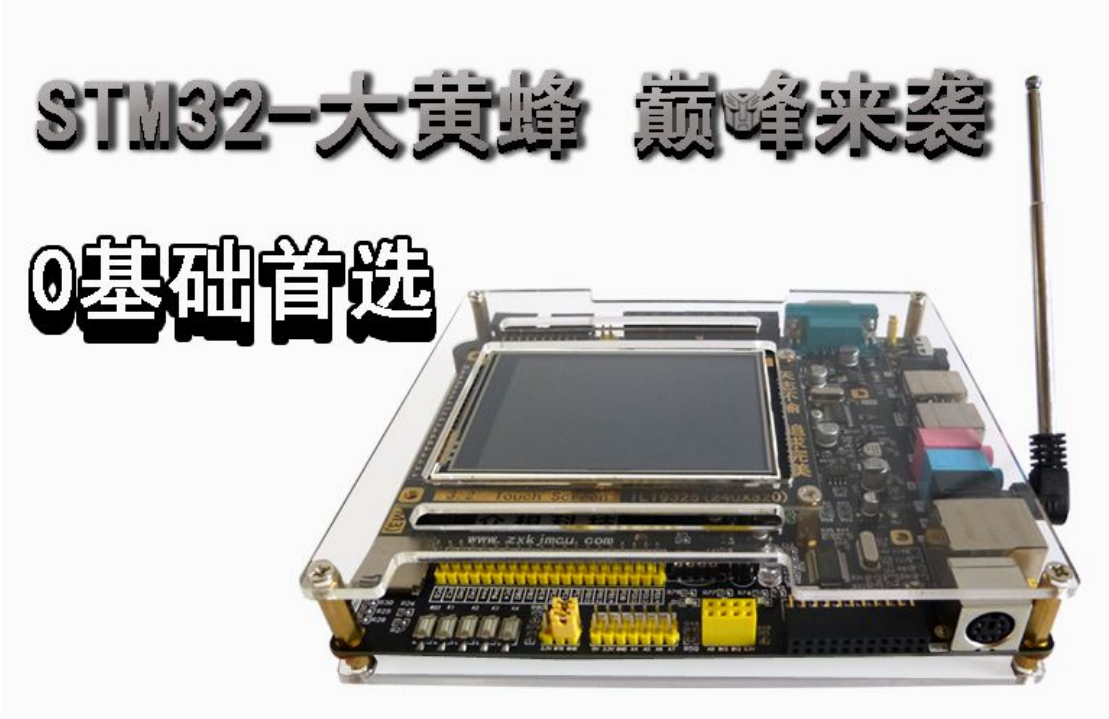
学 ARM 从 STM32 开始

STM32 开发板库函数教程--模块篇



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

目 录

5.05.1 概述.....3
5.05.1.1 矩阵键盘原理.....3
5.05.2 实验目的.....5
5.05.3 硬件设计.....5
5.05.4 软件设计.....6
5.05.5 STM32 系统时钟配置 SystemInit().....9
5.05.6 GPIO 引脚时钟使能.....9
5.05.7 GPIO 管脚电平控制函数.....9
5.05.8 stm32f10x_it.c 文件里的内容是.....10
5.05.9 main.c 文件里的内容是.....10
5.05.10 程序下载.....17
5.05.11 实验效果图.....18

5.05 读取 4×4 矩阵键盘程序设计

5.05.1 概述

4×4 矩阵键盘是运用得最多的键盘形式，也是单片机入门必需掌握的一种键盘识别技术，4×4 矩阵键盘是使用用 8 个 I/O 口就可以对 16 个按键进行识别，节省了 I/O 口资源。

5.05.1.1 矩阵键盘原理

矩阵式键盘由行线和列线组成，按键位于行、列的交叉点上。当键被按下时，其交点的行线和列线接通，相应的行线或列线上的电平发生变化，单片机通过检测行或列线上的电平变化可以确定哪个按键被按下。

矩阵键盘不仅在连接上比单独式按键复杂，它的按键识别方法也比单独式按键复杂。概括讲矩阵键盘的检测方法有多种，常见的有：逐点扫描法、逐行扫描法、全局扫描法。

在本实例中我们采用逐行扫描法来实现按键检测，其中 PD9、PD11、PD13、PD15 作为列线，PE15、PB11、PB13、PB15 作为行线。识别过程如下：

1、判断键盘中是否有键按下。设置所有行线为输出口，并输出低电平；设置列线为输入口，读取列线上的电平状态，只要有一列的电平为低，就表示有按键按下，并且被按下的键位于电平为低的列线与 4 跟行线相交叉的 4 个按键中，若所有列线都为高电平，表示没有按键按下；

2、判断被按下按键所在的位置。在确认有键按下后（进行按键消抖处理后），接下来就是确定具体哪个按键被按下，方法是：依次将每根行线设

置为输出口，并输出低电平（同时剩余行线输出高电平），然后逐列检查每根列线的电平状态，若某列为低电平，则该列线与设置为输出低电平的行线交叉处的按键就是被按下的按键。

3、按键位置确定后，接下来就要给矩阵键盘中的每个按键进行编号，也就是进行按键编码，程序设计中常用计算法和查表法两种方式对按键进行编码。

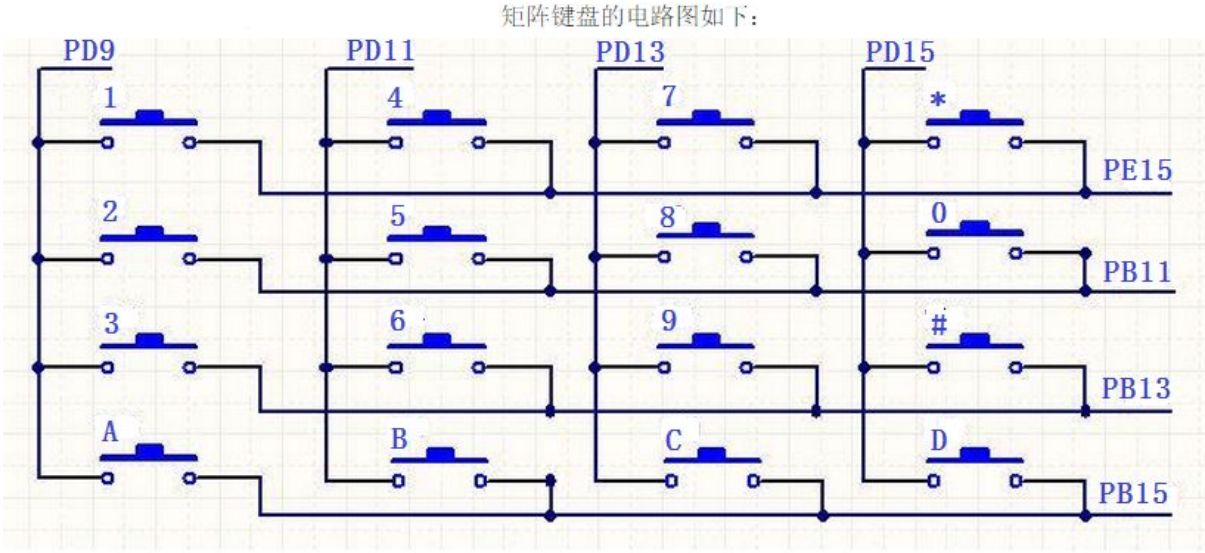


图 5.05.1　矩阵键盘原理图

对于 AVR 单片机来说，我们已经知道在 I/O 口输入状态下，可以使能其内部上拉电阻，所以上面电路图 5.05.1　矩阵键盘原理图中连接 4 根列线的上拉电阻可以不用，直接使能内部上拉电阻即可。

我们选用的是成品键膜。通过一组标准的杜邦连接器直接和实验板连接，非常方便，减少了连线的麻烦。如右图所示。

5.05.2 实验目的

通过扩展的矩阵键盘，实现我们键盘键值判断和读取的程序设计。为了更直观的观察键值是否正确读出来，我们把测量结果输出打印至计算机显示。

5.05.3 硬件设计

选用大黄蜂实验板，把矩阵键盘直接插入到实验板下方双排针最后 8 位，对应的端口是：PE15、PB11、PB13、PB15、PD9、PD11、PD13、PD15。硬件设计见图 5.05.2　矩阵键盘连线图。

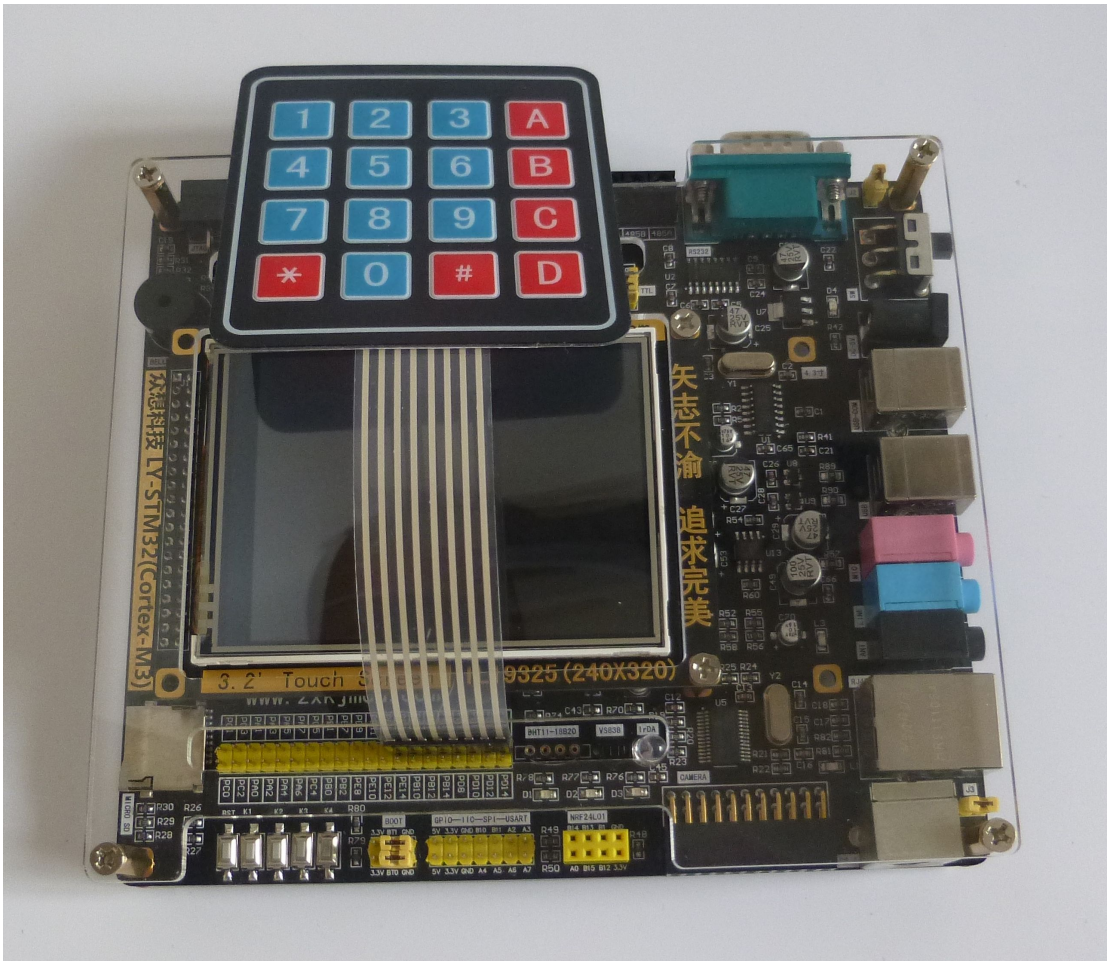


图 5.05.2　矩阵键盘连线图

附表 1 接线方法

STM32 开发板端子	矩阵键盘行列线分布
PE15	行 00
PB11	行 01
PB13	行 02
PB15	行 03
PD9	列 00
PD11	列 01
PD13	列 02
PD15	列 03

5.05.4 软件设计

5.05.4.1 软件设计说明

本实例采用全局扫描法，扫描矩阵键盘行列状态来判断那个键按下。我们还是采用编写库函数的方式进行程序设计。

在这节程序设计中，用到了外部中断函数；prinif 重定向打印输出函数； USART 串口通讯函数；定时器函数。

5.05.4.2 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。

5.05.4.3 自定义头文件

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

我们已经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。

5.05.4.4 pbdata.h 文件里的内容是

```
#ifndef _pbdata_H
#define _pbdata_H  //宏定义

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

//定义函数
void delay_us(u32 nus);//微妙延时程序函数
void delay_ms(u16 nms);//毫妙延时程序函数

#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。

5.05.4.5 pbdata.c 文件里的内容是

下面是 pbdata.c 文件详细内容，在文件开始还是引用“pbdata.h”文件。

```
#include "pbdata.h"
```

```
/*******************************************************************************
* 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)
{
     //注意 delay_ms 函数输入范围是 1-1863
    //所以最大延时为 1.8 秒

    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
}
```

5.05.5 STM32 系统时钟配置 SystemInit()

我们总在强调，每个工程都必须在开始时配置并启动 STM32 系统时钟，这次也不例外。

5.05.6 GPIO 引脚时钟使能

```
SystemInit();//72m
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);//设置 PA 输出端口
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);//设置 PB 输出端口
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD,ENABLE);//设置 PD 输出端口
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);//设置 PE 输出端口
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\PB\PD\PE 端口，所以要把 PA\PB\PD\PE 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。

5.05.7 GPIO 管脚电平控制函数

在主程序中采用 while(1)循环语句，等待外部中断的到来后，主程序中读取缓冲区的温度值，并就打印输出到屏幕。

```
while(1)
{
  delay_ms(10);
  KEY_Scanf();
}
```

5.05.8 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。

```
/* Includes
------------------------------------------------------------------*/
#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);
   }
}
```

5.05.9 main.c 文件里的内容是

键盘处理程序在主程序中初始化完成，利用内部上拉，没有键盘按下时，列总处于高电平状态，当有键盘被按下时，对应列的输入端口会被拉低，其余的还是高电平，这样就能判断出那个位置的键盘被按下。

矩阵键盘中的列总处于读取状态。

```
#include "pbdata.h"

void RCC_Configuration(void);//声明
```

```
    void GPIO_Configuration(void);//
    void NVIC_Configuration(void);//
    void USART_Configuration(void);

    #define Hang_00_L  GPIO_ResetBits(GPIOE, GPIO_Pin_15)//行 00
    #define Hang_00_H  GPIO_SetBits(GPIOE, GPIO_Pin_15)

    #define Hang_01_L  GPIO_ResetBits(GPIOB, GPIO_Pin_11)//行 01
    #define Hang_01_H  GPIO_SetBits(GPIOB, GPIO_Pin_11)

    #define Hang_02_L  GPIO_ResetBits(GPIOB, GPIO_Pin_13)//行 02
    #define Hang_02_H  GPIO_SetBits(GPIOB, GPIO_Pin_13)

    #define Hang_03_L  GPIO_ResetBits(GPIOB, GPIO_Pin_15)//行 03
    #define Hang_03_H  GPIO_SetBits(GPIOB, GPIO_Pin_15)

    #define Lie_00_V GPIO_ReadInputDataBit(GPIOD,GPIO_Pin_9)//列 00
    #define Lie_01_V GPIO_ReadInputDataBit(GPIOD,GPIO_Pin_11)//列 01
    #define Lie_02_V GPIO_ReadInputDataBit(GPIOD,GPIO_Pin_13)//列 02
    #define Lie_03_V GPIO_ReadInputDataBit(GPIOD,GPIO_Pin_15)//列 03

    #define  jpys 20   //宏定义，约定 jpys==20，方便以后程序移植

    void KEY_Scanf(void)
    {
       //第一行
       Hang_00_L;//把第一行输出低电平
       Hang_01_H;
       Hang_02_H;
       Hang_03_H;

       if(Lie_00_V==0)
       {
          delay_ms(jpys); //延时 20 秒，软件消抖
          if(Lie_00_V==0) //如果第一列是低电平，说明有键被按下，如果没有直接
退出 if 语句
          {
              printf("1\r\n"); //打印输出“1”
          }
       }

       if(Lie_01_V==0)//如果第二列是低电平，
       {
          delay_ms(jpys);//延时 20 秒，软件消抖
```

```
        if(Lie_01_V==0)//如果第二列是低电平，说明有键被按下，如果没有直接退出 if 语句
        {
            printf("2\r\n"); //打印输出“2”
        }
    }

    if(Lie_02_V==0)
    {
        delay_ms(jpys);
        if(Lie_02_V==0)
        {
            printf("3\r\n");
        }
    }

    if(Lie_03_V==0)//如果第四列是低电平
    {
        delay_ms(jpys);
        if(Lie_03_V==0)//如果第四列是低电平，说明有键被按下，如果没有直接退出 if 语句
        {
            printf("A\r\n");//打印输出“A”
        }
    }

    //第二行
    Hang_00_H;
    Hang_01_L;//把第二行拉低
    Hang_02_H;
    Hang_03_H;

    if(Lie_00_V==0)//如果第一列是低电平
    {
        delay_ms(jpys);
        if(Lie_00_V==0)//说明有键被按下，如果没有直接退出 if 语句
        {
            printf("4\r\n"); //打印输出“4”
        }
    }

    if(Lie_01_V==0)
    {
        delay_ms(jpys);
```

```
        if(Lie_01_V==0)
        {
            printf("5\r\n");
        }
    }

    if(Lie_02_V==0)
    {
        delay_ms(jpys);
        if(Lie_02_V==0)
        {
            printf("6\r\n");
        }
    }

    if(Lie_03_V==0)
    {
        delay_ms(jpys);
        if(Lie_03_V==0)
        {
            printf("B\r\n");
        }
    }

    //第三行
    Hang_00_H;
    Hang_01_H;
    Hang_02_L;//把第三行置低
    Hang_03_H;

    if(Lie_00_V==0) //如果第一列是低电平
    {
        delay_ms(jpys);//延时 20 秒
        if(Lie_00_V==0)//说明有键被按下，如果没有直接退出 if 语句
        {
            printf("7\r\n"); //打印输出“7”
        }
    }

    if(Lie_01_V==0)
    {
        delay_ms(jpys);
        if(Lie_01_V==0)
        {
```

```
            printf("8\r\n");
        }
    }

    if(Lie_02_V==0)
    {
        delay_ms(jpys);
        if(Lie_02_V==0)
        {
            printf("9\r\n");
        }
    }

    if(Lie_03_V==0)
    {
        delay_ms(jpys);
        if(Lie_03_V==0)
        {
            printf("C\r\n");
        }
    }

    //第四行
    Hang_00_H;
    Hang_01_H;
    Hang_02_H;
    Hang_03_L;//把第四行置低

    if(Lie_00_V==0)//如果第一列是低电平
    {
        delay_ms(jpys);
        if(Lie_00_V==0)//说明有键被按下，如果没有直接退出 if 语句
        {
            printf("*\r\n"); //打印输出“*”
        }
    }

    if(Lie_01_V==0)
    {
        delay_ms(10);
        if(Lie_01_V==0)
        {
            printf("0\r\n");
        }
```

```
    }

    if(Lie_02_V==0)//如果第三列是低电平
    {
       delay_ms(jpys);
       if(Lie_02_V==0)//说明有键被按下，如果没有直接退出 if 语句
       {
           printf("#\r\n"); //打印输出“#”
       }
    }

    if(Lie_03_V==0)
    {
       delay_ms(jpys);
       if(Lie_03_V==0)
       {
           printf("D\r\n");
       }
    }
}

int fputc(int ch,FILE *f)//固定格式
{
    USART_SendData(USART1,(u8)ch);
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    return ch;
}

int main(void)
{
    double temp=0;//定义 double 类型的中间变量 temp，并初始化
    RCC_Configuration(); //系统时钟初始化
    GPIO_Configuration();//端口初始化
    USART_Configuration();
    NVIC_Configuration();

    while(1)
    {
      delay_ms(10);//延时 10ms
      KEY_Scanf();//扫描键盘处理程序
    }
}

void RCC_Configuration(void)
```

```
    {
        SystemInit();//72m
       RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
       RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
       RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD,ENABLE);
       RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);
       RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
       RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
    }

    void GPIO_Configuration(void)
    {
        GPIO_InitTypeDef GPIO_InitStructure;

       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);

       //键盘输出
       GPIO_InitStructure.GPIO_Pin=GPIO_Pin_11|GPIO_Pin_13|GPIO_Pin_15;
       GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
       GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
       GPIO_Init(GPIOB,&GPIO_InitStructure);

       GPIO_InitStructure.GPIO_Pin=GPIO_Pin_15;
       GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
       GPIO_Init(GPIOE,&GPIO_InitStructure);
       //键盘输入
       GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9|GPIO_Pin_11|GPIO_Pin_13|GPIO_P
in_15;
       GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU;
       GPIO_Init(GPIOD,&GPIO_InitStructure);
    }

    void NVIC_Configuration(void)
    {
          NVIC_InitTypeDef NVIC_InitStructure;

       NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
```

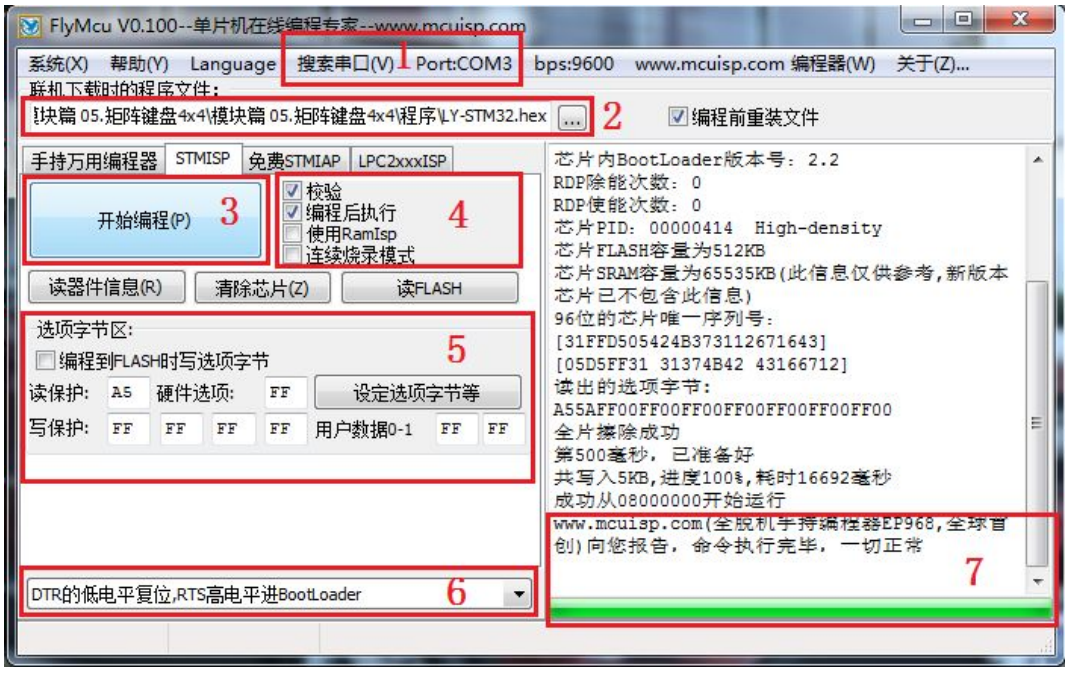
```
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

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_HardwareFlowContr
ol_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);
}
```

5.05.10 程序下载

在这一章节中要掌握矩阵键盘工作方式和扫描过程，做到活学活用。掌握键盘扫描的程序设计结构，要求熟练掌握。

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 COM3。（2）号区域请自己选择程序代码所在的文件夹。（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\2,外设篇\模块篇 05.矩阵键盘 4x4\程序）

5.05.11 实验效果图

把矩阵键盘和开发板连接好，送电下载程序，如“图 5.05.11.1 矩阵键盘输入实验效果图”，打开众想科技多功能监控软件，点击串口调试，当有按键按下的时候，我们在接收区可以观察到实际按下的键盘值。按下键值“1”，在接收区就会打印输出键值“1”。说明我们矩阵键盘工作正常，程序编写很成功。

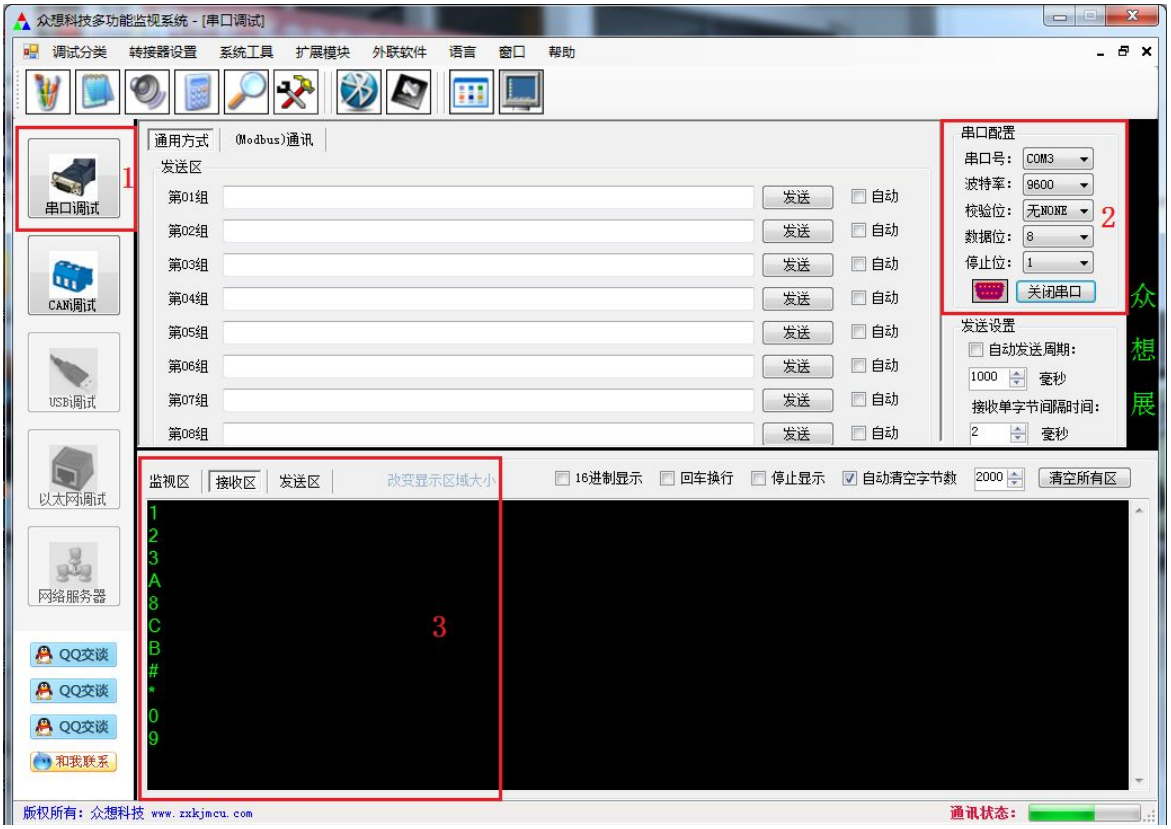


图 5.05.11.1 矩阵键盘输入实验效果图