物联网导论课程报告

适应上班摸鱼需求的 Alt+Tab 老板键自动触发系统设计

物联网 2101 班

姓名：王聚臣

学号：202113385

物联网导论课程报告

适应上班摸鱼需求的 Alt+Tab 老板键自动触发系统设计

[摘要] 本设计的初衷，是希望我们在将来面对超负荷的工作时，能有适度放松的选择。当下年轻人往往承担了巨大的生活压力，同时目前也时常会曝出一些码农加班猝死的事件。鉴于这种社会环境，开发一种能帮助年轻人偶尔在上班时摸鱼的设备，似乎变得有些必要。本结课作业，我基于四周的理论知识学习所得的对物联网工程这一概念的认识，自行采购各种模块，运用已有的 c++编程知识和 arduino 编程知识，设计的一个能够自动监控前方区域，并且在有人/物经过时自动触发所连接的 PC 机的 alt+tab 这一老板键，并实现快速切屏的小装置。

“难登大雅之堂，尽显叛逆之心。”

一．课程设计素材

1.硬件：

arduino uno 开发板　　淘宝链接：https://m.tb.cn/h.UVMj7cO

激光漫反射传感器　　淘宝链接：https://m.tb.cn/h.U48MNqg

蓝牙 HC-06 模块　　淘宝链接：https://m.tb.cn/h.UVMQ07H

充电宝和 USB 转 12V 电源线（用于供电）

杜邦线，USB 转 TTL 接头等杂项若干……

2.软件开发环境：

windows 10 系统，使用 c++语言，编译器为 mingw64，编辑器为 VScode

Arduino IDE

课设涉及到一个 c++串口通讯的开源项目：https://github.com/ayowin/WZSerialPort

串口调试助手 SSCOM

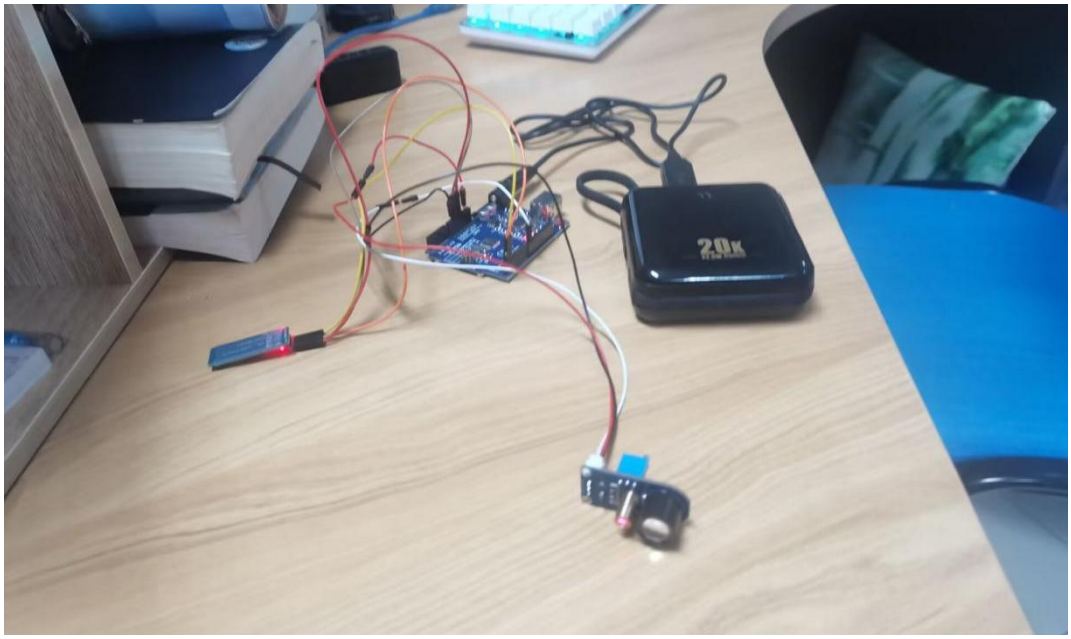
课设中的 pc 端应用程序，以及所有涉及到的源码和本设计的实物工作效果演示视频，均放在了我的 github 仓库里：https://github.com/JmpCliff/Introduction-to-the-Internet-of-Things

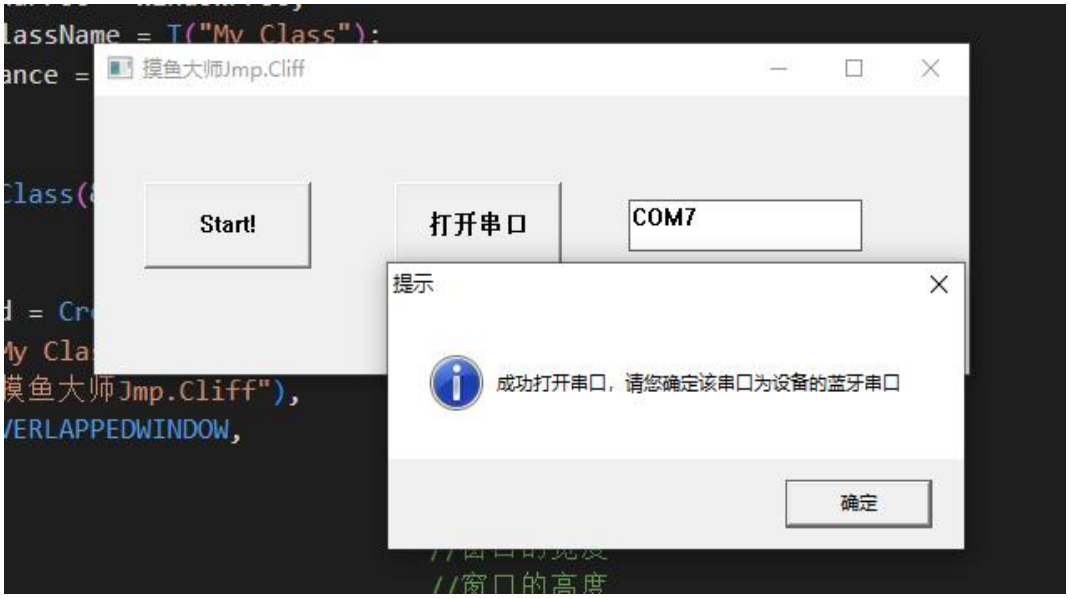
二．设计原理及工作效果阐述：

本设计选用的激光漫反射传感器集成了接受和发射功能，在遮挡物从前方经过时，传感器发射的激光在遮挡物表面发生漫反射并返回，会触发传感器输出低电平。随后单片机在收到低电平信号后会向 pc 端口通过已经事先建立好的蓝牙通讯链路发送消息（在本设计中为字符串”CATCH!”）。此时在 PC 机上运行的程序中会收到串口中的信息并且触发 alt+tab 组合键快速将当前的页面切走。

本设计额外在 pc 端写了一个对用户友好的可视化的操作界面，便于用户进行监控功能的启动和停止以及串口号的选择。

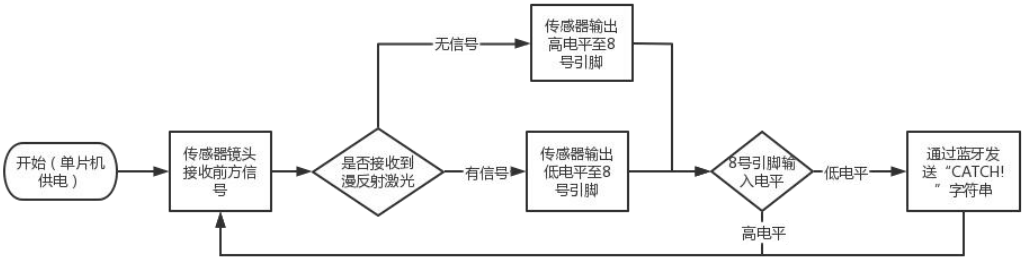
实物图和 PC 端程序效果总览如下图所示，实物详细电路连接图将在设计过程细节中展示。





软件效果：给单片机供电并正确链接蓝牙后，在文本框中输入串口号，点击打开串口按钮，得到成功打开串口的提示，再点击 start 即可实现监控。期间可以通过点击 Stop 键（与 start 键为同一个按钮）暂停监控并调整按钮。触发老板键后，监控会自动中断，按钮会由 stop 变为 start。

设计流程图如下，首先是单片机设备：

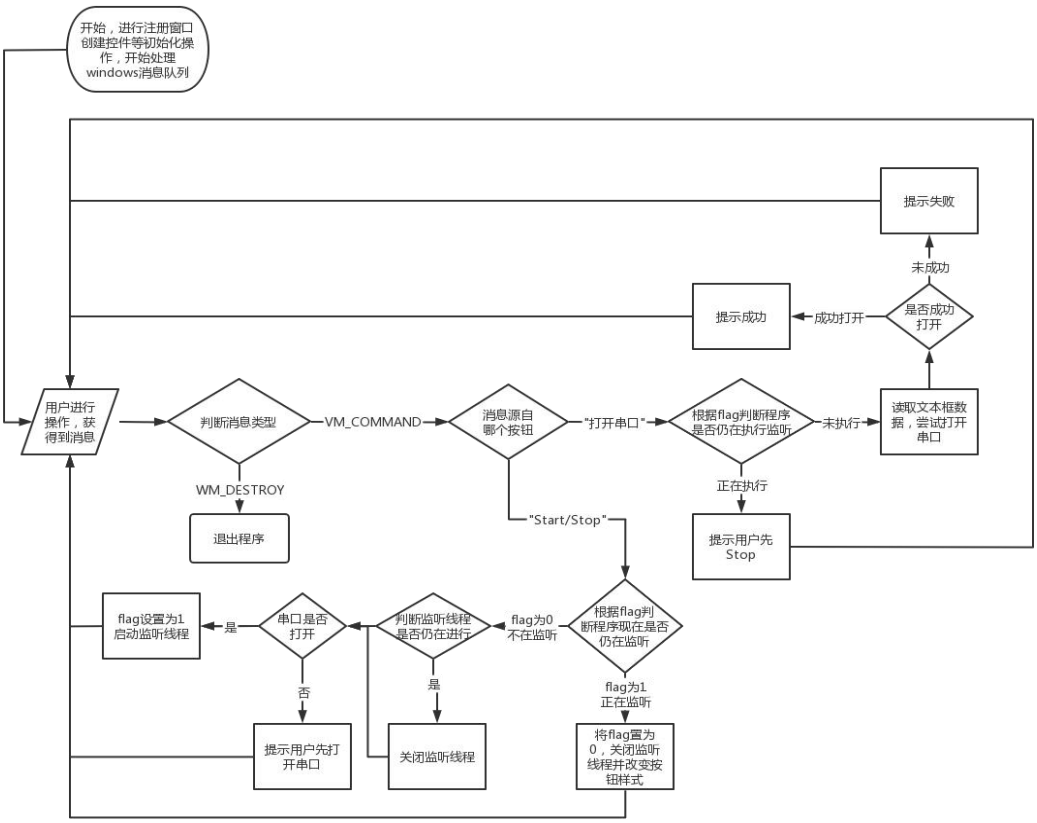


软件部分：

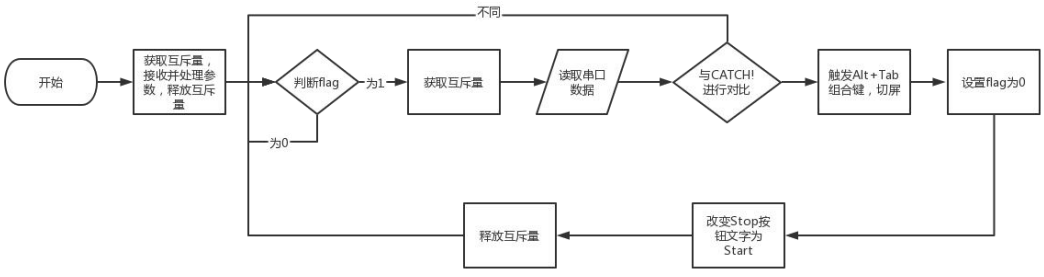
首先是程序主线程部分，即程序的主体。

流程图中提到的子线程（名为 AltTAB），即为可以从串口读取数据并触发老板键的线程。

由于我是转专业学生，大一并未接触过系统化的计算机类专业知识的教学，受限于极为匮乏的知识储备，我对于并发机制并不十分了解，在设计中屡屡碰壁，最后被迫选择采用了一个 flag 来标志程序是否在执行监听。当触发传感器后，flag 会被置为 0 但是子线程并未被关闭，而是采用在线程的 while 循环条件中检查了全局变量 flag 的值来判断是否执行线程核心代码（详见本报告后续的详细设计过程），这造成了资源浪费且不利于代码的维护，算是一处败笔，有待改进。



AltTAB 子线程流程：



三．详细设计过程

1. 调试激光漫反射传感器

我所购买的激光漫反射传感器有三个引脚：OUT，+5V，GND。其中，+5V 和 GND 引脚应分别与 arduino 开发板上 POWER 引脚区中的 5V 和 GND 相连，这两者是用于给传感器提供 5V 的工作电压。接通后，传感器上暗金色的发射管应该发出红色的激光束。用手在前方去遮挡时，传感器背部的红色指示灯会亮起，表明传感器接收到了漫反射的激光信号，即感应到了遮挡物。

激光漫反射传感器背面有一个由一枚小螺丝控制的一个电位器，可以通过旋转它来调节传感器的感应距离。

由于该传感器原理为接受漫反射返回的激光信号，所以这个传感器在检测黑色物体或者表面过于光滑的物体时不太灵敏。受限于经济条件限制问题，本人匮乏的知识储备问题，以及疫情影响带来的调试时间不足问题等诸多因素，我没能找到适应性更强的传感器来实现对这些的检测——也许用个摄像头可以解决？但是这可能又要涉及到图像处理等更多的问题。在这里我仅仅能抛出一个想法而不能解决，略有遗憾。

正常情况下传感器会在 OUT 引脚输出高电平信号，在感应到遮挡物后，传感器会通过 OUT 引脚输出低电平信号。这里我将 OUT 引脚连接到了 arduino uno 开发板的 8 号数字引脚。

这时可以向单片机上传以下测试程序进行调试。我这里是实现了在单片机接受到信号后让 13 号板载 LED 灯亮起 5 秒的功能。根据 LED 等能否正常亮起可以检验该传感器能否正常工作。

```
int LED=13;
int RayIn=8;

void setup(){
    pinMode(LED,OUTPUT);
    pinMode(RayIn,INPUT);
}
void loop(){
    if(digitalRead(RayIn)==LOW){
        digitalWrite(LED,HIGH);
        delay(5000);
        digitalWrite(LED,LOW);
    }
}
```

2. 配置蓝牙模块

我所选购的蓝牙模块型号为 HC-06，这个是蓝牙从机模块，但是由于我们 PC 端自带蓝牙主机，所以选购价格相对低廉的从机就已经足够了。

蓝牙 HC-06 模块有 4 个引脚，分别为 VCC，GND，RXD，TXD。前两者用于给蓝牙模块提供 3.3V 的工作电压（这个模块也可以接受 5V 的工作电压，两种电压 arduino 开发板都有直接提供，无需再搭配电阻构建分压电路），后两者需要与所连接设备的 TX 和 RX 引脚相连（即 T 对 R，R 对 T，T 是发送，R 是接收，发送端当然要对应接收端了）。

蓝牙模块在使用之前需要先用 AT 指令进行一些设置。我们可以先用 USB 转 TTL 接头将蓝牙与 PC 机相连，然后可以下载一个串口调试助手（SSCOM），找到蓝牙对应串口号（我这里是 COM4）。输入 AT 指令后蓝牙会返回 OK 表明接下来可以通过 AT 指令进行蓝牙的设置。在这里我们可以输入 AT+BAUDx 来设定蓝牙的波特率(具体的由 x 对应)，我这里波特率选择的是 9600，对应的是 AT+BAUD4。同时我们还要在这里设置蓝牙的名称和密码，用 AT+NAMEname 和 AT+PINxxx 即可实现，在这里，name 我设置的为 Jmp.Cliff，pin 设置的为 0135，即应该输入 AT+NAMEJmp.Cliff 和 AT+PIN0135，设置成功的话会有回显 OKname 和 OKsetpin。

这里要注意，蓝牙模块校验位默认为 0，数据位为 8 位，停止位为 1，默认为同步传输。

之后即可将蓝牙串口连接到 arduino uno 开发板上，这里要注意，插入 arduino 的 RX 和 TX 引脚后，由

于 arduino 默认只有一个硬件串口，这就会导致我们的程序无法通过原来的串口上传到单片机：因为原来与 PC 机相连的串口被覆盖为了蓝牙串口。所以给单片机上传新的程序时，需要先把 TX，RX 引脚给拔下来，或者使用软件串口增加串口数目解决冲突问题。我这里采用前一个方案。

这里要注意，将 PC 端与蓝牙链接时，不是仅仅在蓝牙列表中与 Jmp.Cliff 这一设备进行蓝牙配对即可解决问题。我们要在 windows 10 的 设置-蓝牙和其他设备-更多蓝牙选项(在右侧)-COM 端口-添加 中，选中传出选项，点击浏览，找到我们的蓝牙设备：Jmp.Cliff，添加串口并记住对应串口号（我这里是 COM7）。（本来想设置为自动配置，但因为这要涉及到更多的 windows API，受时间限制未能进行深入学习，所以做成的东西不够傻瓜式，因此第一次使用需要手动配置。）

蓝牙通讯链路建立完成后就可以把单片机上的程序完成了。这里打开串口时波特率要设置为 9600，与我们先前设置的串口波特率一致。

Text_arduino.ino:

```
int LED=13;
int RayIn=8;

void setup(){
    pinMode(LED,OUTPUT);
    pinMode(RayIn,INPUT);
    Serial.begin(9600);
    //Serial.println("hello");
}
void loop(){

    if(digitalRead(RayIn)==LOW){
        digitalWrite(LED,HIGH);
        Serial.println("CATCH!");
        delay(5000);
        digitalWrite(LED,LOW);
    }
    //Serial.println("This is a test!");
    //delay(1000);
}
```

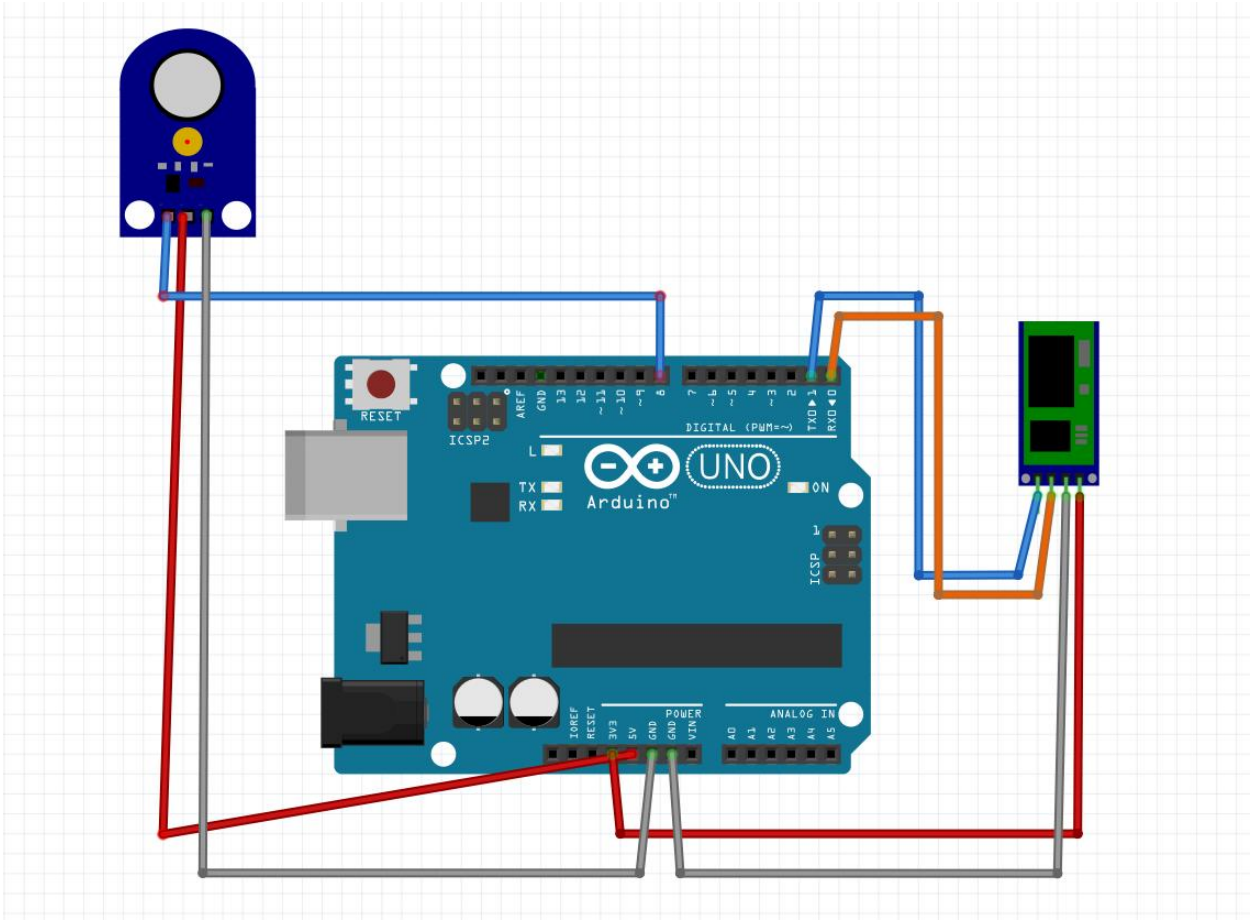
上传程序之后，组装好蓝牙和传感器，单片机的设计工作就已经完成了。

实物的连接方式如下图所示：

其中右侧模块为蓝牙 HC06 模块示意图，元件引脚从右至左依次为 RDX,TDX,GND,VCC。

左上角模块为先前所采用的激光漫反射传感器，元件从左到右的引脚依次为 OUT,+5V,GND。

按照图示进行连接即可。



3. PC 端通过串口读取蓝牙通信消息。（本节内容主要依靠网络上的一个开源项目来实现。并非本人亲自编写）

在之前的操作中，我们已经建立好了这个蓝牙通讯链路，但是要让 PC 端能响应单片机的消息，首先要能让 PC 端能通过串口接收对应的数据，这里我在网络上查找到了一个名为 WzSerialPort 的开源项目，这个项目提供了一个 WzSerialPort 类型，可以及其容易地实现 PC 端通过串口与其他设备进行通信。

因为我的设备有一些不明原因，导致原项目中的宽字符串不能正常编译通过，所以我对这个程序略微做了一些改动，额外包含了一个 tchar.h 头文件，用_T 宏定义处理宽字符串后问题就得到了解决。

此外，该项目原本设定的串口数据读取速度过慢。我这里将 读间隔超时、读时间系数、读时间常量三个变量（均在源码中有注释）均调小了十倍。使得程序能更快地读取串口数据，以及时做出反应。

以下为微调后的 WzSerialPort 项目源码，共两个文件，大致仍与原项目相同

该项目源码较为冗长且在设计中并未全部使用，可以翻到第 12 页暂时跳过，后续调用接口时会有说明。

WzSerialPort.h:

```
//-*- coding : UTF-8 -*-

#ifndef _WZSERIALPORT_H
#define _WZSERIALPORT_H

class WzSerialPort
{
public:
```

```
    WzSerialPort();
    ~WzSerialPort();
    // 打开串口,成功返回 true，失败返回 false
    // portname(串口名): 在 Windows 下是"COM1""COM2"等，在 Linux 下是"/dev/ttyS1"等
    // baudrate(波特率): 9600、19200、38400、43000、56000、57600、115200
    // parity(校验位): 0 为无校验，1 为奇校验，2 为偶校验，3 为标记校验（仅适用于 windows）
    // databit(数据位): 4-8(windows),5-8(linux)，通常为 8 位
    // stopbit(停止位): 1 为 1 位停止位，2 为 2 位停止位,3 为 1.5 位停止位
    // synchronizeflag(同步、异步,仅适用与 windows): 0 为异步，1 为同步
    bool open(const char* portname, int baudrate, char parity, char databit, char stopbit,
char synchronizeflag=1);
    //关闭串口，参数待定
    void close();
    //发送数据或写数据，成功返回发送数据长度，失败返回 0
    int send(const void *buf,int len);
    //接受数据或读数据，成功返回读取实际数据的长度，失败返回 0
    int receive(void *buf,int maxlen);
private:
    int pHandle[16];
    char synchronizeflag;
};
#endif
```

WzSerialPort.cpp

```
//-*- coding : UTF-8 -*-

#include "WzSerialPort.h"
#include "tchar.h"
#include <stdio.h>
#include <string.h>

#include <WinSock2.h>
#include <windows.h>
WzSerialPort::WzSerialPort()
{}
WzSerialPort::~WzSerialPort()
{}

bool WzSerialPort::open(const char* portname,
                        int baudrate,
                        char parity,
                        char databit,
                        char stopbit,
                        char synchronizeflag)
{
```

```
    this->synchronizeflag = synchronizeflag;
    HANDLE hCom = NULL;
    if (this->synchronizeflag)
    {
        //同步方式
        hCom = CreateFileA(portname, //串口名
                                    GENERIC_READ | GENERIC_WRITE, //支持读写
                                    0, //独占方式，串口不支持共享
                                    NULL,//安全属性指针，默认值为 NULL
                                    OPEN_EXISTING, //打开现有的串口文件
                                    0, //0：同步方式，FILE_FLAG_OVERLAPPED：异步方式
                                    NULL);//用于复制文件句柄，默认值为 NULL，对串口而言该
参数必须置为 NULL
    }
    else
    {
        //异步方式
        hCom = CreateFileA(portname, //串口名
                                    GENERIC_READ | GENERIC_WRITE, //支持读写
                                    0, //独占方式，串口不支持共享
                                    NULL,//安全属性指针，默认值为 NULL
                                    OPEN_EXISTING, //打开现有的串口文件
                                    FILE_FLAG_OVERLAPPED, //0：同步方式，
FILE_FLAG_OVERLAPPED：异步方式
                                    NULL);//用于复制文件句柄，默认值为 NULL，对串口而言该
参数必须置为 NULL
    }

    if(hCom == (HANDLE)-1)
    {
        return false;
    }
    //配置缓冲区大小
    if(! SetupComm(hCom,1024, 1024))
    {
         return false;
     }
     // 配置参数
    DCB p;
    memset(&p, 0, sizeof(p));
    p.DCBlength = sizeof(p);
    p.BaudRate = baudrate; // 波特率
    p.ByteSize = databit; // 数据位
    switch (parity) //校验位
```

```
{
case 0:
    p.Parity = NOPARITY; //无校验
    break;
case 1:
    p.Parity = ODDPARITY; //奇校验
    break;
case 2:
    p.Parity = EVENPARITY; //偶校验
    break;
case 3:
    p.Parity = MARKPARITY; //标记校验
    break;
}
switch(stopbit) //停止位
{
case 1:
    p.StopBits = ONESTOPBIT; //1 位停止位
    break;
case 2:
    p.StopBits = TWOSTOPBITS; //2 位停止位
    break;
case 3:
    p.StopBits = ONE5STOPBITS; //1.5 位停止位
    break;
}
if(! SetCommState(hCom, &p))
{
    // 设置参数失败
    return false;
}
//超时处理,单位：毫秒
//总超时＝时间系数×读或写的字符数＋时间常量
COMMTIMEOUTS TimeOuts;
TimeOuts.ReadIntervalTimeout = 100; //读间隔超时
TimeOuts.ReadTotalTimeoutMultiplier = 50; //读时间系数
TimeOuts.ReadTotalTimeoutConstant = 500; //读时间常量
TimeOuts.WriteTotalTimeoutMultiplier = 500; // 写时间系数
TimeOuts.WriteTotalTimeoutConstant = 2000; //写时间常量
SetCommTimeouts(hCom,&TimeOuts);

PurgeComm(hCom,PURGE_TXCLEAR|PURGE_RXCLEAR);//清空串口缓冲区
memcpy(pHandle, &hCom, sizeof(hCom));// 保存句柄
return true;
```

```
}

void WzSerialPort::close()
{
    HANDLE hCom = *(HANDLE*)pHandle;
    CloseHandle(hCom);
}

int WzSerialPort::send(const void *buf,int len)
{
    HANDLE hCom = *(HANDLE*)pHandle;

    if (this->synchronizeflag)
    {
        // 同步方式
        DWORD dwBytesWrite = len; //成功写入的数据字节数
        BOOL bWriteStat = WriteFile(hCom, //串口句柄
                                    buf, //数据首地址
                                    dwBytesWrite, //要发送的数据字节数
                                    &dwBytesWrite,//DWORD*，用来接收返回成功发送的数据字
节数
                                    NULL); //NULL 为同步发送，OVERLAPPED*为异步发送
        if (!bWriteStat)
        {
            return 0;
        }
        return dwBytesWrite;
    }
    else
    {
        //异步方式
        DWORD dwBytesWrite = len; //成功写入的数据字节数
        DWORD dwErrorFlags; //错误标志
        COMSTAT comStat; //通讯状态
        OVERLAPPED m_osWrite; //异步输入输出结构体
        //创建一个用于 OVERLAPPED 的事件处理，不会真正用到，但系统要求这么做
        memset(&m_osWrite, 0, sizeof(m_osWrite));
        m_osWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, _T("WriteEvent"));
        ClearCommError(hCom, &dwErrorFlags, &comStat); //清除通讯错误，获得设备当前状态
        BOOL bWriteStat = WriteFile(hCom, //串口句柄
            buf, //数据首地址
            dwBytesWrite, //要发送的数据字节数
            &dwBytesWrite, //DWORD*，用来接收返回成功发送的数据字节数
            &m_osWrite); //NULL 为同步发送，OVERLAPPED*为异步发送
```

```
        if (!bWriteStat)
        {
            if (GetLastError() == ERROR_IO_PENDING) //如果串口正在写入
            {
                WaitForSingleObject(m_osWrite.hEvent, 1000); //等待写入事件 1 秒钟
            }
            else
            {
                ClearCommError(hCom, &dwErrorFlags, &comStat); //清除通讯错误
                CloseHandle(m_osWrite.hEvent); //关闭并释放 hEvent 内存
                return 0;
            }
        }
        return dwBytesWrite;
    }
}
int WzSerialPort::receive(void *buf,int maxlen)
{
    HANDLE hCom = *(HANDLE*)pHandle;
    if (this->synchronizeflag)
    {
        //同步方式
        DWORD wCount = maxlen; //成功读取的数据字节数
        BOOL bReadStat = ReadFile(hCom, //串口句柄
                                  buf, //数据首地址
                                  wCount, //要读取的数据最大字节数
                                  &wCount, //DWORD*,用来接收返回成功读取的数据字节数
                                  NULL); //NULL 为同步发送，OVERLAPPED*为异步发送
        if (!bReadStat)
        {
            return 0;
         }
         return wCount;
     }
     else
     {
         //异步方式
        DWORD wCount = maxlen; //成功读取的数据字节数
        DWORD dwErrorFlags; //错误标志
        COMSTAT comStat; //通讯状态
        OVERLAPPED m_osRead; //异步输入输出结构体
        //创建一个用于 OVERLAPPED 的事件处理，不会真正用到，但系统要求这么做
        memset(&m_osRead, 0, sizeof(m_osRead));
        m_osRead.hEvent = CreateEvent(NULL, TRUE, FALSE, _T("ReadEvent"));
```

```
        ClearCommError(hCom, &dwErrorFlags, &comStat); //清除通讯错误，获得设备当前状态
        if (!comStat.cbInQue)return 0; //如果输入缓冲区字节数为 0，则返回 false
        BOOL bReadStat = ReadFile(hCom, //串口句柄
            buf, //数据首地址
            wCount, //要读取的数据最大字节数
            &wCount, //DWORD*,用来接收返回成功读取的数据字节数
            &m_osRead); //NULL 为同步发送，OVERLAPPED*为异步发送
        if (!bReadStat)
        {
            if (GetLastError() == ERROR_IO_PENDING) //如果串口正在读取中
            {
                //GetOverlappedResult 函数的最后一个参数设为 TRUE
                //函数会一直等待，直到读操作完成或由于错误而返回
                GetOverlappedResult(hCom, &m_osRead, &wCount, FALSE);
            }
            else
            {
                ClearCommError(hCom, &dwErrorFlags, &comStat); //清除通讯错误
                CloseHandle(m_osRead.hEvent); //关闭并释放 hEvent 的内存
                return 0;
            }
        }
        return wCount;
    }
}
```

至此，WzSerialPort 类已经可以在我的机器上被正常调用。我在用 WzSerialPort 类创建一个 remote 对象后，只需 remote.open(9600,0,8,1);即可打开蓝牙串口（此时要输入密码）。在程序中执行该语句后，蓝牙上的板载指示灯会由闪烁变为常亮。之后可以通过 remote.receive(buf，1024)来将串口消息读入 buf 指针所指向的地址，后一个参数为读入的字节数。同时也可以通过 remote.close 关闭串口，通过 remote.send 来发送消息。

至此我们已经实现了蓝牙通讯功能。接下来需要让 PC 端可以响应单片机通过蓝牙传递的消息。

4. PC 端配套应用程序。

首先是最核心的部分，即程序如何监听串口消息以响应单片机的消息触发实现 ALT+TAB 组合键，这一点及其容易实现。在打开串口的前提下（由 remoteIsOpen 这一变量记录），用上述 WzSerialPort 类 receive 成员函数接收字符串至 buf，然后再使用 strncmp 函数去比较 buf 和”CATCH!”两字符串的前 6 个字节即可实现效果。当然我希望在程序在触发一次 TAB 之后就先暂停监听串口消息（避免老板在传感器前乱晃导致我的电脑频繁切屏而露馅），所以我设置了一个 flag 变量来作为组合键触发的条件。

组合键的触发可以在包含了 windows.h 这一头文件后由 keybd_event 函数实现（如下面源码所示）

每次循环之后，要用 memset 及时刷新 buf 数组，避免残留数据影响程序执行。

```
if(remoteIsOpen){
        while(1){
            char buf[1024];
```

```
            remote.receive(buf,1024);
            if(!strncmp(buf,"CATCH!",6)&&flag){
                keybd_event(VK_MENU,0,0,0);
                keybd_event(VK_TAB,0,0,0);
                keybd_event(VK_TAB,0,KEYEVENTF_KEYUP,0);
                keybd_event(VK_MENU,0,KEYEVENTF_KEYUP,0);
                flag=0;
            }
            memset(buf,0,1024);
        }
    }
```

至此这个系统最为核心的部分已经完成，已经可以实现 Alt+Tab 的快速触发。但是仍存在一些使用上的不便，如串口名称的设置是预先嵌入到源码中的；程序不能反复使用，触发一次后就必须重启等弊端。为了方便用户使用，我给程序写了一个可视化的界面，方便与用户交互。

Text.cpp 为程序的主体，其中的源码我将分段呈现出来并一一解析

```
//-*- coding : GBK -*-

#include <iostream>
#include "tchar.h"
#include <string>
#include "WzSerialPort.cpp"
#include <windows.h>

using namespace std;

#define BUTTON_FLAG 1001
#define BUTTON_COM 1002
#define TEXT_COM 2001

HINSTANCE hg_app;
WzSerialPort remote;
HANDLE ATB=0;
HANDLE hMutex = NULL;
HWND Button_flag;
HWND Button_com;
HWND Text_com;

int flag=0;
int remoteIsOpen=0;
//int remoteIsOn=0;
char com_name[0x20]={0};

DWORD WINAPI AltTAB(LPVOID lpParameter);
```

```
typedef struct Thread_args{
    WzSerialPort * r;
}Thread_args;

LRESULT CALLBACK WindowProc(
    _In_  HWND hwnd,
    _In_  UINT uMsg,
    _In_  WPARAM wParam,
    _In_  LPARAM lParam
);
```

首先先从外部包含进来了一些必要的东西，例如和 win32 开发密切相关的 windows.h，先前提到的串口通信 WzSerialPort.cpp，以及处理宽字符串的 tchar.h 等。

宏定义为程序涉及到的控件的菜单句柄，旨在增加程序的可读性。

全局变量为 win32 编程中涉及到的一些控件句柄（按钮和文本框等），以及后续多线程编程中用到的子线程的句柄和用于确保不发生条件竞争错误而引如的互斥量 hMutex，以及串口通信的对象 remote 即后面的一些标志性的变量 flag 等。

Thread_args 被用于线程传参，传递的是 remote 的指针（写报告的时候发现这里有些冗余，这个结构体是之前失败尝试之后的遗留物，写报告的时候才发现没有必要，其实应该删除）

```
int CALLBACK WinMain(
    _In_  HINSTANCE hInstance,
    _In_  HINSTANCE hPrevInstance,
    _In_  LPSTR lpCmdLine,
    _In_  int nCmdShow
  )
{

    WNDCLASS wc = { };
    wc.style=CS_HREDRAW | CS_VREDRAW ;
    wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
    wc.lpfnWndProc = WindowProc;
    wc.lpszClassName =_T("My Class");
    wc.hInstance = hInstance;

    RegisterClass(&wc);

    HWND hwnd = CreateWindow(
        _T("My Class"),                    //类名，要和刚才注册的一致
        _T("摸鱼大师 Jmp.Cliff"),            //窗口标题文字
        WS_OVERLAPPEDWINDOW^WS_THICKFRAME,  //窗口外观样式,这里 WS_THICKFRAME 是为了禁止
调整大小
        38,                                //窗口相对于父级的 X 坐标
        20,                                //窗口相对于父级的 Y 坐标
        540,                               //窗口的宽度
```

```
        200,                                //窗口的高度
        NULL,                               //没有父窗口，为 NULL
        NULL,                               //没有菜单，为 NULL
        hInstance,                          //当前应用程序的实例句柄
        NULL);                              //没有附加数据，为 NULL
    if(hwnd == NULL)                        //检查窗口是否创建成功
        return 0;

    ShowWindow(hwnd, SW_SHOW);
    UpdateWindow(hwnd);
    FreeConsole();

    MSG msg;

    while(GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return 0;
}
```

WinMain 函数的作用是注册窗口，在注册窗口时，要注意我在外观样式上设置为了 WS_OVERLAPPEDWINDOW^WS_THICKFRAME，后面的异或运算的目的是为了禁用窗口大小调整，使得程序窗口的大小不能被用户拖动。窗口被设置了水平重绘和垂直重绘，同时关闭了控制台。

设置大多很常规化，这里不再赘述。

由于 WinMain 函数在最后必须进入消息机制处理的无限次循环中，不能通过调用 AltTab 函数来启动监听，故 Alt+Tab 组合键的触发功能被我创建为了一个线程函数以实现并发的效果。

AltTAB 函数的调用：

```
ATB=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)AltTAB,&tg,0,NULL);
```

AltTAB 函数的实现：

```
DWORD WINAPI AltTAB(LPVOID lpParameter){
    WaitForSingleObject(hMutex, INFINITE);  //互斥量
    Thread_args * tg=(Thread_args *)lpParameter;
    WzSerialPort * r=tg->r;
    if(remoteIsOpen){
        ReleaseMutex(hMutex);
        while(1){
            WaitForSingleObject(hMutex, INFINITE);  //互斥量
            char buf[1024];
            r->receive(buf,1024);
            if(!strncmp(buf,"CATCH!",6)&&flag){
                keybd_event(VK_MENU,0,0,0);
                keybd_event(VK_TAB,0,0,0);
```

```
                keybd_event(VK_TAB,0,KEYEVENTF_KEYUP,0);
                keybd_event(VK_MENU,0,KEYEVENTF_KEYUP,0);
                flag=0;
                SendMessage(Button_flag, WM_SETTEXT, NULL,(LPARAM)_T("Start!"));
            }
            memset(buf,0,1024);
            ReleaseMutex(hMutex);
        }
    }
}
```

该函数与之前在第 3 小节中涉及到的程序大体相仿。设置为了线程函数，故需要通过一个结构体指针进行传参以获得 remote 指针。同时，由于我在与该函数并发执行的消息处理函数 WindowProc 函数中也访问了 flag，remote 等变量，为了防止出现条件竞争错误而使得程序崩溃，我让线程在执行中必须等待获得一个互斥量 hMutex（针对 WindowProc 函数我也做了同样的处理），以确保线程函数的初始化和每一次触发 alt+tab 都能一气呵成（同时也确保线程函数不会在 WindowProc 进行某一次执行中途将其打断）

触发老板键后，flag 被置零，SendMessage 函数的作用是修改控件信息（令开关上的文字由 Stop 变为 Start）。

然后还需要去创建一个 WindowProc 函数作为注册窗口时的回调函数。其中消息由 uMsg 传递，通过 switch 进行分支处理。

```
LRESULT CALLBACK WindowProc(
    _In_  HWND hwnd,
    _In_  UINT uMsg,
    _In_  WPARAM wParam,
    _In_  LPARAM lParam
)
{
    WaitForSingleObject(hMutex, INFINITE);  //互斥量
    switch(uMsg)
    {
        case WM_DESTROY:
            {
                PostQuitMessage(0);
                return 0;
            }
```

WM_DESTORY 消息对应着程序的退出

```
        case WM_CREATE:
            {
                Button_flag=CreateWindow(_T("Button"),_T("Start!"),WS_VISIBLE |
WS_CHILD | BS_PUSHBUTTON,30,50,100,50,hwnd,(HMENU)BUTTON_FLAG,hg_app,NULL);
                Button_com=CreateWindow(_T("Button"),_T("打开串口"),WS_VISIBLE |
WS_CHILD | BS_PUSHBUTTON,180,50,100,50,hwnd,(HMENU)BUTTON_COM,hg_app,NULL);
                Text_com=CreateWindow(
                TEXT("EDIT"),TEXT(""),WS_VISIBLE|WS_CHILD|WS_BORDER|ES_AUTOHSCROLL,
320, 60, 140, 30, hwnd, (HMENU)TEXT_COM, hg_app,NULL);
```

```
            break;
        }
```

WM_CREATE 消息随着窗口的创建而发出这里要创建各类控件，包含两个按钮，一个用于启动/关闭 AltTAB 线程，一个用于设置串口号，还有一个文本框用于读入串口号。实现效果如下：

下面要给两个按钮按下后的消息添加对应的程序，这里对应控件的菜单句柄存在 wParam 参数的低字节位里面，我们要根据这个量来区分这个 WM_COMMAND 消息是由哪个按钮触发的。

```
    case WM_COMMAND:
        {
            switch(LOWORD(wParam))
            {
```

先看看 BUTTON_FLAG，即启动/关闭 AltTAB 线程的函数

```
            case BUTTON_FLAG:
                {
                    if(flag){
                        flag=0;
                        //remoteIsOn=0;
                        if(ATB)
                            CloseHandle(ATB);
                        else
                            exit(0);
                        ATB=0;
                        SendMessage(HWND(lParam), WM_SETTEXT,
NULL,(LPARAM)"Start!");
                    }else{

                        if(!remoteIsOpen){
                            MessageBox(hwnd, _T("请先打开串口"), _T("提示"),
MB_OK | MB_ICONINFORMATION);
                        }
                        if(ATB)
                            CloseHandle(ATB);
                        Thread_args tg;
                        tg.r=&remote;
                        flag=1;
```

```
                            //remoteIsOn=1;
                            ATB=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)AltTA
B,&tg,0,NULL);
                            SendMessage(HWND(lParam), WM_SETTEXT,
NULL,(LPARAM)_T("Stop"));
                        }
                        break;
                    }
```

首先 flag 记录了该线程目前的状态。如果 flag 为 1 说明线程正在不断监听串口消息，此时就应将 flag 置 0，并通过 CloseHandle 关闭该线程（ATB 为该线程的句柄），容易知道 ATB 为一个指针类型，所以在关闭前一定要检查该指针是否为 0，如果为 0 的话说明程序有异常——因为 ATB 为 0 的时候 flag 不应该为 1，此时应退出程序防止出现更严重的问题。成功关闭后要将 ATB 置零，同时更改按钮的文字为 Start，标志下一次按下将启动监视。

若 flag 为 0 说明线程未启动，这时要先检查串口是否打开，如未打开则弹窗提示用户打开串口。然后检查 ATB 是否为 0，如果不为 0 则先将其关闭（正常应该为 0）。随后创建线程参数结构体，启动线程并传参，将 flag 置为 1，同时更改按钮显示。

接下来，是关于串口设置按钮的事件处理。具体的实现方式如下面代码所示

```
                case BUTTON_COM:
                    {
                        if(flag){
                            MessageBox(hwnd, _T("请先点击 stop 停止监控"), _T("提示"),
MB_OK | MB_ICONINFORMATION);
                            break;
                        }
                        if(ATB)
                            CloseHandle(ATB);

                        memset(com_name,0,0x20);
                        GetDlgItemText(hwnd,TEXT_COM,com_name,0x20);    //这里有宽
字符串问题会报错，但是我这里运行起来毫无问题，可以忽略
                        remote.close();
                        if(remote.open(com_name,9600,0,8,1)){
                            remoteIsOpen=1;
                            MessageBox(hwnd, _T("成功打开串口，请您确定该串口为设备的
蓝牙串口"), _T("提示"), MB_OK | MB_ICONINFORMATION);
                        }else{
                            remoteIsOpen=0;
                                MessageBox(hwnd, _T("打开蓝牙串口失败！"),
_T("提示"), MB_OK | MB_ICONINFORMATION);
                             }
                    }
                }
                break;
            }
```

```
    }
    ReleaseMutex(hMutex);
    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}
```

为了预防某些未知的隐患，我们不应在程序正在监视时去更改串口，故 flag 为 1 时应该提示用户先点击 Stop 停止监控。如 flag 为 0，则读取文本框中的字符串信息，并尝试打开串口。并根据 open 的返回值给用户弹出状态提示窗口。且每一次打开串口前都会先执行 close 函数将先有的可能已经打开的串口关闭。

至此，程序已经完全设计完成。Text.cpp 的源码即上述代码组合而成。

四．写在后面

十分感谢老师能认真看完我这篇冗长的报告(截至此处这篇报告已有六七千字了，除去代码后的纯文本也有至少四千字)。一开始着手去设计这个系统的时候，我内心十分忐忑不安，毕竟我做的这个课程设计可不是什么正经的东西。但是想到老师上课时对我们以朋友相待，给我们发表自我观点的机会，想必老师不会因为这个设计略有些叛道离经，而让我挂科或者给我一个很低的分数。于是就放心大胆地完成了这次课程设计。作为一名转专业的学生，大一没有经历过原专业学生经历的两次编程语言课的课程设计，所以这还是我第一次做课设。第一次课设，不是按部就班地完成任务，而是一次自由大胆的设计和探索，对我来说，也算是不留遗憾了。

关于课设实现的效果——上班摸鱼——这件事，实际上我个人保持一个中立态度，作为接受过马克思主义教育的大学生，自然是要认同劳动光荣偷懒可耻的观念。但是我所做出的劳动，究竟有无价值？究竟有何种价值？是否于我付出的劳动相抵？是否会不利于我去创造更多的长期价值？这更值得我们深思。摸鱼多少会有些愧疚，但是成为被“压榨”地熬夜加班直到猝死这一事件的主角，我还是倾向前者。

所以，谨以此不完善的课程设计，作为未来的我，在生活重担之下的一个无奈的选择。

物联网工程导论，为期四周的课程，让我有了一次和老师面对面交流探讨的机会，这在我的大学生活中还是第一次。虽然说，在这种导论式的课程中，我没有学到多少具体的系统化的知识，但是却从老师那里收获了一些有指导性的学习思想和路线，这对我来说是无比珍贵的。

期待能在以后的课程中与老师再次相遇，想必到时候具备了一定专业基础知识的我，能在老师的引领下，再感受一场更为盛大的，思维火花的对撞——或许，我，就是某一次碰撞的中心。

五．参考资料

跟我一起玩 win32 开发系列：https://blog.csdn.net/tcjiaan/category_9261413.html

arduino 开发系列：https://blog.csdn.net/tonyiot/category_7777146.html

c++多线程：https://www.runoob.com/w3cnote/cpp-multithread-demo.html

蓝牙模块 HC-06 的基本设置和它的 AT 指令集：https://blog.csdn.net/qq_44986938/article/details/113359043

windows 纯 c++实现串口通信：https://blog.csdn.net/Touch_Dream/article/details/82915553