实验十一 外接组件扩展实验

本实验是对本书内容的综合扩展应用，从实际应用的角度来介绍硬件扩展平台外接组件的使用，给出了各外接组件的基本工作原理、电路原理以及编程实践。首先介绍了各种被控单元（传感器）的原理、电路接法和编程实践，然后结合“照葫芦画瓢”的框架外接被控单元（传感器），实现综合运用，以达到熟练掌握的目的。

11.1 开关量输出类驱动构件

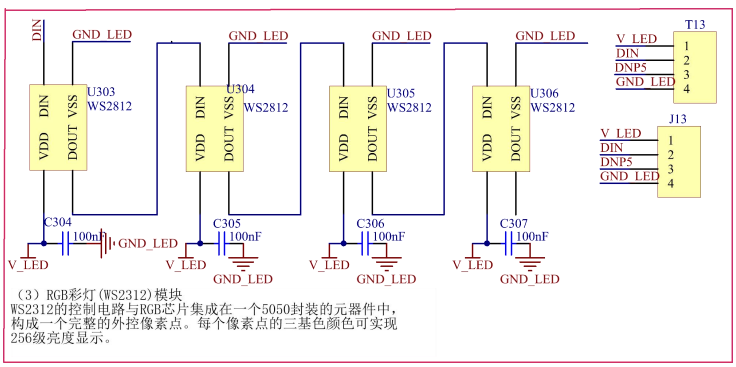
11.1.1 彩灯

1．原理概述

彩灯的控制电路与 RGB 芯片集成在一个 5050 封装的元器件中，构成了一个完整的外控像素点，每个像素点的三基色颜色可实现 256 级亮度显示。像素点内部包含了智能数字接口数据锁存信号整形放大驱动电路、高精度的内部振荡器和可编程定电流控制部分，有效保证了像素点光的颜色高度一致，数据协议采用单线归零码的通讯方式，通过发送具有特定占空比的高电平和低电平来控制彩灯的亮暗。

2．电路原理

彩灯的电路原理图如图 6-5（a）所示，其实物图如图 6-5（b）所示。



（a）彩灯电路原理图　　（b）彩灯实物图

图 6-5 彩灯

VDD 是电源端，用于供电，DOUT 是数据输出端，用于控制数据信号输出，VSS 用于信号接地和电源接地，DIN 控制数据信号的输入。彩灯使用串行级联接口，能够通过一根信号线完成数据的接收与解码。使用 USB 数据线一端连接 J5 口（GPIO 接口），另一端连接到彩灯。

3．编程实践

程序可参考“.. \04-Soft\ch11-1\User_ColorLight”工程，其编程步骤如下：

1）准备阶段

（1）拷贝 User 工程并重命名

拷贝“..\04-Soft \ch03”下的 User_Frame 工程，并重命名为 User_ColorLight。

（2）添加彩灯传感器应用构件

将“...\04-Soft\ch03\driver_component\ws2812”下的 ws2812.c 和 ws2812.h 应用构件拷贝到.. \User_ColorLight \05_UserBoard 下。

（3）添加彩灯应用构件头文件和彩灯引脚宏定义

在 05_UserBoard \user.h 中添加彩灯应用构件头文件（ws2812.h），查看 04_GEC\gec.h 文件，找到彩灯所接具有 GPIO 功能的引脚，并在 user.h 中添加彩灯宏定义（如宏名为 COLORLIGHT）。

（4）定义彩灯相关参数

在 main.c 的数据段中定义彩灯提示信息以及初始颜色值。

```
//（1.1）声明main函数使用的局部变量
//彩灯测试数据（一种颜色占3个字节，按grb顺序）
uint_8 grbw[12]={0xFF,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF};
uint_8 rwgb[12]={0x00,0xFF,0x00,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0xFF};
uint_8 black[12]={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
```

2）应用阶段

在 main.c 文件中对彩灯传感器进行初始化，并设置引脚方向为输出，初始状态为低电平以及设置彩灯颜色变换。

```
//（1.5）用户外设模块初始化
    printf("点亮彩灯\n\0");
    WS_Init(COLORLIGHT);
    WS_SendOnePix(COLORLIGHT,grbw,4);
    Delay_ms(2000);
    printf("熄灭彩灯\n\0");
    WS_SendOnePix(COLORLIGHT,black,4);
    Delay_ms(2000);
    printf("改变彩灯颜色\n\0");
    WS_SendOnePix(COLORLIGHT,rwgb,4);
    Delay_ms(2000);
//（2）=======主循环部分（结尾）=============================================
...
```

4．运行结果

彩灯的运行效果如图 6-6 所示。

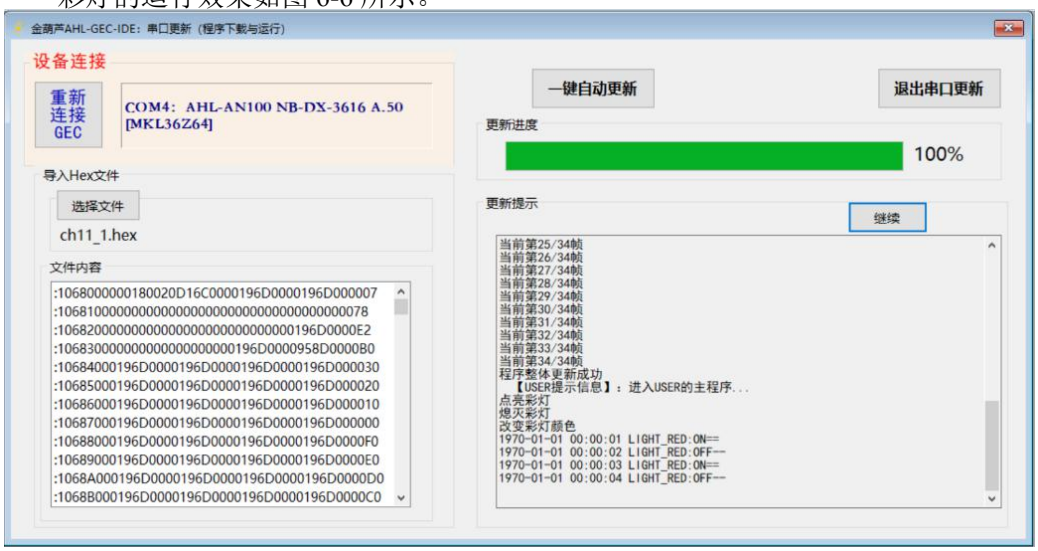


图 6-6 彩灯的运行结果

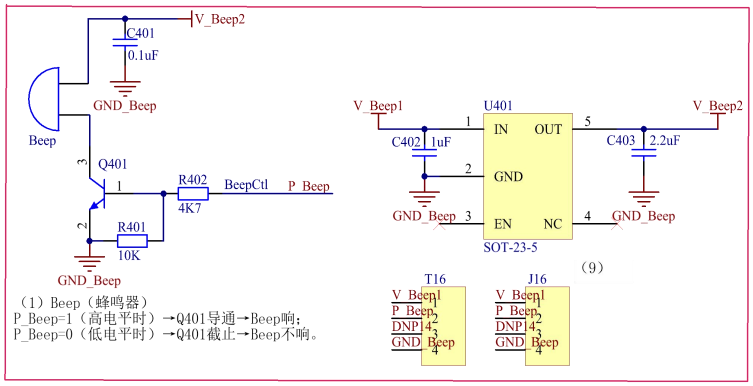
11.1.2 蜂鸣器

1．原理概述

蜂鸣器输出端电平设置为高电平，蜂鸣器发出声响；输出端电平设置为低电平，蜂鸣器不发出声响或停止发出声响。蜂鸣器初始化默认是低电平，不发出声响。

2．电路原理

蜂鸣器的电路原理图如图 6-7（a）所示，其实物图如图 6-7（b）所示。蜂鸣器通过 P_Beep 引脚来控制输出引脚的高低电平。当 P_Beep 对应的状态值为 1 即高电平时，Q401 导通，蜂鸣器发出声响；反之，当 P_Beep 对应的状态值为 0 即低电平时，Q401 截止，蜂鸣器不发出声响或停止发出声响。



（a）蜂鸣器电路原理图　　(b) 蜂鸣器实物图

图 6-7 蜂鸣器

使用 USB 数据线一端连接 J4 口（SPI1 接口），另一端连接到蜂鸣器。

3．编程实践

程序可参考“.. \04-Soft\ch11-2\User_BEEF”工程，其编程步骤如下：

1）准备阶段

（1）拷贝 User 工程并重命名

拷贝“..\04-Soft \ch03”下的 User_Frame 工程，并重命名为 User_ BEEF。

（2）添加蜂鸣器引脚宏定义

查看 04_GEC\gec.h 文件，找到蜂鸣器传感器所接具有 GPIO 功能的引脚，并在 05_UserBoard \user.h 中添加蜂鸣器传感器的宏定义（如宏名为 BEEF）。

2）应用阶段

在 main.c 文件主函数中，对蜂鸣器模块进行初始化，并设置引脚方向为输出，端口引脚初始状态为低电平。

```
//（1.5）用户外设模块初始化
printf("蜂鸣器发出声音\n\0");
gpio_init(BEEF,GPIO_OUTPUT,1);              //蜂鸣器发出声音
Delay_ms(2000);                             //延时
printf("蜂鸣器停止发出声音\n\0");
gpio_init(BEEF,GPIO_OUTPUT,0);              //蜂鸣器停止发出声音
Delay_ms(3000);
printf("蜂鸣器发出声音\n\0");
gpio_init(BEEF,GPIO_OUTPUT,1);              //蜂鸣器发出声音
```

```
//（2）=======主循环部分（结尾）==================================================
…
```

4．运行结果

蜂鸣器的运行效果如图 6-8 所示。

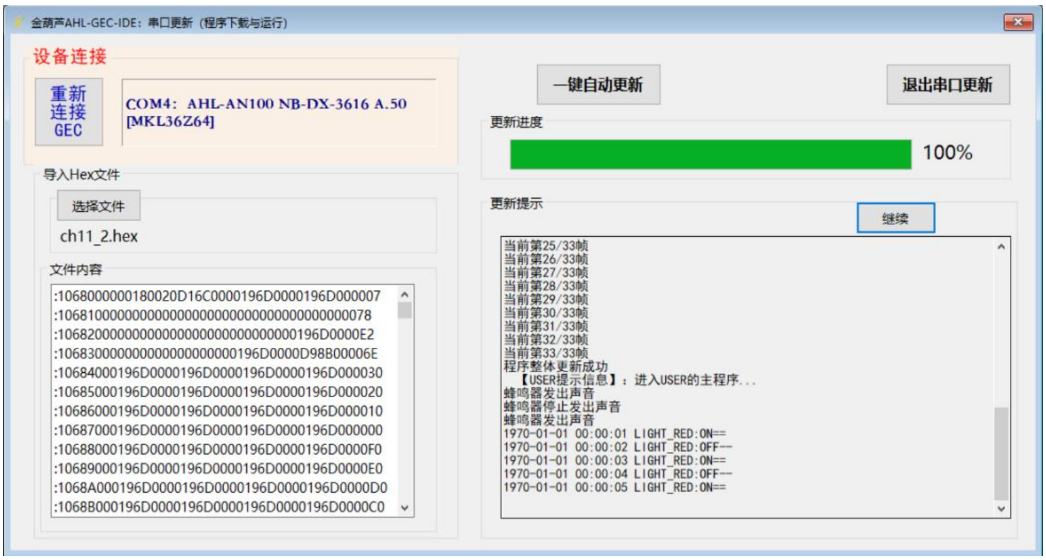


图 6-8 蜂鸣器运行结果

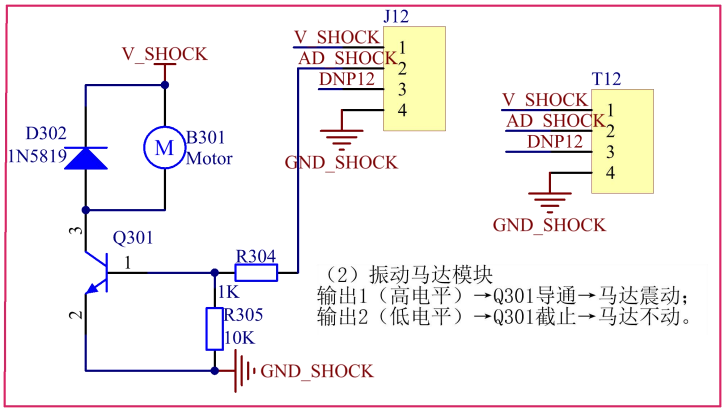
11.1.3 马达

1．原理概述

输出端电平设置为高电平，马达开始振动；输出端电平设置为低电平，马达不振动或停止振动；马达初始化默认是低电平，不振动。

2．电路原理

马达的电路原理图如图 6-9（a）所示，其实物图如图 6-9（b）所示。马达通过 AD_SHOCK 引脚来控制输出引脚的高低电平。当 AD_SHOCK 对应的状态值为 1 即高电平时，Q401 导通，马达开始振动；反之，当 AD_SHOCK 对应的状态值为 0 即低电平时，Q401 截止，马达不振动或停止振动。



（a）马达电路原理图　　（b）马达实物图

图 6-9 马达

使用 USB 数据线一端连接 J1 口，另一端连接到马达。

3．编程实践

程序可参考“.. \04-Soft\ch11-3\User_MOTOR”工程，其编程步骤如下：

1）准备阶段

（1）拷贝 User 工程并重命名

拷贝“..\04-Soft \ch03”下的 User_Frame 工程，并重命名为 User_MOTOR。

（2）添加马达引脚宏定义

查看 04_GEC\gec.h 文件，找到马达所接具有 GPIO 功能的引脚，并在 05_UserBoard \user.h 中添加马达的宏定义（如宏名为 MOTOR）。

2）应用阶段

在 main.c 文件主函数中，对马达模块进行初始化，并设置引脚方向为输出，端口引脚初始状态为低电平。

```
//（1.5）用户外设模块初始化
printf("马达开始振动\n\0");
gpio_init(MOTOR,GPIO_OUTPUT,1);
 Delay_ms(2000);
 printf("马达停止振动\n\0");
 gpio_init(MOTOR,GPIO_OUTPUT,0);
 Delay_ms(3000);
 printf("马达再次振动\n\0");
gpio_init(MOTOR,GPIO_OUTPUT,1);
//（2）======主循环部分（结尾）==========================================
…
```

4．运行结果

马达的运行效果如图 6-10 所示。

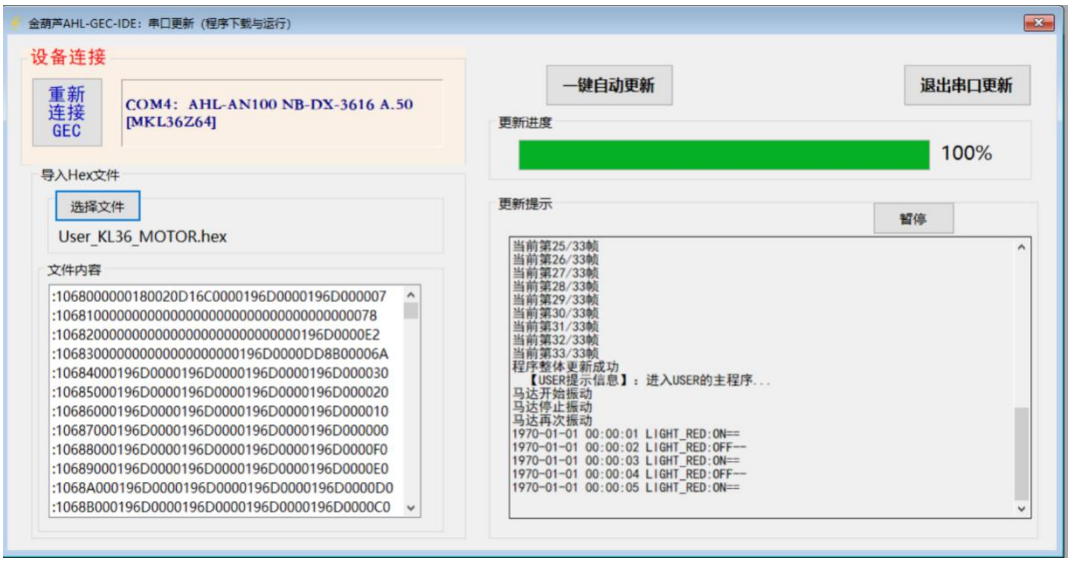


图 6-10 马达运行结果

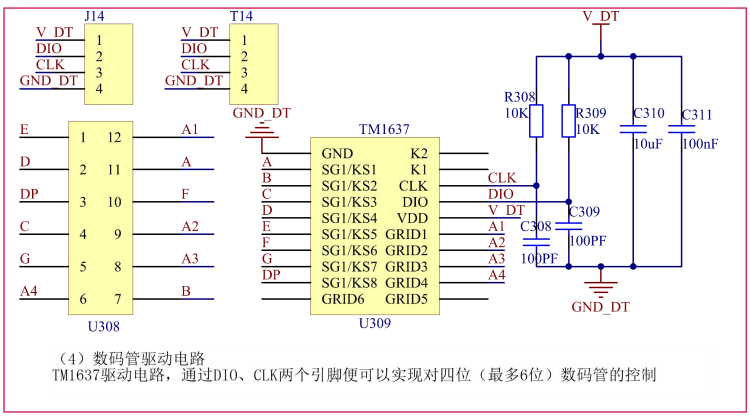
11.1.4 LED

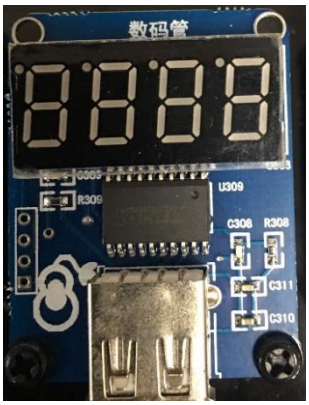
1．原理概述

在主函数中通过调用 TM1637_Display（a,a1,b,b1,c,c1,d,d1）函数可以点亮数码管，其中数码管的数字显示可在调用函数时设置，a、b、c、d 为要显示的 4 位数字大小；而 a1、b1、c1、d1 为四位数字后面的小数点显示，值为 0 则不显示小数点，值为 1 则显示小数点。

2．电路原理

数码管的电路原理图如图 6-11（a）所示，其实物图如图 6-11（b）所示。TM1637 驱动电路，通过 DIO 和 CLK 两个引脚实现对四位数码管的控制。DIO 引脚为数据输入输出，CLK 为时钟输入。数据输入的开始条件是 CLK 为高电平时，DIO 由高变低；结束条件是 CLK 为高电平时，DIO 由低电平变为高电平。





（a）LED 电路原理图　　（b）LED 实物图

图 6-11 LED

使用 USB 数据线一端连接 J7 口，另一端连接到 LED。

3．编程实践

程序可参考“.. \04-Soft\ch11-4\User_LED”工程，其编程步骤如下：

1）准备阶段

（1）拷贝 User 工程并重命名

拷贝“..\04-Soft \ch03”下的 User_Frame 工程，并重命名为 User_LED。

（2）添加 LED 应用构件

将“...\04-Soft\ch03\driver_component\ tm1637”下的 tm1637.c 和 tm1637.c 应用构件拷贝到.. \User_LED\05_UserBoard 下。

（3）添加 LED 应用构件头文件和 LED 引脚宏定义

在 05_UserBoard \user.h 中添加 LED 应用构件头文件（TM1637.h），查看 04_GEC\gec.h 文件，找到 LED 时钟引脚和数据引脚所对应的 GPIO 引脚，并在 05_UserBoard \user.h 中添加 LED 的宏定义（如宏名为 TM1637_CLK 和 TM1637_DIO）。

2）应用阶段

在 main.c 文件的主函数 main 中，要对数码管进行初始化，以及将数码管初始显

示为 1234.

```
//（1.5）用户外设模块初始化
TM1637_Init(TM1637_CLK,TM1637_DIO);          //初始化时钟引脚和数据引脚
printf("显示1234\n\0");
TM1637_Display(1,1,2,1,3,1,4,1);             //显示1234
Delay_ms(3000);
printf("显示4321\n\0");
TM1637_Display(4,1,3,1,2,1,1,1);             //显示4321
//  （2）=======主循环部分（开头）==========================================
…
```

4．运行结果

LED 运行效果如图 6-12、图 6-13 所示。

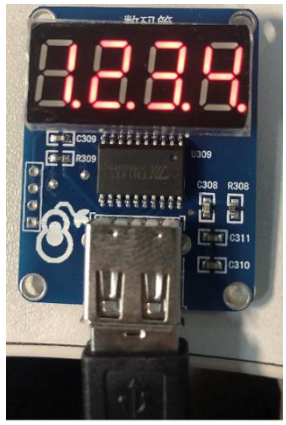


图 6-12 数码管初始显示 1234

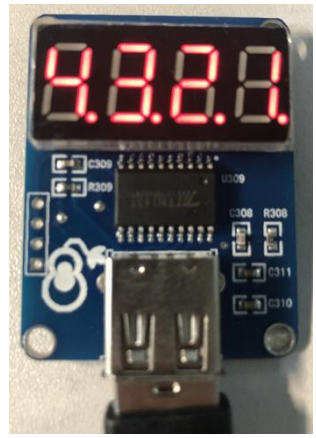


图 6-13 数码管显示 4321

11.2 开关量输入类驱动构件

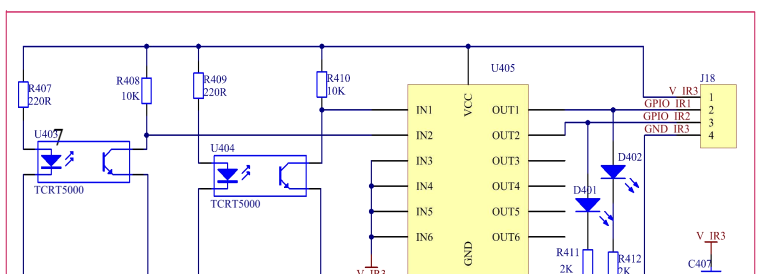
11.2.1 红外寻迹传感器

1．原理概述

当遮挡物体距离传感器红外发射管 2~2.5cm 之内，发射管发出的红外射线会被反射回来，红外接收管打开，模块输出端为高电平，指示灯亮；反之，若红外射线未被反射回来或反射回的强度不够大时，红外接收管处于关闭状态，模块输出端为低电平，指示灯不亮。

2．电路原理

红外寻迹传感器的电路原理图如图 6-14（a）所示，其实物图如图 6-14（b）所示。其中，V_IR3 引脚为左右两侧的红外发射器供电。GPIO_IR1 引脚为右侧的红外输出脚，并控制右侧的小灯亮暗；GPIO_IR2 引脚为左侧的红外输出脚，并控制左侧的小灯亮暗。



（a）红外寻迹传感器电路原理图　　（b）红外寻迹传感器实物图

图 6-14 红外寻迹传感器

红外寻迹传感器：使用 USB 线接到 J3 端口（SPI1 接口），用纸张靠近红外循迹传感器，红灯亮；撤掉纸张，红灯灭。

3．编程实践

红外寻迹传感器的程序可参考“.. \04-Soft\ch11-5\User_Ray”工程，其编程步骤如下：

1）准备阶段

（1）拷贝 User 工程并重命名

拷贝“..\04-Soft \ch03”下的 User_Frame 工程，并重命名为 User_Ray。

（2）给红外寻迹传感器取名

由于编程不是针对红外寻迹传感器这个实物进行的，而是根据它所接的引脚进行程序编写，考虑到程序的移植性问题，一般不直接使用所接引脚名，而是通过宏定义方式取个别名，以方便之后的识别与使用。因此，先查看 04_GEC\gec.h 文件，找到红外寻迹传感器所对应的 GPIO 引脚，然后在 05_UserBoard\user.h 中添加红外寻迹传感器的宏定义（如宏名为 RAY_LEFT 和 RAY_RIGHT）。

（3）添加中断处理程序宏定义

查找芯片的启动文件，找到具有 GPIO 中断功能的中断向量名，然后在 05_UserBoard\user.h 中重定义该中断向量（如宏定义名为 PORTC_PORTD_IRQHandler）。

（4）给 GPIO 中断触发条件取名

GPIO 中断的触发条件有上升沿触发、下降沿触发和双边沿触发，为了方便编程和程序的可移植性，应该采用宏定义的方式取个编程时使用的名称，具体代码已在 gpio.h 中给出。

```
// GPIO引脚中断类型宏定义
#define LOW_LEVEL      (8)        //低电平触发
#define HIGH_LEVEL     (12)       //高电平触发
#define RISING_EDGE    (9)        //上升沿触发
#define FALLING_EDGE (10)         //下降沿触发
#define DOUBLE_EDGE    (11)       //双边沿触发
```

2）应用阶段

（1）初始化外设模块并使能

在 main.c 文件的主函数中，要对红外寻迹传感器模块进行初始化，设置所接引脚

方向为输入，初始状态为低电平，并使能该模块。

使用模块的第一步是对模块初始化，在 main.c

```
//初始化红外循迹传感器两个引脚 设置为低电平输入
gpio_init(RAY_RIGHT,GPIO_INPUT,0);
gpio_init(RAY_LEFT,GPIO_INPUT,0);
//将引脚中断设为上升沿触发
gpio_enable_int(RAY_LEFT,RISING_EDGE);
gpio_enable_int(RAY_RIGHT,RISING_EDGE);
```

（2）编写中断处理程序

在中断处理程序 isr.c 中定义 PORTC_PORTD_IRQHandler，当 GPIO 引脚上升沿到来时，该中断被触发，在中断中会先判断是哪个引脚触发的，然后输出检测到有物体的提示信息。

```
void PORTC_PORTD_IRQHandler(void)
{
DISABLE_INTERRUPTS;                                    //关总中断
//---------------------------------------------------------------------
if(gpio_get_int(RAY_LEFT))
{
      gpio_clear_int(RAY_LEFT);
      printf("左侧红外检测有物体\r\n");
}
if(gpio_get_int(RAY_RIGHT))
{
      gpio_clear_int(RAY_RIGHT);
      printf("右侧红外检测有物体\r\n");
}
//---------------------------------------------------------------------
ENABLE_INTERRUPTS;                                     //开总中断
}
```

4．运行结果

红外寻迹传感器运行结果如下图 6-15 所示，通过串口输出提示信息。

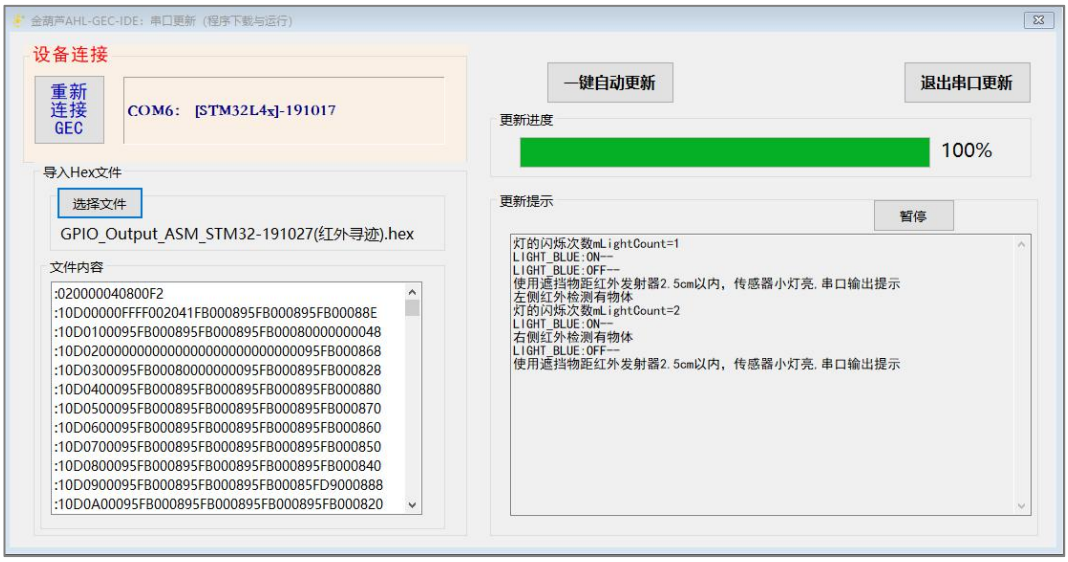


图 6-15 红外寻迹传感器运行结果

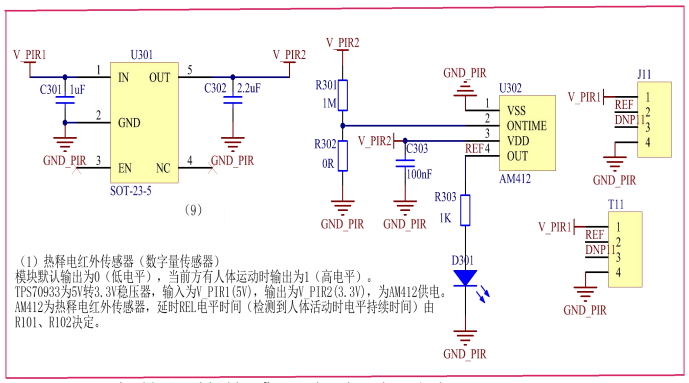
11.2.2 人体红外传感器

1．原理概述

任何发热体都会产生红外线，辐射的红外线波长（一般用μm）跟物体温度有关，表面温度越高，辐射能量越强。人体都有恒定的体温，所以会发出特定波长 10μm 左右的红外线，人体红外传感器通过检测人体释放的红外信号，判断一定范围内是否有人体活动。默认输出是低电平，当传感器检测到人体运动时，会触发高电平输出，小灯亮（有 3s 左右的延迟）。

2．电路原理

人体红外的电路原理图如图 6-16（a）所示，其实物图如图 6-16（b）所示。其中，V_PIR1 用于供电。REF 为输出引脚。



（a）人体红外传感器电路原理图　　（b）人体红外传感器实物图

图 6-16 人体红外传感器

人体红外传感器：使用 USB 线接到 J3 端口（SPI1 接口），当用手靠近靠近人体红外传感器，红灯亮；远离，延迟 3 秒左右，红灯灭。

3．编程实践

人体红外传感器的程序可参考“.. \04-Soft\ch11-6\User_RayHuman”工程，其编程步骤如下：

1）准备阶段

（1）拷贝 User 工程并重命名

拷贝“..\04-Soft \ch03”下的 User_Frame 工程，并重命名为 User_ RayHuman。

（2）给人体红外传感器取名

先查看 04_GEC\gec.h 文件，找到人体红外传感器所对应的 GPIO 引脚，然后在 05_UserBoard\user.h 中添加人体红外传感器的宏定义（如宏名为 RAY_HUMAN）。

（3）添加中断处理程序宏定义

查找芯片的启动文件，找到具有 GPIO 中断功能的中断向量名，然后在 05_UserBoard\user.h 中重定义该中断向量（如宏定义名为 PORTC_PORTD_IRQHandler）。

（4）给 GPIO 触发中断条件取名

参照 11.2.1 节

2）应用阶段

（1）模块初始化并使能

在 main.c 文件的主函数中，要对人体红外传感器模块进行初始化，并设置所接引脚方向为输入，初始状态为低电平，并使能该模块。

```
int main(void)
{
……
//（1.5）用户外设模块初始化
gpio_init(RAY_HUMAN,GPIO_INPUT,0);//初始化人体红外传感器模块
//（1.6）使能模块中断
gpio_enable_int(RAY_HUMAN,RISING_EDGE);//设置模块为上升沿触发
……
//（2）=======主循环部分（结尾）==============================================
}
```

（2）编写中断处理程序

在中断处理程序 isr.c 中定义 PORTC_PORTD_IRQHandler 中断，当 GPIO 引脚上升沿到来时，将触发该中断，输出检测到有人的提示信息。

```
void PORTC_PORTD_IRQHandler(void)
{
DISABLE_INTERRUPTS;                                    //关总中断
//-----------------------------------------------------------------
if(gpio_get_int(RAY_HUMAN))
{
    gpio_clear_int(RAY_HUMAN);
    printf("  红外检测有人\r\n");
}
//-----------------------------------------------------------------
ENABLE_INTERRUPTS;                                     //开总中断
}
```

4．运行结果

红外寻迹传感器运行结果如下图 6-17 所示，通过串口输出提示信息。

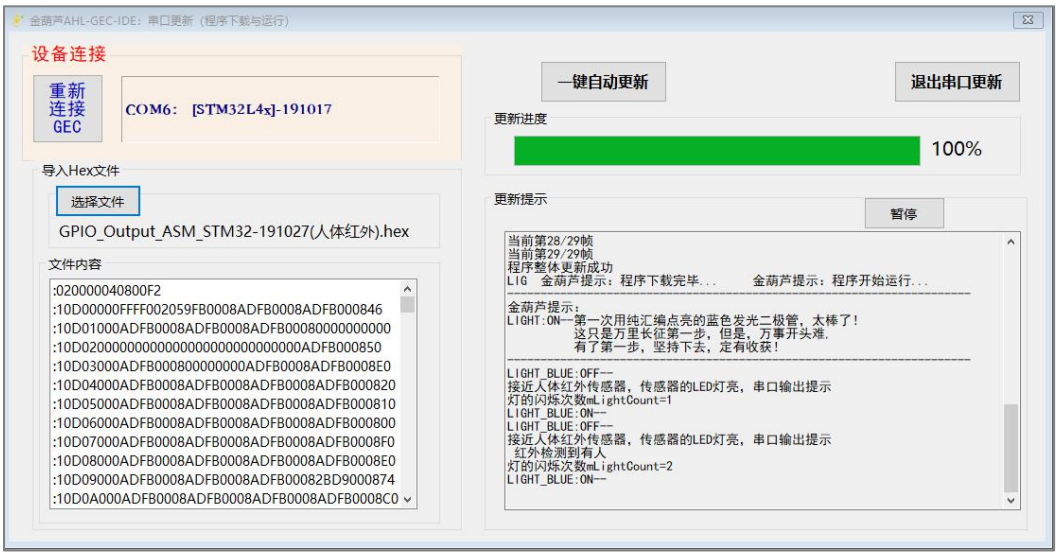


图 6-17 人体红外传感器运行结果

11.2.3 按钮

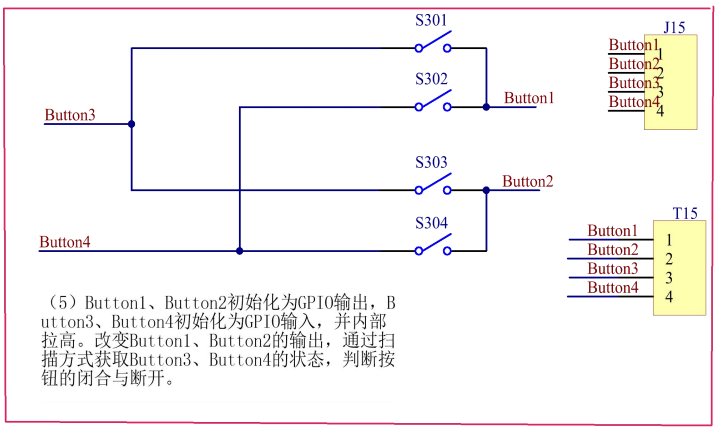
1．原理概述

按钮的工作原理很简单，对于常开触头，在按钮未被按下前，触头是断开的，按下按钮后，常开触头被连通，电路也被接通；对于常闭触头，在按钮未被按下前，触头是闭合的，按下按钮后，触头被断开，电路也被分断。

Button1、Button2 初始化为 GPIO 输出，Button3、Button4 初始化为 GPIO 输入，并内部拉高（设置为高电平）。改变 Button1、Button2 的输出，通过扫描方式获取 Button3、Button4 的状态，判断按钮的闭合与断开。若将 Button1 设置为低电平、Button2 设置为高电平，则 Button3 为低电平时，S301 闭合；Button3 为高电平时，S301 断开。同样，Button4 为低电平时，S302 闭合；Button4 为高电平时，S302 断开。若将 Button1 设置为高电平、Button2 设置为低电平，则 Button3 为低电平时，S303 闭合；Button3 为高电平时，S303 断开。同样，Button4 为低电平时，S304 闭合，Button4 为高电平时，S304 断开。

2．电路原理

按钮的电路原理图如图 6-18（a）所示，其实物图如图 6-18（b）所示。



（a）按钮电路原理图　　（b）按钮实物图

图 6-18 按钮

按钮：使用 USB 线接到 J6 端口（BUTTON 接口），另一端连接按钮。S301 对应 Button1 被按下的提示信息，S302 对应 Button2 被按下的提示信息，S303 对应 Button3 被按下的提示信息，S304 对应 Button4 被按下的提示信息。

3．编程实践

按钮的程序可参考“.. \04-Soft\ch11-7\User_Button”工程，其编程步骤如下：

1）准备阶段

（1）拷贝 User 工程并重命名

拷贝“..\04-Soft \ch03”下的 User_Frame 工程，并重命名为 User_ Button。

（2）添加定时器构件

将“...\04-Soft\ch03\driver_component\timer”下的 timer.c 和 timer.h 构件拷贝到..\User_ Button \03_MCU\MCU_drivers 下。

（3）添加定时器头文件及宏定义

在 04_GEC\gec.h 中添加定时器构件头文件（timer.h），查看 04_GEC\gec.h 文件，找到定时器所对应的引脚，然后在 05_UserBoard\user.h 中添加定时器的宏定义（如宏名为 TIMER_USER）。

（4）添加按钮引脚宏定义

查看 04_GEC\gec.h 文件，找到按钮所对应的具有 GPIO 引脚，然后在 05_UserBoard\user.h 中添加按钮的宏定义（如宏名分别为 Button1、Button2、Button3 和 Button4）。

（5）添加中断处理程序宏定义

查找芯片的启动文件，找到具有 GPIO 中断功能的中断向量名，然后在 05_UserBoard\user.h 中重定义该中断向量（如宏定义名为 TIMER_USER_Handler）。

（6）定义全局变量

在 includes.h 中定义四个按钮开关的全局变量

```
//定义按钮开关全局变量
G_VAR_PREFIX uint_8 switch1;
G_VAR_PREFIX uint_8 switch2;
G_VAR_PREFIX uint_8 switch3;
G_VAR_PREFIX uint_8 switch4;
```

（7）定义引脚方向

在 gpio.h 中定义 GPIO 引脚方向，输入为 0，输出为 1。

```
// GPIO引脚方向宏定义
#define GPIO_INPUT    (0)          //GPIO输入
#define GPIO_OUTPUT (1)           //GPIO输出
```

2）应用阶段

（1）外设初始化并使能

在 main.s 文件的主函数中，要对按钮模块进行初始化，将 Button1、Button2 初始化为 GPIO 输出，Button3、Button4 初始化为 GPIO 输入，并内部拉为高电平。同时，要初始化定时器并开启定时器中断。

```
int main(void)
{
……
timer_init(TIMER_USER,1000);                          //初始化定时器
gpio_init(Button1,GPIO_OUTPUT,1);                  //Button1
gpio_init(Button2,GPIO_OUTPUT,1);                  //Button2
gpio_init(Button3,GPIO_INPUT,0);                    //Button3
gpio_init(Button4,GPIO_INPUT,0);                    //Button4
gpio_pull(Button3,1);                                  //内部拉高
gpio_pull(Button4,1);                                  //内部拉高
timer_enable_int(TIMER_USER);      //开启定时器中断
```

（2）编写中断处理程序

在中断处理程序 isr.c 中定义 TIMER_USER_Handler 中断，当运行到达规定的时间时，将触发该中断，判断被用户按下的按钮是哪个，然后输出按钮被按下的提示信息。根据按钮的电路原理图可知四个按钮并不是单独工作的，所以要使用定时器中断

来判断按钮是否被用户按下。

```
//============================================================================
//文件名称：isr.c（中断处理程序源文件）
//框架提供：苏大arm技术中心(sumcu.suda.edu.cn)
//版本更新：2017.01：1.0；2019.02：A.12
//功能描述：提供中断处理程序编程框架
//============================================================================
#include "includes.h"
static int countKey=0;       //LPT中断计数
//============================================================================
//程序名称：TIMER_USER_Handler（TIMERA模块中断处理程序）
//触发条件：定时器计时达到初始化时设置的计时间隔时，触发定时器溢出中断
//============================================================================
void TIMER_USER_Handler(void)
{
    DISABLE_INTERRUPTS;                                  //关总中断
    //---------------------------------------------------------------------
    //（在此处增加功能）
    timer_clear_int(TIMER_USER);                         //清中断标志
    //Button1为0，Button2为1,检测switch1、switch2的状态
    if(countKey%2==0)
    {
    gpio_set(Button1,0);
    gpio_set(Button2,1);
    if(gpio_get(PTC_NUM|3)==0)
    {
    //Button3为低，说明switch1闭合
    switch1=1;
    printf("Button1 on\r\n");
    }
    else if(gpio_get(PTC_NUM|3)==1)
    {
    //Button3为高，说明switch1断开
    switch1=0;
    }
    if(gpio_get(PTC_NUM|0)==0)
    {
    //Button4为低，说明switch2闭合
    switch2=1;
    printf("Button2 on\r\n");
    }
    else if(gpio_get(PTC_NUM|0)==1)
    {
    //Button4为高，说明switch2断开
    switch2=0;
    }
    }
    else
```

```
        {
        //Button1为1，Button2为0,检测switch3、switch4的状态
        gpio_set(Button1,1);
        gpio_set(Button2,0);
        if(gpio_get(Button3)==0)
        {
        //Button3为低，说明switch3闭合
        switch3=1;
        printf("Button3 on\r\n");
        }
        else if(gpio_get(Button3)==1)
        {
        //Button3为高，说明switch3断开
        switch3=0;
        }
        if(gpio_get(Button4)==0)
        {
        //Button4为低，说明switch4闭合
        switch4=1;
        printf("Button4 on\r\n");
        }
        else if(gpio_get(Button4)==1)
        {
        //Button4为高，说明switch4断开
        switch4=0;
        }
        }
        if(countKey>100)
        {
        countKey=0;
        }
        countKey++;
        //-----------------------------------------------------------------
        ENABLE_INTERRUPTS;                              //开总中断
}
```

4．运行结果

当用户按下按钮时，串口烧录界面如图 6-19 所示，给出对应按钮按下的提示信息。

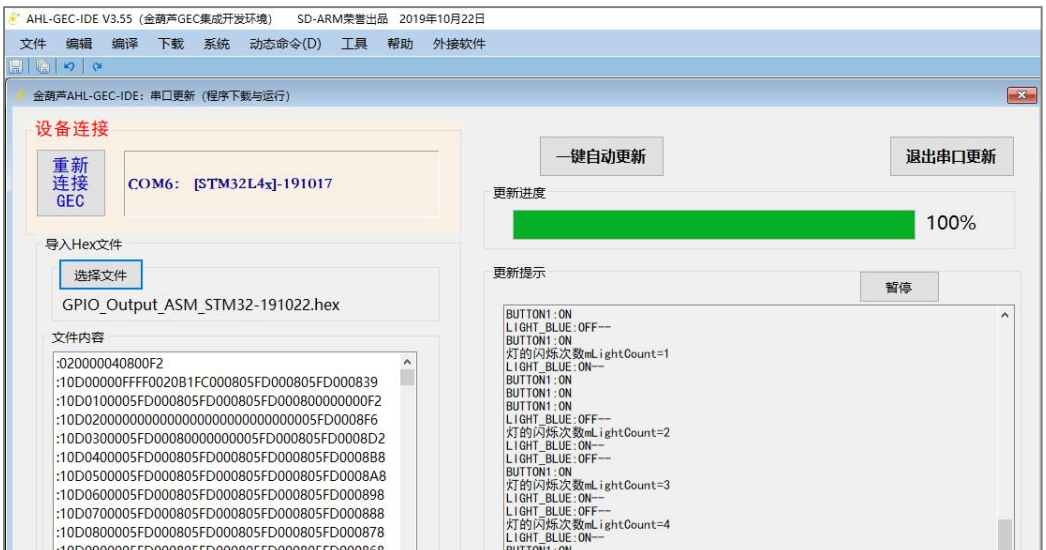


图 6-19 按钮运行结果

11.3 声音与加速度传感器驱动构件

11.3.1 声音传感器

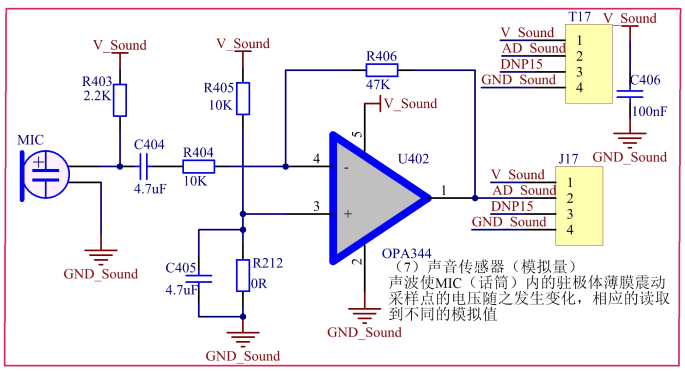
1．原理概述

声音传感器内置一个对声音敏感的电容式驻极体话筒（MIC）。声波使话筒内的驻极体薄膜振动，导致电容的变化，而产生与之对应变化的微小电压。这一电压随后被转化成 0-5V 的电压，经过 A/D 转换被数据采集器接受，并传送给计算机。

2．电路原理

声音传感器的电路原理图如图 6-20（a）所示，其实物图如图 6-20（b）所示。对于一个驻极体的声音传感器，内部有一个振膜、垫片和极板组成的电容器。当膜片受到声音的压强时产生振动，从而改变膜片与极板的距离，此时会引起电容的变化。由于膜片上的充电电荷是不变的，所以必然会引起电压的变化，这样就完成了声信号转换成电信号。但由于这个信号非常微弱且内阻非常高，需要通过 U402 电路进行阻抗变化和放大，将放大后的电信号通过 ADSound 采集后被微机处理。

使用 USB 数据线一端连接 J3 口，另一端连接到声音传感器。



（a）声音传感器电路原理图　　（b）声音传感器实物图

图 6-20 声音传感器

3．编程实践

程序可参考“.. \04-Soft\ch11-8\User_ADSound”工程，其编程步骤如下：

1）准备阶段

（1）拷贝 User 工程并重命名

拷贝“..\04-Soft \ch03”下的 User_Frame 工程，并重命名为 User_ADSound。

（2）添加 ADC 驱动构件

将“...\04-Soft\ch03\driver_component\adc”下的 adc.c 和 adc.h 驱动构件拷贝到..\User_ADSound\ 03_MCU\MCU_drivers 下。

（3）添加 ADC 引脚宏定义

查看 04_GEC\gec.h 文件，找到声音传感器所对应的具有 ADC 功能的引脚，然后在 05_UserBoard\user.h 中添加声音传感器的宏定义（如宏名为 ADCSound）。

（4）添加 ADC 构件的头文件

在“04_GEC\gec.h”文件中包含 ADC 构件的头文件（adc.h）

2）应用阶段

在“..\User_ADSound\07_NosPrg\main.c”文件中进行 ADC 的初始化、声音值的读取并输出。

（1）初始化 ADC

```
//（1.5）用户外设模块初始化
adc_init(ADCSound,16);       //初始化ADC，采样精度16
```

（2）读取声音 AD 值并显示

```
//（2）主循环部分
printf("采集声音AD值为：%d\n",adc_read(ADCSound));      //输出声音传感器ADC值
```

4．运行结果

烧录程序后，打开串口调试工具，用力向声音传感器吹去，采集到的声音 AD 值会相应发生变化，如图 6-21 所示。

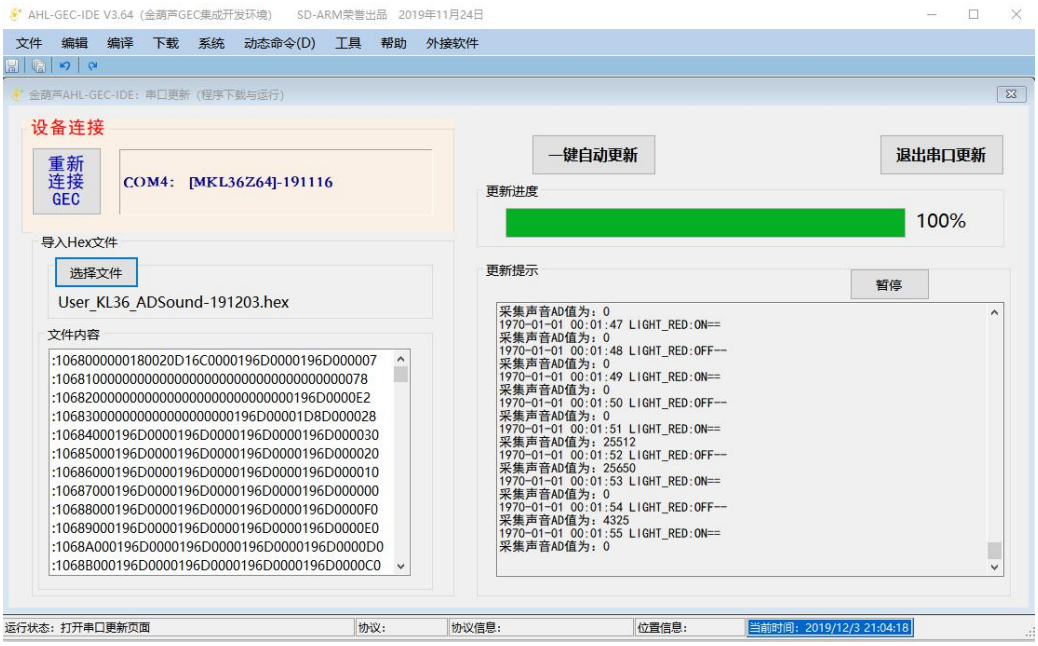


图 6-21 采集声音 AD 值结果

11.3.2 加速度传感器

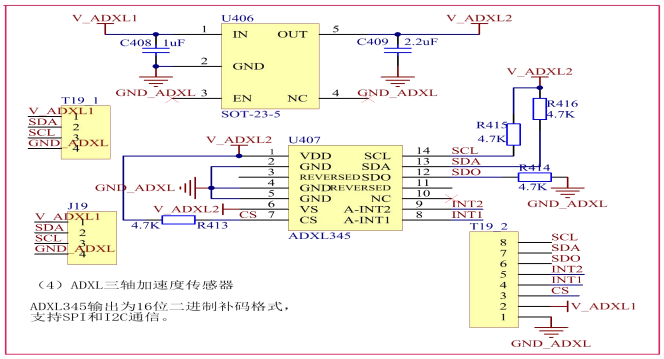
1．原理概述

加速度传感器首先由前端感应器件感测加速度的大小（因为传感器内的差分电容

会因为加速度而改变，从而传感器输出的幅度与加速度成正比），然后由感应电信号器件转为可识别的电信号，这个信号首先是模拟信号，然后通过 AD 转换器可以将模拟信号转换为数字信号，再通过串口读取数据。

2．电路原理

加速度的电路原理图如图 6-22（a）所示，其实物图如图 6-22（b）所示。因为传感器内的差分电容会因为加速度而改变，从而传感器输出的幅度与加速度成正比，所以可以通过 SPI 或者 I2C 方法，获得输出的 16 进制数，从而显示出来。



（a）加速度传感器电路原理图　　（b）加速度传感器实物图

图 6-22 加速度传感器电路原理

使用 USB 数据线一端连接 J2 口（I2C0 接口），另一端连接到加速度传感器。

3．编程实践

程序可参考“.. \04-Soft\ch11-9\User_Acceleration”工程，其编程步骤如下：

1）准备阶段

（1）拷贝 User 工程并重命名

拷贝“..\04-Soft \ch03”下的 User_Frame 工程，并重命名为 User_Acceleration。

（2）添加 I2C 驱动构件

将“...\04-Soft\ch03\driver_component\i2c”下的 i2c.c 和 i2c.h 驱动构件拷贝到“.. \User_Acceleration\ 03_MCU\MCU_drivers”下，在 gec.h 中包含 I2C 驱动构件的头文件（i2c.h）

（3）添加加速度传感器应用构件

将“...\04-Soft\ch03\driver_component\ adlx345”下的 adlx345.c 和 adlx345.c 应用构件拷贝到.. \User_Acceleration\05_UserBoard 下。

（4）添加头文件和 I2C 模块宏定义

在 user.h 中添加加速度传感器的应用构件头文件（adlx345.h），查看 04_GEC\gec.h 文件，找到加速度传感器所对应的具有 I2C 功能的引脚，然后在 05_UserBoard\user.h 中添加加速度传感器的宏定义（如宏名为 i2cAcceleration）。

2）应用阶段

在“..\ User_Acceleration\07_NosPrg\main.c”文件中进行加速度传感器的初始化、加速度值的读取并输出。

（1）定义加速度传感器使用的局部变量

```
//（1.1）声明main函数使用的局部变量
uint_8 xyzData[6];              //x、y、z轴倾角，均占两个字节
```

```
uint_16 xdata,ydata,zdata; //x轴倾角
uint_8 checkdata;                //ADLX345的验证数据,正确接收为0xe5
```

（2）初始化加速度传感器

```
//（1.5）用户外设模块初始化
adlx345_init(i2cAcceleration,0x0B,0x08,0x08,0x80,0x00,0x00,0x05);//初始化ADLX345(J2端口)
adlx345_read1(0x00,&checkdata);      //读取adxl345校验数据
```

（3）读取加速度传感器数据并通过串口输出 x，y，z 轴倾角

```
//加速度传感器初始化及读取操作
adlx345_init( i2cAcceleration ,0x0B,0x08,0x08,0x80,0x00,0x00,0x05);//初始化ADLX345(J2端口)
adlx345_read1(0x00,&checkdata);      //读取adxl345校验数据
Delay_ms(5);
adlx345_readN(0x32,xyzData,6);                                    //读倾角传感器数值
xdata = (xyzData[1]<<8)+xyzData[0];      //x方向倾角
ydata = (xyzData[3]<<8)+xyzData[2];      //y方向倾角
zdata = (xyzData[5]<<8)+xyzData[4];      //z方向倾角
printf("xdata=%d",xdata);                //输出x方向倾角
printf("ydata=%d",ydata);                //输出y方向倾角
printf("zdata=%d\n",zdata);              //输出z方向倾角
```

4．运行结果

烧录程序后，打开串口调试工具，晃动加速度传感器，采集到的 x，y，z 倾角值会相应发生变化，如图 6-23 所示。

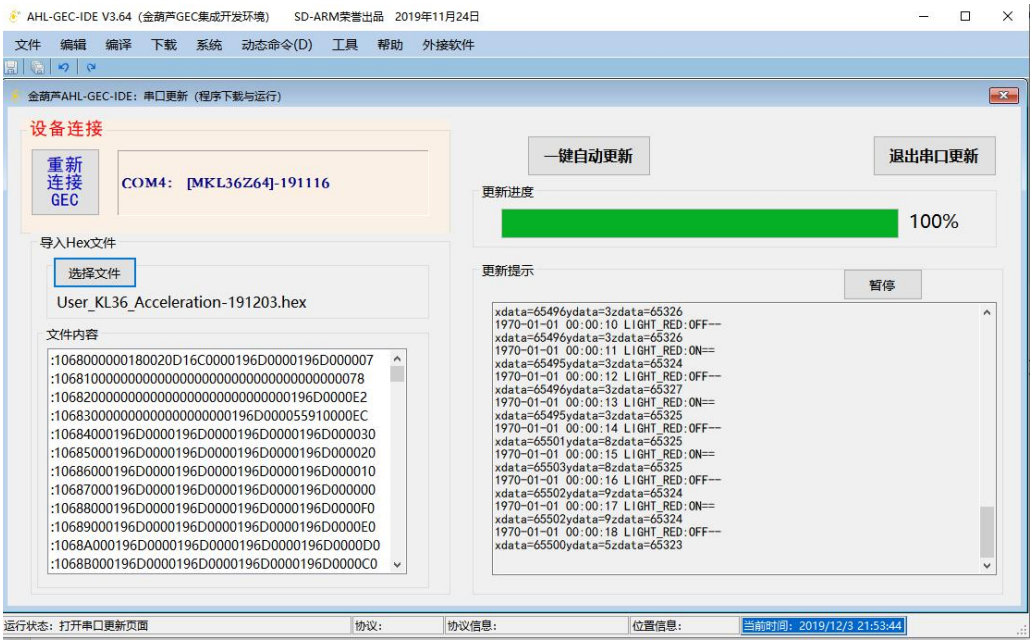


图 6-23 加速度传感器采集 x，y，z 倾角值结果