        while(1)
        {
                Timer2OutEnable();
                SetBellF(MusicData[0]);
                DelayNmS(400);
                SetBellF(MusicData[0]);
                DelayNmS(400);
                SetBellF(MusicData[4]);
                DelayNmS(400);
                SetBellF(MusicData[4]);
                DelayNmS(400);
                SetBellF(MusicData[5]);
                DelayNmS(400);
                SetBellF(MusicData[5]);
                DelayNmS(400);
                SetBellF(MusicData[4]);
                DelayNmS(400);
                DelayNmS(400);
```

```
        SetBellF(MusicData[3]);
        DelayNmS(400);
        SetBellF(MusicData[3]);
        DelayNmS(400);
        SetBellF(MusicData[2]);
        DelayNmS(400);
        SetBellF(MusicData[2]);
        DelayNmS(400);
        SetBellF(MusicData[1]);
        DelayNmS(400);
        SetBellF(MusicData[1]);
        DelayNmS(400);
        SetBellF(MusicData[0]);
        DelayNmS(400);
        DelayNmS(400);
        Timer2OutDisable();
        DelayNmS(1000);
}

//从 C 播放到 B
while(1)
{
        SetBellF(MusicData[i]);
        Timer2OutEnable();
        DelayNmS(200);
        i++;
        if(i>=7)
        {
                i = 0;
                Timer2OutDisable();
```

```
                DelayNmS(2000);
            }
        }
}
```

9 AD 实验

连接好串口，在 PC 上通过串口调试工具可以看到 AD 转化的值。

STM32 的 AD 非常强大，有许多用法。本例程中只实现了和以前的 MCU 类似的查询结果方式。如果想深入研究 STM32 的 AD，请参考 ST 网上的资料。

```
#include "Include.h"

void ADInit(void)
{
        GPIO_InitTypeDef GPIO_InitStructure;

        ADC_InitTypeDef ADC_InitStructure;

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);//使能外设时钟
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);//使能外设时钟

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
        GPIO_Init(GPIOB, &GPIO_InitStructure);

        ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
        ADC_InitStructure.ADC_ScanConvMode = DISABLE;
        ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
        ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
```

```
        ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Left;
        ADC_InitStructure.ADC_NbrOfChannel = 1;
        ADC_Init(ADC1, &ADC_InitStructure);

        ADC_Cmd(ADC1, ENABLE);

        ADC_RegularChannelConfig(ADC1,          ADC_Channel_9,          1,
ADC_SampleTime_13Cycles5);

        ADC_ResetCalibration(ADC1);
        /* Check the end of ADC1 reset calibration register */
        while(ADC_GetResetCalibrationStatus(ADC1));

        /* Start ADC1 calibaration */
        ADC_StartCalibration(ADC1);
        /* Check the end of ADC1 calibration */
        while(ADC_GetCalibrationStatus(ADC1));
}

unsigned char GetADV(void)
{
        unsigned int DataValue;

        ADC_SoftwareStartConvCmd(ADC1, ENABLE);

        while(RESET == ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));

        DataValue = ADC_GetConversionValue(ADC1);

        return (DataValue>>8);
```

```
}

void ADTest(void)
{
        while(1)
        {
                UART1SendByte(GetADV());
                DelayNmS(200);
        }
}
```

10 SPI 实验

74HC595 是很常用的串行输入转并行输出的芯片。我们这里用来驱动一个数码管。做该实验可以看到数码管 0~9 的循环显示。

```
#include "Include.h"

#define SET_595CS {GPIO_SetBits(GPIOB, GPIO_Pin_5);}
#define CLR_595CS {GPIO_ResetBits(GPIOB, GPIO_Pin_5);}

const unsigned char DisplayData[] =
{
        0xFA,0xC0,0x76,0xF4,0xCC,0xBC,0xBE,0xD0,0xFE,0xFC,0
};

void Led8IOInit(void)
{
        GPIO_InitTypeDef GPIO_InitStructure;
```

```
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_Init(GPIOB, &GPIO_InitStructure);

	SET_595CS
}

void SPIInit(void)
{
	GPIO_InitTypeDef GPIO_InitStructure;
	SPI_InitTypeDef SPI_InitStructure;

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);//使能 SPI1 时钟
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
	GPIO_Init(GPIOA, &GPIO_InitStructure);

	SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
	SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
	SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
	SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
	SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
	SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
	SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
```

```
        SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
        SPI_InitStructure.SPI_CRCPolynomial = 7;
        SPI_Init(SPI1, &SPI_InitStructure);

        SPI_Cmd(SPI1, ENABLE);//使能 SPI1

        Led8IOInit();

        DisplayNumber(0xFF);
}

unsigned char SPI1Action(unsigned char SendData)
{
        while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)==RESET);
        SPI_I2S_SendData(SPI1,SendData);
        while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE)==RESET);
        return(SPI_I2S_ReceiveData(SPI1));
}

void DisplayNumber(unsigned char Number)
{
        if(Number>10)
        {
                Number = 10;
        }

        CLR_595CS
        SPI1Action(DisplayData[Number]);
        SET_595CS
}
```

```
void SPITest(void)
{
        unsigned char i = 0;

        Led8IOInit();

        while(1)
        {
                DisplayNumber(i);
                i++;
                if(i>10)
                {
                        i = 0;
                }
                DelayNmS(200);
        }

        while(1)
        {
                CLR_595CS
                SPI1Action(0x55);
                SET_595CS
                DelayNmS(200);
        }
}
```

11 RTC 实验

STM32 内部包含一个 RTC、还有一个靠外部供电的 RAM 区域。

这个 RTC 只是不停的走动，没有年、月、日等寄存器，需要自己写程序转化。下面有例程会实现这一部分功能。

```
void RTCInit(void)
{
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR | RCC_APB1Periph_BKP, ENABLE);
	PWR_BackupAccessCmd(ENABLE);

	if(BKP_ReadBackupRegister(BKP_DR1) != 0xA5A5)
	{
		RCC_LSEConfig(RCC_LSE_ON);
		while(RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET)
		{
		}
		RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);

		RCC_RTCCLKCmd(ENABLE);

		//Output
		BKP_TamperPinCmd(DISABLE);
		//BKP_RTCOutputConfig(ENABLE);
		BKP_RTCOutputConfig(DISABLE);

		RTC_WaitForSynchro();
		RTC_WaitForLastTask();

		//设置时间
		RTC_SetCounter(0);

		BKP_WriteBackupRegister(BKP_DR1, 0xA5A5);
	}
```

```
}

void RTCTest(void)
{
        unsigned long RTCCounterValue;

        while(1)
        {
                RTCCounterValue = RTC_GetCounter();

                UART1SendByte(RTCCounterValue>>24);
                UART1SendByte(RTCCounterValue>>16);
                UART1SendByte(RTCCounterValue>>8);
                UART1SendByte(RTCCounterValue);

                DelayNmS(1000);
        }
}
```

12 I2C 实验

I2C 读写 E2PROM 实验是在参考 21ic 上 ST 论坛上的帖子：

http://bbs.21ic.com/club/bbs/bbsView.asp?boardid=49

这里向作者 lut1lut 敬礼！

13 USB 虚拟串口实验

USB 虚拟串口实验是直接在 ST 的例子上修改了，仅仅改了控制 USB 上拉电阻的 IO 位置。先安装好 ST 提供的 USB 驱动。连接好串口、USB 口，选择驱动，我们可以看到 PC 上多了个串口。这个串口就是 STM32 的 USB 虚拟出来的。

STM32 提供了完善的 USB 例程，我们可以很方便的实现自己想要的功能。大家可以参考 ST 网上的说明。

五 趣味实验

1 播放简单歌曲实验

只要任意按一个按键就可以听到 SuperStar 的音乐。

在第 8 个实验的基础上又跟进了一步。编写了一个输入数组，播放歌曲的函数。只要安装自定义格式编写数组，就能放出歌曲来。

2 实时时钟实验

这里实现了通过程序把 RTC 的计数值转化成年、月、日、时、分、秒、星期。该程序使用简易算法只能在 2000~2099 年有效。