**MySTC\_B BSP\_Ver3.4**

**“STC\_B学习板”软件支持包**

**使用说明**

建议查看STCBSP中各模块头文件

头文件更新、更准确、更专业

湖南大学信息科学与工程学院

徐成

2021年9月5日更新

目录

1. STCBSP总体说明
2. 系统函数库Sys.Lib
3. 显示模块函数库Displayer.lib
4. 按键模块函数库Key.lib
5. 无源蜂鸣器函数库Beep.lib
6. 音乐模块函数库Music.lib
7. 霍尔传感器模块函数库Hall.lib
8. 振动传感器模块函数库Vib.lib
9. 模数转换模块函数库ADC.lib
10. 异步通信模块函数库Uart1.lib
11. 异步通信模块函数库Uart2.lib
12. 红外模块函数库IR.lib
13. 步进电机控制模块函数库StepMotor.lib
14. 实时时钟模块函数库DS1302.lib
15. 非易失性存储器模块函数库M24C02.lib
16. FM收音机模块函数库FM\_Radio.lib
17. EXT模块函数库（电子秤、超声波测距、编码器、PWM）EXT.lib
18. STCBSP测试Demo：STC\_DemoV3

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* STCBSP\_V3.5总体说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(1) 系统工作时钟频率可以在main.c中修改 SysClock赋值（单位Hz）。

如：code long SysClock=11059200; 定义系统工作时钟频率为11059200Hz（也即11.0592MHz）

系统工作频率必须与实际工作频率（下载时选择的）一致，以免与定时相关的所有功能出现误差或错误。

(2) 使用方法：

1，在工程中加载main.c文件和STC\_BSP.lib库文件

2，在main.c中选择包含以下头文件：

（如果要使用可选模块提供的函数和方法，就必须包含其头文件）

#include "STC15F2K60S2.H" //必须，"STC-B学习板"使用MCU指定的头文件

#include "sys.H" //必须，sys（MySTC\_OS调度程序）头文件

#include "display.H" //可选，display（显示模块）头文件。

#include "key.H" //可选，key（按键模块）头文件。

#include "hall.H" //可选，hall（霍尔传感器模块）头文件。

#include "Vib.h" //可选，Vib（振动传感器模块）头文件。 #include "beep.H" //可选，beep（蜂鸣器模块）头文件。 #include "music.h" //可选，music（音乐播放）头文件。

#include "adc.h" //可选，adc（热敏、光敏、导航按键、扩展接口ADC）头文件 #include "uart1.h" //可选，uart1（串口1通信）头文件。

#include "uart2.h" //可选，uart2（串口2通信）头文件。

#include "IR.h" //可选，38KHz红外通信

#include "stepmotor.h" //可选，步进电机

#include "DS1302.h" //可选，DS1302实时时钟

#include "M24C02.h" //可选，24C02非易失性存储器

#include "FM\_Radio.h" //可选，FM收音机

#include "EXT.h" //可选，EXT扩展接口

（电子秤、超声波测距、旋转编码器、PWM输出控制电机快慢和正反转）

3，MySTC\_Init()是sys初始化函数，必须执行一次；

MySTC\_OS()是sys调度函数，应置于while（1）循环中；

4，各可选模块如果选用，必须在使用模块其它函数和方法前执行一次模块所提供的驱动函数（设置相关硬件、并在sys中加载其功能调度）：

DisplayerInit(); //显示模块驱动

Key\_Init(); //按键模块驱动

BeepInit(); //蜂鸣器模块驱动

MusicPlayerInit(); //蜂鸣器播放音乐驱动

HallInit(); //霍尔传感器模块驱动

VibInit(); //振动传感器模块驱动

AdcInit(); //模数转换ADC模块驱动（含温度、光照、导航按键与按键Key3、EXT扩展接口上的ADC）

StepMotorInit(); //步进电机模块驱动

DS1302Init(); //DS1302实时时钟驱动

FMRadioInit(); //FM收音机驱动

EXTInit(); //扩展接口驱动（含电子秤、超声波测距、旋转编码器、PWM输出，但不含EXT上Uart2和与之相关应用）

Uart1Init(); //Uart1（串口1）驱动：USB上（与计算机通信）

Uart2Init(); //Uart2（串口2）驱动：485接口、或EXT扩展接口（多机通信、Uart方式模块如蓝牙模块）

IrInit(); //38KHz红外通信模块驱动

补充说明：有部分模块不需要驱动；

驱动函数有些有参数。

（具体见各模块头文件说明）

5，sys和各模块共提供以下事件：

numEventSys1mS: 1mS事件 （"1毫秒时间间隔到"事件）

enumEventSys10mS: 10mS事件 （"10毫秒时间间隔到"事件）

enumEventSys100mS: 100mS事件 （"100毫秒时间间隔到"事件）

enumEventSys1S: 1S事件 （"1秒时间间隔到"事件）

enumEventKey: 按键事件 （K1、K2、K3 三个按键有"按下"或"抬起"操作）

enumEventHall: 霍尔传感器事件 （霍尔传感器有"磁场接近"或"磁场离开"事件）

enumEventVib: 振动传感器事件 （振动感器检测到"振动"事件）

enumEventNav: 导航按键事件（导航按键5个方向、或按键K3 有"按下"或"抬起"操作）

enumEventXADC: 扩展接口上完成一次ADC转换事件（P1.0、P1.1采取到一组新数据）

enumEventUart1Rxd: Uart1收到了一个（数据包头匹配、数据包大小一致）数据包

enumEventUart2Rxd: Uart2收到了一个（数据包头匹配、数据包大小一致）的数据包

enumEventIrRxd: 红外接收器Ir上收到一个数据包

对以上这些事件，应采用"回调函数"方法响应（即用sys提供的SetEventCallBack()设置用户回调函数），以提高系统性能。

6，各可选模块提供的其它函数和具体使用方法请参见：

各模块头文件中的说明；

main.c提供的推荐程序框架和部分示例；

其它可能技术文档或应用示例

编写：徐成（电话18008400450） 2021年2月26日设计，2021年9月1日更新

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* sys Ver3.5 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(1) sys.c构成STC程序基本架构。提供：

a, 系统初始化MySTC\_Init()

系统调度函数MySTC\_OS()

加载用户回调函数SetEventCallBack()

b, 若干可设置和触发回调函数的事件；

enumEventSys1mS : 1mS定时到

enumEventSys10mS : 10mS定时到

enumEventSys100mS : 100mS定时到

enumEventSys1S : 1S定时到

enumEventKey : 按键事件

enumEventNav : 导航按键事件“

enumEventHall : hall传感器事件

enumEventVib : 振动传感器事件

enumEventXADC : 扩展接口上新的AD值事件

enumEventUart1Rxd : 串口1上收到一个符合格式定义的事件。

串口1：USB上与PC机通信

enumEventUart2Rxd : 串口2上收到一个符合格式定义的事件。

串口2：485、或EXT上（由初始化确定）

enumEventIrRxd : 红外收到一个数据包事件

c, 获取系统运行性能评价参数GetSysPerformance()

(2) MySTC\_Init()：sys初始化函数，必须执行一次。

(3) MySTC\_OS()：sys调度函数，应在while（1）循环中。

(4) SetEventCallBack(char event, void \*(user\_callback)):加载"事件"用户回调函数.

(5) SysPerF GetSysPerformance(void) :获取系统运行性能评估参数

函数参数：无

函数返回值：结构 struct SysPerF。定义如下：

typedef struct //系统性能评估参数,每秒更新一次

{ unsigned long MainLoops; //每秒主循环次数（应大于1000以上）

unsigned char PollingMisses; //每秒轮询丢失次数（理想值为0）

} SysPerF;

(6) 补充说明：

sys.c 基本调度时间为1mS，非抢占式，要求用户程序片段，其单次循环执行时间累加起来应小于1mS。

编写：徐成（电话18008400450） 2021年2月26日设计，2021年8月31日更新

\*/

#ifndef \_sys\_h\_

#define \_sys\_h\_

typedef struct //系统性能评估参数,每秒更新一次

{ unsigned long MainLoops; //每秒主循环次数（应大于1000以上）

unsigned char PollingMisses; //每秒轮询丢失次数（理想值为0）

} SysPerF;

extern void MySTC\_Init(); //sys初始化函数

extern void MySTC\_OS(); //sys调度函数，应在while（1）循环中

extern void SetEventCallBack(char event, void \*(user\_callback)); //加载"事件"用户回调函数

extern SysPerF GetSysPerformance(void);

enum event{enumEventSys1mS, //系统1mS事件

enumEventSys10mS, //系统10mS事件

enumEventSys100mS, //系统100mS事件

enumEventSys1S, //系统1S事件

enumEventKey, //按键事件

enumEventHall, //hall传感器事件

enumEventVib, //振动传感器事件

enumEventNav, //导航按键事件“

enumEventXADC, //扩展接口上新的AD值事件

enumEventUart1Rxd, //串口1上收到一个符合格式定义的事件

enumEventUart2Rxd, //串口2上收到一个符合格式定义的事件

enumEventIrRxd //红外Ir上收到一个数据包

};

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* display V2.0 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

displayer用于控制“STC-B学习板”上8个7段数码管（Seg7）和8个指示灯（Led）工作.提供显示模块加载和三个应用函数：

1. displayerInit()：显示模块加载函数；

(2) SetDisplayerArea(char Begin\_of\_scan,char Ending\_of\_Scan)：设置LED启用区域。8个数码管从左至右编号分别为0——7，

函数参数：

Begin\_of\_scan设定启用数码管起始编号，Ending\_of\_Scan为结束编号。设定范围内的数码管才工作和显示。

注：正常情况下，两个参数取值范围为0——7，且Ending\_of\_Scan>Begin\_of\_scan。但利用动态扫描和人眼视觉效果，可设置超出

该范围的参数，以实现特殊显示效果：如软件调整显示亮度，或非灯亮度显示，等；

(3) Seg7Print(char d0,char d1,char d2,char d3,char d4,char d5,char d6,char d7)：将8个参数值分别译码显示到对应的数码管上。

显示译码表(code char decode\_table[])在main.c中,用户可以修改和增减.

(4) LedPrint(char led\_val)：控制8个指示灯开关。参数light\_val的8个bit位对应8个指示灯的开关，“1”——指示灯“亮”

编写：徐成（电话18008400450） 2021年2月26日设计，2021年3月15日更新

\*/

#ifndef \_displayer\_H\_

#define \_displayer\_H\_

extern void DisplayerInit();

extern void SetDisplayerArea(char Begin\_of\_scan,char Ending\_of\_Scan);

extern void Seg7Print(char d0,char d1,char d2,char d3,char d4,char d5,char d6,char d7);

extern void LedPrint(char led\_val);

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* key V2.0 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Key模块用于获取“STC-B学习板”上三个按键的状态。提供按键模块加载和一个应用函数,一个“按键事件：enumEventKey：

1. Key\_Init()：按键模块加载函数；

(2) char GetKeyAct(char Key)：获取按键状态。

函数参数：Key，指定要获取状态的按键。Key取值：

enumKey1

enumKey2

enumKey3

（当参数取值超出有效范围，函数将返回fail）

函数返回值：

enumKeyNull（无按键动作）

enumKeyPress（按下）

enumKeyRelease（抬起）

enumKeyFail（失败）

返回值是经过多次检测按键实时状态和统计检测结果后（软件消抖）的有效事件。

每个按键查询一次后,事件值变成enumKeyNull。事件值仅查询一次有效。

(3) 按键事件：enumEventKey

当三个按键（enumKey1,enumKey2,enumKey3）中任意一个按键有”按下“或”抬起“动作时，将产生一个”按键事件“，响应按键事件的用户处理函数由用户编写,并有sys中提供的SetEventCallBack函数设置.

补充说明：

如果启用了ADC模块，按键3（Key3）任何操作在本模块不可检测到和有任何信息反应，这时按键3（Key3）任何操作将在ADC模块中检测和反应。使用方法相同，具体见ADC模块说明。

编写：徐成（电话18008400450） 2021年3月5日设计，2021年8月26日更新

\*/

#ifndef \_key\_H\_

#define \_key\_H\_

extern void Key\_Init();

extern unsigned char GetKeyAct(char Key) ;

enum KeyName {enumKey1,enumKey2,enumKey3};

enum KeyActName {enumKeyNull,enumKeyPress,enumKeyRelease,enumKeyFail};

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Beep V2.0 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Beep用于控制“STC-B学习板”上无源蜂鸣器的发声。Beep模块共提供1个驱动函数、2个应用函数：

1. BeepInit()：蜂鸣器模块驱动函数

(2) Set\_Beep(unsigned int Beep\_freq, unsigned char Beep\_time)：控制蜂鸣器发声；

函数参数：

Beep\_freq：指定发声频率，单位Hz。小于<10 Hz, 不发音

Beep\_time：指定发声时长。发声时长＝10\*Beep\_time (mS) ，最长 655350mS

函数返回值：

enumSetBeepOK：调用成功，

enumSetBeepFail：调用失败（或因蜂鸣器正在发音

(3) GetBeepStatus(void): 获取Beep当前状态

函数返回值：

enmuBeepFree:空闲

enumBeepBusy ,正在发音

(4) Beep模块使用了STC内部CCP模块1通道

编写：徐成（电话18008400450） 2021年3月3日设计，2021年3月26日更新

\*/

#ifndef \_beep\_H\_

#define \_beep\_H\_

extern void BeepInit();

extern char SetBeep(unsigned int Beep\_freq, unsigned int Beep\_time);

extern unsigned char GetBeepStatus(void);

enum BeepActName {enumBeepFree=0,enumBeepBusy,enumSetBeepOK,enumSetBeepFail};

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* music V2.0 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Music模块在Beep和Displayer模块基础上再次封装，用于控制“STC-B学习板”上播放音乐。加载该模块，需同时加载Beep模块、displayer模块。music模块共提供1个music驱动函数、4个应用函数：

1. MusicPlayerInit()：驱动music模块；

（2）char PlayTone(unsigned char tone, unsigned char beatsPM ,unsigned char scale, unsigned char beats)：播放音乐音阶，实现用指定音调、指定节拍率、发指定音阶、发音节拍数。

函数参数：

tone：指定音调，有效值：0xFA、0xFB、0xFC、0xFD、0xFE、0xFF、0xF9分别对应音乐A、B、C、D、E、F、G调

beatsPM：节拍率，即每分钟节拍数，值范围10~255拍/分钟

scale：音乐简谱音高，1字节。

0x00——休止符

高4位：1——低8度音 2——中8度音 3——高8度音

低3位：1-7对应简谱音。其它值无效。

如：0x13表示低音3（mi）

beats：音长(节拍数),单位1/16拍。

如：16（0x10）对应1拍，32（0x20）对应2拍，8（0x08）对应半拍...... 。

函数返回值：enumBeepOK：调用成功

enumBeepBusy：忙（上一音未按设定发完，或因蜂鸣器正在发音）

enumBeepFail：调用失败（音调参数tone不对，或音高编码scale不对）

（见Beep.h中定义BeepActName）

（3）SetMusic(unsigned char beatsPM, unsigned char tone, unsigned char \*pt, unsigned int datasize, unsigned char display)：设定或改变要播放音乐和播放参数。

函数参数：

beatsPM：节拍率，即每分钟节拍数，值范围10~255拍/分钟，如果参数值为“enumModeInvalid”将不改变原beatsPM；

tone：指定音调， 有效值：0xFA、0xFB、0xFC、0xFD、0xFE、0xFF、0xF9分别对应音乐A、B、C、D、E、F、G调，参数值为“enumModeInvalid”或其它值将不改变原tone

\*pt ：指向要播放的音乐编码的首地址

datasize：要播放的音乐编码的长度（字节数）

说明：\*pt 和 datasize 只有有一个参数 = enumModeInvalid，将不改变 \*pt 和 datasize

display：Seg7和Led是否用来显示播放音乐信息，取值：

enumMscNull —— 不用

enumMscDrvSeg7 —— 用7段数码管Seg7（显示信息）

enumMscDrvLed —— 用Led指示灯（打拍） enumMscDrvSeg7andLed —— 用Seg7 和 Led

参数值为“enumModeInvalid”或其它值将不改变原display

音乐编码规则：

1，常规音乐简谱发音编码（成对出现，不可分开，中间不能插入其它编码和控制字）

基本格式：音高（1字节），节拍数（1字节），音高，节拍，......

其中“音高”部分：

0x11 — 0x17 ：对应低音 do、re、mi、fa、so、la、si、 0x21 — 0x27 ：对应中音 do、re、mi、fa、so、la、si

0x31 — 0x37 ：对应高音 do、re、mi、fa、so、la、si

其中“节拍数”部分：

0x01-0xFF：单位1/16拍。也即十六进制中，高4位表示整拍数，低4位表示分拍数（1/16）

如：发音2拍： 0x20

发音半拍： 0x08

发音1拍半：0x18

2，音乐编码中可以插入以下控制字，用于设定音乐播放参数等（前6个也可以通过函数，用程序设定和实现）:

enumMscNull ： 不用

enumMscDrvSeg7 ： 用7段数码管Seg7（显示信息）

enumMscDrvLed ： 用Led指示灯（打拍）

enumMscDrvSeg7andLed ： 用Seg7 和 Led

enumMscSetBeatsPM ： 设置节拍率， 后面再跟 节拍率（1字节）

enumMscSetTone ： 设置音调， 后面再跟 音调（1字节）

0xFA 或 0xFB 或 0xFC 或 0xFD 或 0xFE 或 0xFF 或 0xF9

分别对应音乐：A调 或B调………

enumMscRepeatBegin ： 设置音乐播放重复开始处。重复一次（暂不支持多次），暂不能嵌套（嵌套无效或可能导致不可预期结果）

enumMscRepeatEnd ： 设置音乐播放重复结束处

（4）SetPlayerMode(unsigned char play\_ctrl) ：音乐播放控制函数。

函数参数：

play\_ctrl：enumModePlay ： ”播放“

enumModePause ： ”暂停“

enumModeStop ：”停止/结束“

（其它参数无效）

所有操作在当前”音“播放完成后生效；

（5）char GetPlayerMode(void)： 获取当前播放状态

函数返回值：（play\_ctrl值）

enumModePlay ： 播放状态

enumModePause ： 暂停状态

enumModeStop ： 停止/结束

（其它功能应用型函数可根据需要设置和增加）

编写：徐成（电话18008400450） 2021年3月5日设计，2021年3月26日更新

\*/

#ifndef \_music\_H\_

#define \_music\_H\_

extern void MusicPlayerInit();

extern char PlayTone(unsigned char tone, unsigned char beatsPM ,

unsigned char scale, unsigned char beats);

extern void SetPlayerMode(unsigned char play\_ctrl);

extern char GetPlayerMode(void); //获取当前播放状态

enum PlayerMode {enumModeInvalid=0, //播放模数非法

enumModePlay, //播放

enumModePause, //暂停（可恢复续放）

enumModeStop}; //停止（结束）

enum MusicKeyword {enumMscNull=0xF0, //播放音乐时，不用7段数码管、LED指示灯

（显示音乐播放相关信息）

enumMscDrvSeg7, //播放音乐时， 用7段数码管

enumMscDrvLed, //播放音乐时， 用 LED指示灯

enumMscDrvSeg7andLed, //播放音乐时 用7段数码管、LED指示灯

enumMscSetBeatsPM, //音乐编码中关键字：设置 音乐节拍

enumMscSetTone, //音乐编码中关键字：设置 音调

enumMscRepeatBegin, //音乐编码中关键字： 设置 重复开始

enumMscRepeatEnd}; //音乐编码中关键字： 设置 重复结束

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* hall V2.0 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Hall模块用于获取“STC-B学习板”上hall传感器状态。hall模块共提供1个加载函数、1个应用函数，一个Hall事件：enumEventHall

1. HallInit()：hall模块初始化函数

（2）unsigned char GetHallAct(void)：获取hall事件。

函数返回值：

enumHallNull（无变化）

enumHallGetClose（磁场接近）

enumHallGetAway（磁场离开）

查询一次后,事件值变成enumEventHall (仅查询一次有效)

（3）hall传感器事件：

当Hall检测到有"磁场接近"或"磁场离开"事件时，将产生一个Hall传感器事件(enumEventHall).响应事件的用户处理函数由用户编写, 并有sys中提供的SetEventCallBack()函数设置事件响应函数.

编写：徐成（电话18008400450） 2021年3月15日设计 2021年3月26日修改

\*/

#ifndef \_hall\_H\_

#define \_hall\_H\_

extern void HallInit(void);

extern unsigned char GetHallAct(void);

enum HallActName {enumHallNull,enumHallGetClose,enumHallGetAway};

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 振动传感器SV V2.0 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SV模块用于获取"STC-B学习板"上Vib传感器状态.提供一个模块加载函数和一个应用函数,一个Vib事件enumEventVib：

1. VibInit()：振动传感器Vib模块初始化函数；

(2) char GetVibAct()：获取Vib事件。

函数返回值：

enumVibNull——无，

enumVibQuake——发生过振动

查询一次后,事件值变成enumVibNull，(仅查询一次有效)

(3) Vib传感器事件enumEventVib：

当Vib检测到有”振动“事件时，将产生一个”振动事件“，响应事件的用户处理函数由用户编写,并有sys中提供的SetEventCallBack()函数设置振动事件用户处理函数.

编写：徐成（电话18008400450） 2021年3月5日设计，2021年3月26日更新

\*/

#ifndef \_Vib\_H\_

#define \_Vib\_H\_

extern void VibInit();

extern unsigned char GetVibAct(void) reentrant;

enum VibActName {enumVibNull,enumVibQuake};

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC V1.1 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ADC模块用于“STC-B学习板”上与ADC相关电路:温度Rt、光照Rop、导航按键Nav、扩展接口EXT上的ADC转换。提供ADC模块初始化函数、2个应用函数,2个事件：

(1) AdcInit(char ADCmodel)：ADC模块初始化函数；

函数参数: ADCmodel选择扩展接口EXT是否用作ADC功能，

取值：enumAdcincEXT : 含对扩展接口EXT设置ADC功能

（EXT上P1.0、P1.1不可作数字IO功能使用）

enumAdcexpEXT : 不含对扩展接口EXT设置ADC功能

（EXT上P1.0、P1.1可作数字IO功能使用）

(2) struct\_ADC GetADC()：获取ADC值。

函数参数：无

函数返回值：返回数据结构 struct\_ADC。

数据结构定义：

typedef struct //ADC转换结果

{ unsigned int EXT\_P10; // 扩展接口EXT上P1.0脚ADC（10bit）

unsigned int EXT\_P11; // 扩展接口EXT上P1.1脚ADC（10bit）

unsigned int Rt; // 热敏电阻上ADC（10bit）

unsigned int Rop; // 光敏电阻上ADC（10bit）

unsigned int Nav; // 导航按键上ADC（10bit）

} struct\_ADC;

说明1：每个数字表示VCC/1024（伏），其中VCC为供电电压（USB一般为5V左右）

说明2：对于导航按键，下面GetAdcNavAct函数输出消抖后导航按键事件和状态，更方便使用；

(3) char GetAdcNavAct(char Nav\_button)：获取导航按键（包含K3）状态

函数参数：Nav\_button：指定要获取状态的导航按键。取值：

enumAdcNavKey3（K3键）,

enumAdcNavKeyRight（右按）,

enumAdcNavKeyDown（下按）,

enumAdcNavKeyCenter（中心按）

enumAdcNavKeyLeft（左按）,

enumAdcNavKeyUp（上按）.

函数返回值：返回当前按键事件，返回值：（同Key模块GetAdcKeyAct（）返回值）

enumKeyNull（无按键动作）,

enumKeyPress（按下）,

enumKeyRelease（抬起）,

enumKeyFail（失败） （函数参数取值不在有效范围）

返回值是经过多次检测按键实时状态和统计检测结果后（软件消抖）的有效事件。

每个按键查询一次后,事件值变成 enumKeyNull (仅查询一次有效)

(4) 导航按键事件“：enumEventNav

当导航按键5个方向或按键K3有任意”按下“或”抬起“动作时，将产生一个”导航按键事件“enumEventNav。响应导航按键事件的用户回调函数由用户编写,并由sys提供的SetEventCallBack()函数设置响应函数.

(5) 扩展接口EXT上P1.0、P1.1两个端口有新的AD值事件“：enumEventXADC

当ADC模块对P1.0、P1.1进行ADC转换，获得了它们新的ADC结果时，将产生enumEventXADC事件，通知用户进行处理。响应enumEventXADC事件的用户回调函数由用户编写,并有sys提供的SetEventCallBack()函数设置响应函数.

ADC模块对P1.0、P1.1进行ADC转换速度为3mS，也即每3mS或每秒钟333次转换.

(6) 补充说明：

a：对EXT上P1.0、P1.0的转换速度为3mS，也即每秒钟提供333次转换结果，提供了有新转换结果事件：enumEventXADC，方便用户处理

b：对于Rt、Rop，转换速度为9mS，也即每秒钟提供111次转换结果。没有提供相应“事件”，用户随时用函数GetAdcResult()查询和使用

c：对导航按键进行了软件消抖处理，最快可支持导航按键每秒12次操作速度，提供了导航按键发生了操作事件：enumEventNav

d：由于导航按键与K3键共用了单片机同一个端口（P1.7），启用ADC模块后，P1.7口IO功能失效，只能用GetAdcNavAct(char Nav\_button)函数获取K3按键的事件或状态。

e: STC-B板上Rt型号为：10K/3950 NTC热敏电阻，光敏电阻Rop型号为：GL5516. 它们的ADC值与温度、光照强度关系请参阅它们的数据手册与STC-B电路图进行换算。

编写：徐成（电话18008400450） 2021年3月25日设计,2021年8月30日改进

\*/

#ifndef \_adc\_H\_

#define \_adc\_H\_

typedef struct //ADC转换结果

{ unsigned int EXT\_P10; // 扩展接口EXT上P1.0脚ADC

unsigned int EXT\_P11; // 扩展接口EXT上P1.1脚ADC

unsigned int Rt; // 热敏电阻上ADC

unsigned int Rop; // 光敏电阻上ADC

unsigned int Nav; // 导航按键上ADC

} struct\_ADC;

extern void AdcInit(char ADCmodel);

enum ADCmodel\_name {enumAdcincEXT=0x9B, //ADC模式：ADC时包含扩展接口

enumAdcexpEXT=0x98}; // 不包含扩展接口

extern struct\_ADC GetADC();

extern unsigned char GetAdcNavAct(char Nav\_button);

//获取导航按键5个方向（右、下、中心、左、上）操作，以及按键K3操作的事件

//返回值：enuKeyNull, enuKeyPress, enuKeyRelease, enuKeyFail

enum KN\_name {enumAdcNavKey3=0, //导航按键：按键K3

enumAdcNavKeyRight, //导航按键：右

enumAdcNavKeyDown, //导航按键：下

enumAdcNavKeyCenter, //导航按键：中心

enumAdcNavKeyLeft, //导航按键：左

enumAdcNavKeyUp}; //导航按键：上

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Uart1串行通信模块 V1.1 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Uart1模块提供Uart1模块初始化函数、3个应用函数,1个事件（enumEventRxd）：

1. Uart1Init(unsigned int band)：Uart1模块初始化函数。

函数参数：unsigned int band：定义串口1的通信波特率（单位：bps）

（固定：8个数据位、1个停止位，无奇偶校验位）

函数返回值：无

1. void SetUart1Rxd ( char \*RxdPt,

unsigned int Nmax,

char \*matchhead,

unsigned int matchheadsize);

设置串口1接收参数：数据包存放位置、大小，包头匹配字符、匹配字符个数。

收到符合条件的数据包时将产生enumEventRxd事件。

函数参数： char \*RxdPt： 指定接收数据包存放区（首地址）

unsigned int Nmax： 接收数据包大小（字节数），最大65535

char \*matchhead 需要匹配的数据包头（首地址）

unsigned int matchheadsize： 需要匹配的字节数

补充说明：

a, Nmax=1：为单字节接收，即收到一个字节就产生enumUart1EventRxd事件（如果定义了匹配，还需满足匹配条件）；

b, 0 < matchheadsize < Nmax：要求接收数据中连续matchheadsize个字节与\*matchhead处数据完全匹配，才在收到Nmax数据时产生enumEventRxd事件；

matchheadsize = Nmax：设定接收数据包完全匹配

matchheadsize=0 或 matchheadsize > Nmax：将不做匹配，接收到任意Nmax数据时产生enumEventRxd事件；

c, 在enumEventRxd事件发出后，接收到的数据包应及时使用或取出，收到下一个数据时将破坏和覆盖前面收到的数据包

函数返回值：无

(3) char Uart1Print(void \*pt, unsigned int num)：发送数据包，非阻塞函数（即函数不等到所设定任务全部完成才返回），该函数从被调用到返回大约1uS左右时间。

函数参数： void \*pt ：指定发送数据包位置

unsigned int num：发送数据包大小；

函数返回值：enumTxOK：调用成功，即所设定的发送数据包请求已被系统sys正确接受，sys将尽硬件资源最大可能及时发送数据。

enumTxFailure：调用失败（主要原因是：串口正忙（上一数据包未发完）

补充说明：

串口上发送1个字节数据大约需要时间0.1mS～10mS（视所设定的波特率），对计算机来说，如果发送多个字节是一个很要时间才能完成的事。类似于前面用蜂鸣器演奏音乐，对这类事件与程序"异步"的问题，编程时不仅要注意程序逻辑性、还有注意程序时效性。（这个问题其实总是要注意、必须要注意的）

(4) char GetUart1TxStatus(void)： 获取Uart1发送状态

函数参数：无

函数返回值：enumUart1TxFree:串口1发送空闲

enumUart1TxBusy,串口1发送正忙

(5) Uart1接收事件：enumEventUart1Rxd。表示收到了一个符合指定要求（数据包头匹配、数据包大小一致）的数据包。

补充说明：

(1) 串口1、串口2波特率可独立设置，互不影响.

(2) 串口1、串口2、红外通信可同时工作，互不影响

(3) 串口1、串口2用法基本上完全一致，红外通信用法也基本相同。不同地方是：

a，串口1固定在USB接口上，可用于与计算机通信；而串口2可初始化在EXT扩展、或485接口上（在485接口上时仅单工工作）；

b，红外通信速率固定不可变（大约相当于500～800 bps），通信时没有包头匹配功能。红外模块除通信功能外，还提供用于电器红外遥控的应用函数；

c，红外通信模块仅为单工工作。不发送时自动进入接收状态；有数据发送时自动进入发送状态，但正在接收数据包过程中不会进入发送状态。

编写：徐成（电话18008400450） 2021年3月28日设计 2021年9月1日更新

\*/

#ifndef \_uart1\_H\_

#define \_uart1\_H\_

extern void Uart1Init(unsigned int band); //串口初始化，参数：波特率

extern void SetUart1Rxd ( void \*RxdPt,

unsigned int Nmax,

void \*matchhead,

unsigned int matchheadsize);

//设置接收条件：数据包存放位置、大小，

包头匹配字符、匹配字符个数。

符合条件的包将产生enumEventRxd事件

extern char Uart1Print(void \*pt, unsigned int num);

//发送数据包。非阻塞函数。数据包位置、大小。

返回值：enumTxOK调用成功，

enumTxFailure失败（串口忙，上一数据包未发完）

extern char GetUart1TxStatus(void);

//获取串口1发送状态,

返回值：enumUart1TxFree:串口1发送空闲,

enumUart1TxBusy,串口1发送正忙

enum Uart1ActName { enumUart1TxFree=0,

enumUart1TxBusy,

enumUart1TxOK,

enumUart1TxFailure};

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Uart2串行通信模块 V1.0 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Uart2模块提供Uart2模块初始化函数、3个应用函数,1个事件（enumUart2EventRxd）：

1. void Uart2Init(unsigned int band,unsigned char Uart2mode)：Uart2模块初始化函数

函数参数：unsigned int band：定义串口2的通信波特率（单位：bps）

（固定8个数据位、1个停止位，无奇偶校验位）

unsigned char Uart2mode：定义串口2位置

取值：Uart2UsedforEXT —— 串口2在EXT扩展插座上

Uart2Usedfor485 —— 串口2用于485通信

（485为半双工。发送数据包时不能接收数据）

函数返回值：无

1. void SetUart2Rxd ( char \*RxdPt,

unsigned int Nmax,

char \*matchhead,

unsigned int matchheadsize);

设置串口2接收参数：数据包存放位置、大小，包头匹配字符、匹配字符个数。

收到符合条件的数据包时将产生enumUart2EventRxd事件。

函数参数： char \*RxdPt： 指定接收数据包存放区（首地址）

unsigned int Nmax： 接收数据包大小（字节数），最大65535

char \*matchhead： 需要匹配的数据包头（首地址）

unsigned int matchheadsize： 需要匹配的字节数

补充说明：

a, Nmax=1：为单字节接收，即收到一个字节就产生enumUart2EventRxd事件（如果定义了匹配，还需满足匹配条件）；

b, 0 < matchheadsize < Nmax：要求接收数据中连续matchheadsize个字节与\*matchhead处数据完全匹配，才在收到Nmax数据时产生enumUart2EventRxd事件；

matchheadsize = Nmax：设定接收数据包完全匹配

matchheadsize=0 或 matchheadsize > Nmax：将不做匹配，接收到任意Nmax数据时产生enumUart2EventRxd事件；

c, 在enumUart2EventRxd事件发出后，接收到的数据包应及时使用或取出，收到下一个数据时将破坏和覆盖前面收到的数据包

函数返回值：无

1. char Uart2Print(void \*pt, unsigned int num)：发送数据包，非阻塞函数（即函数不等到所设定任务全部完成才返回），该函数从被调用到返回大约1uS左右时间。

函数参数： void \*pt ：指定发送数据包位置

unsigned int num：发送数据包大小；

函数返回值：enumUart2TxOK：调用成功，即所设定的发送数据包请求已被系统sys正确接受，sys将尽硬件资源最大可能及时发送数据。

enumUart2TxFailure：调用失败（主要原因是：串口正忙（上一数据包未发完）

补充说明：

串口上发送1个字节数据大约需要时间0.1mS～10mS（视所设定的波特率），对计算机来说，如果发送多个字节是一个很要时间才能完成的事。类似于前面用蜂鸣器演奏音乐，对这类事件与程序"异步"的问题，编程时不仅要注意程序逻辑性、还有注意程序时效性。（这个问题其实总是要注意、必须要注意的）

1. char GetUart2TxStatus(void)： 获取Uart2发送状态

函数参数： 无

函数返回值：enumUart2TxFree:串口2发送空闲

enumUart2TxBusy,串口2发送正忙

(5) Uart2接收事件：enumUart2EventRxd。表示收到了一个符合指定要求（数据包头匹配、数据包大小一致）的数据包。

补充说明：

(1) 串口1、串口2波特率可独立设置，互不影响.

(2) 串口1、串口2、红外通信可同时工作，互不影响

(3) 串口1、串口2用法基本上完全一致，红外通信用法也基本相同。不同地方是：

a，串口1固定在USB接口上，可用于与计算机通信；而串口2可初始化在EXT扩展、或485接口上（在485接口上时仅单工工作）；

b，红外通信速率固定不可变（大约相当于500～800 bps），通信时没有包头匹配功能。红外模块除通信功能外，还提供用于电器红外遥控的应用函数；

c，红外通信模块仅为单工工作。不发送时自动进入接收状态；有数据发送时自动进入发送状态，但正在接收数据包过程中不会进入发送状态。

编写：徐成（电话18008400450） 2021年3月28日设计

\*/

#ifndef \_uart2\_H\_

#define \_uart2\_H\_

extern void Uart2Init(unsigned int band,unsigned char Uart2mode); //串口2初始化

enum Uart2PortName { Uart2UsedforEXT=00,

Uart2Usedfor485};

extern void SetUart2Rxd ( void \*RxdPt,

unsigned int Nmax,

void \*matchhead,

unsigned int matchheadsize);

//设置接收条件：数据包存放位置、大小，包头匹配字符、匹配字符个数。

符合条件的包将产生enumUart2EventRxd事件

extern char Uart2Print(void \*pt, unsigned int num);

//发送数据包。非阻塞函数。数据包位置、大小。

返回值: enumUart2TxOK调用成功，

enumUart2TxFailure失败（串口忙，上一数据包未发完）

extern char GetUart2TxStatus(void);

//获取串口2发送状态,

返回值: enumUart2TxFree:串口2发送空闲,

enumUart2TxBusy,串口2发送正忙

enum Uart2ActName { enumUart2TxFree=0,

enumUart2TxBusy,

enumUart2TxOK,

enumUart2TxFailure };

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* IR V1.0 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IR模块用于控制“STC-B学习板”上红外发送与接收控制，支持PWM、PPM红外编码协议的发送，PWM红外编码的接收，可用于制作红外遥控器、红外通信等。

IR模块已不与串口通信（uart和uart2）冲突，可用与它们同时工作。（以前冲突）

IR模块提供1个驱动函数、5个API应用函数、1个红外接收事件（enumEventIrRxd：红外Ir上收到一个数据包）。

(1) void IrInit(unsigned char Protocol)：IR模块初始化函数。

函数参数：unsigned char Protocol，定义红外协议。

Protocol 暂仅提供取值：NEC\_R05d

（定义红外协议基本时间片时长 = 13.1\*Protocol uS）

函数返回值：无

1. char IrTxdSet(unsigned char \*pt,unsigned char num)：以自由编码方式控制IR发送。

可用于编写任意编码协议的红外发送，如各种电器红外遥控器等

函数参数：unsigned char \*pt，指向待发送红外编码数据的首地址。编码规则如下：

码1红外发送时长，码1红外发送停止时长

，码2红外发送时长，码2红外发送停止时长

......

......

，码n红外发送时长，码n红外发送停止时长

unsigned char num，待发送红外编码数据的大小（字节数）

函数返回值：enumIrTxOK：调用成功，即所设定的发送数据包请求已被系统sys正确接受，sys将尽硬件资源最大可能及时发送数据。

enumIrTxFailure：调用失败（主要原因是：红外发送正忙（上一数据包未发完）、或红外正在接收一个数据包进行中

（同IrPrint()函数返回值）

补充说明：

编码数据单位=协议基本时间片的个数值，最大255。如当协议基本时间片为0.56mS时，数值1代表0.56mS时长，3代表1.68mS时长，.....

参照该格式定义和电器遥控器编码格式，可实现任何38KHz红外遥控器功能。

1. char IrPrint(void \*pt, unsigned char num)：以NEC的PWM编码方式发送数据包。

函数参数：void \*pt ：指定发送数据包位置

数据包不含引导码、结束码信息，仅待发送的有效数据

unsigned char num：发送数据包大小（字节数，不含引导码、结束码）

函数返回值：

enumIrTxOK：调用成功，即所设定的发送数据包请求已被系统sys正确接受，sys将尽硬件资源最大可能及时发送数据。

enumIrTxFailure：调用失败（主要原因是：红外发送正忙（上一数据包未发完）、或红外正在接收一个数据包进行中

（同IrTxdSet()函数返回值）

补充说明：

1，发送数据格式为：a+b+c

a，引导码： 发（16\*基本时间片），停（8\*基本时间片）

当基本时间片为0.56mS时：发9mS、停4.5mS

b，数据编码： "0" -- 发（1\*基本时间片），停（1\*基本时间片）

"1" -- 发（1\*基本时间片），停（3\*基本时间片）

先发高位、后发低位

c，结束码： 发（1\*基本时间片），停（1\*基本时间片）

可用于符合该函数发送格式的部分电器遥控器；

与GetIrRxNum()、SetIrRxd()配合，可进行红外双机通信，等用途。

2，非阻塞函数，该函数从被调用到返回大约1uS左右时间,但所指定的数据经红外发送完毕则需要较长时间（每字节大约需要10mS量级时间）。

3，IrPrint()函数用法完全类似与uart模块的Uart1Print()和Uart2Print()用法，（仅num参数为unsigned char，可参照使用）

1. SetIrRxd(void \*RxdPt)：设置红外接收数据包存放位置。

收到一个数据包时将产生numEventIrRxd事件

与它机IrPrint()函数配合，可实现红外数据通信

函数参数： void char \*RxdPt：指定接收数据包存放区（首地址）

函数返回值：无

1. unsigned char GetIrRxNum(void)：获取红外接收数据包的大小（字节数）

当收到一个数据包的numEventIrRxd事件产生后，

可用该函数获取红外接收数据包的大小（字节数）。

其它时间访问，其值不确定。

与SetIrRxd()配合，可实现红外数据包接收。

（它机应使用IrPrint()函数发送数据包）

函数参数： 无

函数返回值：红外接收数据包大小（字节数）。

(6) char GetIrStatus(void)： 获取Ir状态

函数返回值：enumIrFree：红外空闲

enumIrBusy：红外正忙（正在发送数据包，或正在接收数据包）

1. 红外接收事件enumEventIrRxd：红外Ir上收到一个符合格式的数据包

（红外格式见IrPrint()函数说明）。

补充说明：

(1) 串口1、串口2波特率可独立设置，互不影响.

(2) 串口1、串口2、红外通信可同时工作，互不影响

(3) 串口1、串口2用法基本上完全一致，红外通信用法也基本相同。不同地方是：

a，串口1固定在USB接口上，可用于与计算机通信；而串口2可初始化在EXT扩展、或485接口上（在485接口上时仅单工工作）；

b，红外通信速率固定不可变（大约相当于500～800 bps），通信时没有包头匹配功能。红外模块除通信功能外，还提供用于电器红外遥控的应用函数；

c，红外通信模块仅为单工工作。不发送时自动进入接收状态；有数据发送时自动进入发送状态，但正在接收数据包过程中不会进入发送状态。

编写：徐成（电话18008400450） 2021年8月24日设计，2021年9月1日更新

\*/

#ifndef \_IR\_H\_

#define \_IR\_H\_

extern void IrInit(unsigned char Protocol); //IR模块初始化。参数：红外协议名

enum IrProtocalName {NEC\_R05d=43}; //定义红外IR协议基本周期 = 43\*13（560 (uS)

extern char IrTxdSet(unsigned char \*pt,unsigned char num);

extern char IrPrint(void \*pt, unsigned char num);

extern void SetIrRxd(void \*RxdPt);

extern unsigned char GetIrRxNum(void);

enum IrActName {enumIrTxFree=0 //红外发送空闲

,enumIrTxBusy // 忙

,enumIrTxOK // 发送成功

,enumIrTxFailure}; // 发送失败（正忙）

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*StepMotor 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

StepMotor用于STC-B板控制步进电机。共提供1个驱动函数、3个应用函数：

1. StepMotorInit()：步进电机模块驱动函数

(2) SetStepMotor(char StepMotor,unsigned char speed ,int steps ) 指定步进电机、按指定转动速度、转动指定步

函数参数：StepMotor 指定步进电机，取值（enum StepMotorName中定义）

enumStepMotor1：SM 接口上的步进电机

enumStepMotor2：此时，用L0～L3四个LED模拟一个4相步进电机

enumStepMotor3：此时，用L4～L7四个LED模拟一个4相步进电机

speed 步进电机转动速度（0～255），单位：步/S。 (实际每步时间=int(1000mS/speed) mS），与设置速度可能存在一定误差

steps 步进电机转动步数（-32768～32767），负值表示反转

函数返回：enumSetStepMotorOK：调用成功（enum StepMotorActName中定义）

enumSetStepMotorFail：调用失败（电机名不在指定范围，或speed=0,或调用时正在转动）

（3）EmStop(char StepMotor) 紧急停止指定步进电机转动

函数参数：StepMotor 指定步进电机。函数参数不对将返回0值。

函数返回：剩余未转完的步数

（4）GetStepMotorStatus(char StepMotor) 获取指定步进电机状态

函数参数：StepMotor 指定步进电机

函数返回：enumStepMotorFree:自由（enum StepMotorActName中定义）

enumStepMotorBusy,忙（正在转动）

enumSetStepMotorFail：调用失败（步进电机名不在指定范围）

编写：徐成（电话18008400450） 2021年4月16日设计，2021年4月18日更新

\*/

#ifndef \_StepMotor\_H\_

#define \_StepMotor\_H\_

extern void StepMotorInit(); // 步进电机模块初始化

extern char SetStepMotor(char StepMotor,unsigned char speed ,int steps );

// 指定步进电机、按指定转动速度、转动指定步

// 函数参数：StepMotor 指定步进电机，取值（enum StepMotorName中定义）

// enumStepMotor1： SM 接口上的步进电机

// enumStepMotor2： 此时，用L0～L3四个LED模拟一个4相步进电机

// enumStepMotor3： 此时，用L4～L7四个LED模拟一个4相步进电机

// speed 步进电机转动速度（0～255），单位：步数/秒 // steps 步进电机转动步数（-32768～32767），负值表示反转

// 函数返回： // enumSetStepMotorOK：调用成功

// enumSetStepMotorFail：调用失败（电机名不在指定范围，或speed=0,或调用时正在转动）

extern int EmStop(char StepMotor); // 紧急停止指定步进电机转动

// 函数参数：StepMotor 指定步进电机(函数参数不对将返回0值)

// 函数返回：剩余未转完的步数

extern unsigned char GetStepMotorStatus(char StepMotor); // 获取指定步进电机状态

// 函数参数：StepMotor 指定步进电机

// 函数返回：enumStepMotorFree:自由

// enumStepMotorBusy,忙（正在转动）

// enumSetStepMotorFail：调用失败（步进电机名不在指定范围）

enum StepMotorName {enumStepMotor1=0,

enumStepMotor2,

enumStepMotor3};

enum StepMotorActName {enumStepMotorFree,

enumStepMotorBusy,

enumSetStepMotorOK,

enumSetStepMotorFail};

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DS1302 V1.1 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DS1302模块用于控制“STC-B学习板”上DS1302芯片操作。

DS1302提供RTC（实时时钟）和NVM（非易失存储器）功能（断电后，RTC和NVM是依靠纽扣电池BAT维持工作的）。其中：

RTC提供：年、月、日、星期、时、分、秒功能

NVM提供：31 Bytes非易失存储器功能(地址为：0～30）。地址30单元被DS1302Init函数用于检测DS1302是否掉电，用户不能使用。

DS1302模块共提供1个驱动函数、4个应用函数：

(1) void DS1302Init(struct\_DS1302\_RTC time)：DS1302驱动函数。使用DS1302，需用该函数初始化和驱动一次

函数参数：结构struct\_DS1302\_RTC time

如果DS1302掉电（初始化时检测RTC数据失效），则以参数time定义的时间初始化RTC

函数返回值：无

(2) struct\_DS1302\_RTC RTC\_Read(void)：读取DS1302内部实时时钟RTC内容

函数参数：无

函数返回值：结构struct\_DS1302（见结构struct\_DS1302定义）

(3) void RTC\_Write(struct\_DS1302\_RTC time) ：写DS1302内部实时时钟RTC内容

函数参数：结构struct\_DS1302 time（见结构struct\_DS1302定义）

函数返回值：无

(4) unsigned char NVM\_Read(unsigned char NVM\_addr): 读取NVM一个指定地址内容

函数参数：NVM\_addr：指定非易失存储单元地址，有效值0～30（31个单元）

函数返回值：当函数参数正常时，返回NVM中对应单元的存储数值（1Byte）

当函数参数错误时，返回enumDS1302\_error

(5) unsigned char NVM\_Write(unsigned char NVM\_addr, unsigned char NVM\_data)：向NVM一个指定地址写入新值

函数参数：NVM\_addr：指定非易失存储单元地址，有效值：0～30（31个单元。其中，第30单元被DS1302Init()函数用于检测DS1302是否掉电，用户不能使用)）

NVM\_data：待写入NVM单元的新值（1Byte）

函数返回值：当函数参数正常时，返回enumDS1302\_OK

当函数参数错误时，返回enumDS1302\_error

结构struct\_DS1302\_RTC定义：（参见DS1302Z数据手册）

typedef struct

{ unsigned char second; //秒（BCD码，以下均为BCD码）

unsigned char minute; //分

unsigned char hour; //时

unsigned char day; //日

unsigned char month; //月

unsigned char week; //星期

unsigned char year; //年

} struct\_DS1302\_RTC;

关于DS1302内部非易失性存储器补充说明：

a, DS1302提供的非易失性存储器为低功耗RAM结构，靠纽扣电池保持掉电后其存储内容不变。

b, 与M24C01提供的NVM区别：容量小（仅31字节），但无”写“寿命问题，且写周期很短（可忽略：即两次写操作之间无需等待）；

c, 读、写DS1302内部NVM每一个字节均需要花费一定操作时间（数十uS）；

d, 仅在需要时使用以上读或写函数读写需要的特定字节内容，应避免对其进行无效、大量、重复操作！

编写：徐成（电话18008400450） 2021年8月5日设计，2021年8月15日改进

\*/

#ifndef \_DS1302\_H\_

#define \_DS1302\_H\_

typedef struct

{ unsigned char second; //秒（BCD码，以下均为BCD码）

unsigned char minute; //分

unsigned char hour; //时

unsigned char day; //日

unsigned char month; //月

unsigned char week; //星期

unsigned char year; //年

} struct\_DS1302\_RTC;

extern void DS1302Init(struct\_DS1302\_RTC time); //DS1302初始化

extern struct\_DS1302\_RTC RTC\_Read(void); //读RTC（读RTC时钟内容）

extern void RTC\_Write(struct\_DS1302\_RTC time); //写RTC（校对RTC时钟）

extern unsigned char NVM\_Read(unsigned char NVM\_addr);

//读NVM（读DS1302中的非易失存储单元内容）

extern unsigned char NVM\_Write(unsigned char NVM\_addr, unsigned char NVM\_data);

//写NVM（写DS1302中的非易失存储单元）

enum DS1302name {enumDS1302\_OK,enumDS1302\_error};

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* M24C02 V1.0 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

M24C02模块用于控制“STC-B学习板”上IIC接口非易失存储器M24C02芯片操作。

M24C02提供2K bits（256 Bytes）非易失存储器(NVM)，字节单元地址为：00～0xff。

M24C0402模块共提供2个应用函数(本模块不需要初始化）

(1) unsigned char M24C02\_Read(unsigned char NVM\_addr): 读取指定地址内容

函数参数： NVM\_addr，指定非易失存储单元地址，有效值00～0xff

函数返回值：返回M24C02中对应单元的存储数值（1Byte）

(2) void M24C02\_Write(unsigned char NVM\_addr, unsigned char NVM\_data)：向M24C02一个指定地址写入新值

函数参数： NVM\_addr：指定非易失存储单元地址，有效值：00～0xff

NVM\_data：待写入非易失存储单元的新值（1Byte）

补充说明：

a, M24C02为非易失性存储器，其主要特点是：存储的内容在断电后能继续保存，一般用于保存断电需保留的工作系统参数；

b, 读、写M24C02内部每一个字节均需要花费一定时间（每次读写操作大约数十uS，写周期为5～10mS），且有”写“寿命限制（每一单元大约”写“寿命为10万次量级寿命）；

c, 与DS1302内部NVM主要区别：容量大（M24C02提供256字节，M24CXX系列最大可提供64K字节），但有“写”寿命限制（一般为数十万次“写”寿命“，且写周期长（5～10mS）

d，因此要求：

两次写操作之间需间隔5～10mS以上；

仅在需要时使用以上读或写函数读写需要的特定字节内容，应避免对其进行无效、大量、重复操作！

编写：徐成（电话18008400450） 2021年8月8日设计

#ifndef \_M24C02\_H\_

#define \_M24C02\_H\_

extern unsigned char M24C02\_Read(unsigned char NVM\_addr);

//读NVM（读M24C02中的非易失存储单元内容）

extern void M24C02\_Write(unsigned char NVM\_addr, unsigned char NVM\_data);

//写NVM（写M24C02中的非易失存储单元）

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FMRadio V1.1 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FMRadio模块用于控制“STC-B学习板”上FM收音机操作。

FMRadio模块模块共提供1个初始化函数、2个应用函数：

(1) void FMRadioInit(struct\_FMRadio FMRadio); //收音机模块初始化函数。

函数参数：FMRadio (见结构struct\_FMRadio定义)

函数返回值：无

(2) void SetFMRadio(struct\_FMRadio FMRadio); //设置FM收音机控制参数。

函数参数：FMRadio (见结构struct\_FMRadio定义)

函数返回值：无

（3） struct\_FMRadio GetFMRadio(void); //获取当前FM收音机参数。

函数参数：无

函数返回值：返回FM控制模型数据(见结构struct\_FMRadio定义)

结构struct\_FMRadio定义：

typedef struct //FM收音机控制模型

{ unsigned int frequency; // FM收音频率，范围：887～1080（单位：0.1MHz）

unsigned char volume; // FM音量，范围：0～15。0为最小音量。

unsigned char GP1; // FM指示灯1。=0 输出低，GP1亮；!=0 输出高，GP1灭

unsigned char GP2; // FM指示灯2。=0 输出低，GP1亮；!=0 输出高，GP1灭 unsigned char GP3; // FM指示灯3。=0 输出低，GP1亮；!=0 输出高，GP1灭

} struct\_FMRadio;

编程注意事项：

1，本版本暂未输出调谐、自动搜索、电台信号等控制和状态信息，因此，暂不能完成自动搜索电台等收音机功能。

编写：徐成（电话18008400450） 2021年8月10日设计，2021年8月16日改进

\*/

#ifndef \_FM\_Radio\_H\_

#define \_FM\_Radio\_H\_

typedef struct //FM收音机控制模型

{ unsigned int frequency; // FM收音机收音频率 (887～1080。单位：0.1MHz）

unsigned char volume; // FM收音机音量 (0～15。0为最小音量）

unsigned char GP1; // FM指示灯1。 =0 输出低，GP1亮； !=0 输出高，GP1灭

unsigned char GP2; // FM指示灯2。 =0 输出低，GP1亮； !=0 输出高，GP1灭

unsigned char GP3; // FM指示灯3。 =0 输出低，GP1亮； !=0 输出高，GP1灭

} struct\_FMRadio;

extern void FMRadioInit(struct\_FMRadio FMRadio);

//收音机模块初始化函数。输入FM控制模型数据，无返回值

extern void SetFMRadio(struct\_FMRadio FMRadio); //设置FM收音机控制参数

extern struct\_FMRadio GetFMRadio(void); //获取当前FM收音机参数

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EXT V1.0 说明 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EXT模块用于控制“STC-B学习板”上扩展接口EXT上相关操作。

EXT模块根据应用需要，在外接相应模块或部件后，可实现多种相应功能。这里提供部分（称重、超声波测距、旋转编码器、PWM）应用驱动和API函数。

EXT模块这里提供1个驱动函数和若干个应用层API函数。EXT模块的API函数不是同时有效的，而是根据初始化函数参数不同而分别有效。

(1) void EXTInit(char EXTfunction)：EXT初始化函数。

函数参数：EXTfunction。定义EXT接口功能

取值：enumEXTWeight （秤重：由HX710、HX711组成的电子秤）

enumEXTPWM （脉宽调制：可用于控制直流电机正反转、速度等）

enumEXTDecode （增量式计数）

enumEXTUltraSonic （超声波测距）

函数返回值：无

(2)API函数

(a) 电子秤。当EXTInit(char EXTfunction)使用enumEXTWeight参数时，GetWeight函数有效。

int GetWeight(void) //获取电子秤ADC秤重数据

// 16bit，带符号整数。未清零、未标定。

// 参见HX710、HX711数据手册（仅返回高16bit）

(b) PWM脉宽调制输出。当EXTInit(char EXTfunction)使用enumEXTPWM参数时，SetPWM函数有效。

void SetPWM(unsigned char PWM1, unsigned char freq1, unsigned char PWM2, unsigned char freq2);

//设置EXT上输出PWM

//参数PWMx为占空比（输出高电平时间的比例）：0～100,单位%。x=1或2

// freqx为频率：1～255Hz）

//实际频率 = 1000/int(1000/freqx)。

//即：1000/i=4，5，6...1000，或250，200，167，143，125，……，1

//可用于控制直流电机正反转、转速（配合H型桥式电路），灯亮度，等

(c) 旋转编码器、或增量式编码器。当EXTInit(char EXTfunction)使用enumEXTDecode参数时，GetDecode函数有效。

int GetDecode(void); //获取增量编码器增量值（相对上次读取后的新增量）

(d) 超声波测距。当EXTInit(char EXTfunction)使用enumEXTUltraSonic参数时，GetUltraSonic函数有效。

int GetUltraSonic(void); //获取超声波测距值（每秒5次测量，返回值单位：cm）

(e) RFID读卡（暂缓）

(g) 串口、外接蓝牙模块，见串口2

(h) 气敏、数据采集、电子尺等，见ADC

编写：徐成（电话18008400450） 2021年8月24日设计

#ifndef \_EXT\_H\_

#define \_EXT\_H\_

extern void EXTInit(char EXTfunction); //扩展接口初始化

enum EXTname {enumEXTWeight //电子秤

,enumEXTPWM, //PWM，控制直流电机转动方向、快慢

,enumEXTDecode //增量式计数（旋转编码器）

,enumEXTUltraSonic //超声波测距

//串口2，蓝牙：见uart2模块

//气敏、数据采集、电子尺、额温枪等：见ADC

extern int GetWeight(void); //获取电子秤ADC秤重数据

//HX711输出高16bit，带符号数整数。未清零、未标定

extern int GetDecode(void); //获取增量编码器增量值（相对上次读取后的新增量）

extern int GetUltraSonic(void); //获取超声波测距值（每秒5次测量，返回值单位：cm）

extern void SetPWM(unsigned char PWM1, unsigned char freq1, unsigned char PWM2, unsigned char freq2);

//设置EXT上输出PWM

//参数PWMx为占空比（输出高电平时间的比例）：0～100,单位%）。x=1或2

//freqx为频率：1～255Hz

//实际频率 = 1000/int(1000/freqx)。即：1000/i=4，5，6...1000

//或250，200，167，143，125，111，100，91，83...1

//可用于控制直流电机正反转、转速（配合H型桥式电路），灯亮度，等

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* STCBSP测试Demo：STC\_DemoV3 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

STCBSP提供的底层支持和应用层函数支持以下"STC-B学习板"功能同时工作：

1，数码管显示

2，LED指示灯显示

3，实时时钟

4，温度光照测量

5，音乐播放

6，FM收音机

7，EXT扩展接口（电子秤、超声波测距、旋转编码器、PWM控制，4选1工作）

8，振动传感器

9，霍尔传感器

10，步进电机控制

11，串口1通信

12，串口2通信（485、EXT扩展接口，2选1）

13，红外遥控

14，红外收发通信

15，非易失性NVM存储（DS1302提供31字节，M24C02提供256字节）

具体使用方法请参见各模块头文件。

STC\_DemoV3示例程序（源程序由main.c、main.h、function.c三部分组成）功能描述如下：

1，按键Key2切换"显示"和"按键操作"模式。模式值在LED上显示（二进制数）。模式有：

模式1：实时时钟（年月日）

模式2：实时时钟（时分秒）

模式3：温度、光照测量

模式4：音乐播放

模式5：FM收音机

模式6：超声波测距（需要在EXT上接超声波测距模块）

模式7：电子秤 （需要在EXT上接电子秤模块）

2，数码管显示、导航按键功能：

在模式1下： 数码管显示实时时钟RTC "年年-月月-日日"，

导航按键 "中"键 进入或退出"设置年月日"；"上、下、左、右"设置

在模式2下： 数码管显示实时时钟RTC "时时-分分-秒秒"，

导航按键 "中"键 进入或退出"设置时分秒"；"上、下、左、右"设置

在模式3下： 数码管显示温度、光照AD值"温温温 光光光",

导航按键无操作

在模式4下： 数码管显示 "节拍率 音调音高 音符"，

导航按键 "中"调暂停/播放，"上、下"调整音调，"左、右"调整节拍率

在模式5下： 数码管显示 "音量 收音频率（MHz）"，

导航按键 "中"保存收音参数到NVM（下次上电时用）

"上、下"调整音亮，"左、右"调整收音频率

在模式6下： 数码管显示 " 距离值（cm）" ，

导航按键无操作

在模式7下： 数码管显示 "称重ADC值（符号数）" ，

导航按键无操作

3，按键1（按下时）：红外向外发送 "大家好！"。

其它"STC-B学习板"（如用本程序）用串口助手（设置波特率1200bps，文本接收方式）可看到接收到的"大家好！"文本。

按键3（抬起时）：串口1向计算机发送程序运行性能参数。

性能参数描述：程序主循环每秒次数、每秒用户程序调度丢失数

4，振动传感器：（如果有音乐播放）控制音乐播放"暂停/继续"

5，霍尔传感器：有磁场接近时，（如果蜂鸣器空闲）发声报警（1350Hz，发1秒时间）

6，串口1：与计算机双向通信（波特率1200bps）

如果收到一个"AA 55"开头、大小8字节数据包，则将第7字节+1，再以波特率2400bps向串口2（485、EXT可选）转发此数据包

如果红外接口收到数据包（引导码+数据+结束码），则向串口1转发

7，串口2：可选择485（双向单工）、EXT（TTL双向双工）两个位置，工作波特率2400bps

如果收到一个"AA 55"开头、大小8字节数据包，则将第7字节+2，再向红外Ir口转发此数据包（NEC\_R05d编码协议）

8，红外收发接口：红外NEC\_R05d编码协议解码

1. 如果红外接口收到数据包（引导码+数据+结束码），则蜂鸣器发声（1000Hz，300mS），并向串口1转发；

b，如果数据包包头为"AA 55"，并第3字节为：

F1：调整收音频率（第4、5字节，BCD码收音频率值，0.1MHz，887～1080），

音量（第6字节数据）（数据需有效：0～15），

F2：调整RTC时分秒（第4、5、6字节BCD码）（数据需有效），

F3：调整RTC年月日（第4、5、6字节BCD码）（数据需有效）.

第7字节+4,

调整值存入NVM，

蜂鸣器发声（1000Hz，600mS）

并向串口1转发；

设计：徐成（电话：18008400450） 2021年9月5日

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Main.h文件：

#include "STC15F2K60S2.H" //必须。

#include "sys.H" //必须。

#include "displayer.H"

#include "key.h"

#include "hall.h"

#include "Vib.h"

#include "beep.h"

#include "music.h"

#include "adc.h"

#include "uart1.h"

#include "uart2.h"

#include "stepmotor.h"

#include "DS1302.h"

#include "M24C02.h"

#include "FM\_Radio.h"

#include "EXT.h"

#include "IR.h"

code unsigned long SysClock=11059200; //必须。定义系统工作时钟频率(Hz)，

用户必须修改成与实际工作频率（下载时选择的）一致

#ifdef \_displayer\_H\_ //显示模块选用时必须。（数码管显示译码表，用戶可修改、增加等）

code char decode\_table[]=

{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x00,0x08,0x40,0x01, 0x41, 0x48,

/\* 序号: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 \*/

/\* 显示: 0 1 2 3 4 5 6 7 8 9 (无) 下- 中- 上- 上中- 中下- \*/

0x3f|0x80,0x06|0x80,0x5b|0x80,0x4f|0x80,0x66|0x80,0x6d|0x80,0x7d|0x80,0x07|0x80,0x7f|0x80,0x6f|0x80 };

/\* 带小数点 0 1 2 3 4 5 6 7 8 9 \*/

#endif

function.c文件：

struct\_FMRadio FM\_NVMread(void) //举例，取NVM中读出收音频率、音量

{ struct\_FMRadio a={955,5,0xff,0,0xff};

unsigned int tempuint;

unsigned char tempuchar;

tempuint = ((NVM\_Read(0x01)<<8)) + NVM\_Read(0x02);

if ((tempuint >= 870) & (tempuint <= 1080))

a.frequency=tempuint;

tempuchar = NVM\_Read(0x03);

if ( tempuchar < 15 ) a.volume = tempuchar;

return(a);

}

void FM\_NVMwrite(struct\_FMRadio FM) //NVM中保存收音频率、音量

{ NVM\_Write(0x01,FM.frequency>>8);

NVM\_Write(0x02,FM.frequency);

NVM\_Write(0x03,FM.volume);}

void dealwithIrRxd()

{ unsigned char a1,a2;

if(GetIrRxNum()!=0) //如果红外接口收到数据包

{ if((\*(rxd+0)==0xAA) & (\*(rxd+1)==0x55)) //如果包头 AA 55

{ switch(\*(rxd+2))

{ case 0xF1: a1=\*(rxd+3); //设置FM收音机,

a2=\*(rxd+4);

FM.frequency=((a1/16\*10+a1%16)\*100)+(a2/16\*10+a2%16);

//将第4、5字节调节FM收音频率,BCD码换成二进制

FM.volume=\*(rxd+5); //将 第6字节 调节FM收音音量

SetFMRadio(FM); //设置收音机新收音参数

FM\_NVMwrite(FM); //保存收音参数到NVM中

break;

case 0xF2: t.hour=(\*(rxd+3)); //设置RTC //将第4字节设成RTC时，BCD码

t.minute=(\*(rxd+4)); //将第5字节设成RTC分

t.second=(\*(rxd+5)); //将第6字节设成RTC秒

RTC\_Write(t); //设置RTC

break;

case 0xF3: t.year=(\*(rxd+3)); //设置RTC //将第4字节设成RTC年，BCD码

t.month=(\*(rxd+4)); //将第5字节设成RTC月

t.day=(\*(rxd+5)); //将第6字节设成RTC日

RTC\_Write(t); //设置RTC

break;

default: break;

}

(\*(rxd+6)) += 4; //第7字节加4

SetBeep(1000,60);

}

else SetBeep(1000,30);

Uart1Print(&rxd, sizeof(rxd)); //从串口1发送出去

}

}

void dealwithDisp()

{ unsigned char d0,d1,d2,d3,d4,d5,d6,d7;

int a;

static unsigned char ct100mS=10;

if(funcmode == Music) return; //更新显示，（Music函数自带显示更新，不需更新）

if (--ct100mS ==0 ) ct100mS=10;

switch(funcmode)

{ case RTC\_YMD: if(tempadj ==0) t=RTC\_Read(); //非调时钟时， 读RTC

d0=t.year>>4;

d1=t.year&0x0f;

d2=12;

d3=t.month>>4;

d4=t.month&0x0f;

d5=12;

d6=t.day>>4;

d7=t.day&0x0f;

break;

case RTC\_HMS: if(tempadj ==0) t=RTC\_Read();

d0=t.hour>>4;

d1=t.hour&0x0f;

d2=12;

d3=t.minute>>4;

d4=t.minute&0x0f;

d5=12;

d6=t.second>>4;

d7=t.second&0x0f;

break;

case Rt\_Rop : ADCresult=GetADC();

d0=ADCresult.Rt/100;

d1=ADCresult.Rt/10%10;

d2=ADCresult.Rt%10;

d3=10;

d4=10;

d5=ADCresult.Rop/100;

d6=ADCresult.Rop/10%10;

d7=ADCresult.Rop%10;

break;

case FMradio: d0=FM.volume/10; if(d0==0) d0=10;

d1=FM.volume%10;

d2=10;

d3=10;

d4=FM.frequency/1000; if(d4==0) d4=10;

d5=FM.frequency/100%10;

d6=FM.frequency/10%10+16; //加小数点

d7=FM.frequency%10;

break;

case UltroSonic: a=GetUltraSonic(); //a为测得距离，正数，单位cm

d0=10;

d1=10;

d2=10;

d3=10;

d4=a/1000; if(d4 == 0 ) d4=10; //不显示最高位0

a=a%1000;

d5=a/100+16; //显示小数点

a=a%100;

d6=a/10;

d7=a%10;

break;

case Weight: a=GetWeight(); //a为称重AD数据，带符号整数（16bit补码），未标定

d0=10;

d1=10;

d2=10; if (a < 0 ) { a = -a; d2=12; } //a变正数，d2符号

d3=a/10000; a=a%10000;

d4=a/1000; a=a%1000;

d5=a/100; a=a%100;

d6=a/10;

d7=a%10;

break;

default: break;

}

if(ct100mS >= 8)

switch (tempadj)

{ case 1: d0=10; d1=10; break;

case 2: d3=10; d4=10; break;

case 3: d6=10; d7=10; break;

default: break;

}

Seg7Print(d0,d1,d2,d3,d4,d5,d6,d7);

}

void dealwithmyKN()

{ unsigned char a;

switch(funcmode)

{case RTC\_HMS: if(GetAdcNavAct(enumAdcNavKeyRight) == enumKeyPress)

{ if(tempadj) if(tempadj<3) {tempadj++;SetBeep(1000,20);}}

if(GetAdcNavAct(enumAdcNavKeyLeft) == enumKeyPress)

{ if(tempadj) if(tempadj>1) {tempadj--;;SetBeep(1000,20);}}

if(GetAdcNavAct(enumAdcNavKeyUp)== enumKeyPress)

{ if(tempadj == 1) { a=t.hour/16\*10+t.hour%16;

if((++a) >=24) a =0;

t.hour=a/10\*16+a%10;

SetBeep(1000,20);

}

if(tempadj == 2) { a=t.minute/16\*10+t.minute%16;

if((++a) >=60) a =0;

t.minute=a/10\*16+a%10;

SetBeep(1000,20);

}

if(tempadj == 3) { a=t.second/16\*10+t.second%16;

if((++a) >=60) a =0;

t.second=a/10\*16+a%10;

SetBeep(1000,20);

}

}

if(GetAdcNavAct(enumAdcNavKeyDown)== enumKeyPress)

{ if(tempadj == 1) { a=t.hour/16\*10+t.hour%16;

if((a--) == 0) a =23;

t.hour=a/10\*16+a%10;

SetBeep(1000,20);

}

if(tempadj == 2) { a=t.minute/16\*10+t.minute%16;

if((a--) == 0) a =59;

t.minute=a/10\*16+a%10;

SetBeep(1000,20);

}

if(tempadj == 3) { a=t.second/16\*10+t.second%16;

if((a--) == 0) a =59;

t.second=a/10\*16+a%10;

SetBeep(1000,20);

}

}

if(GetAdcNavAct(enumAdcNavKeyCenter)== enumKeyPress)

{ SetBeep(1000,20);

if(tempadj == 0 ) tempadj=3;

else { tempadj=0;

RTC\_Write(t);

}

}

break;

case RTC\_YMD: if(GetAdcNavAct(enumAdcNavKeyRight) == enumKeyPress)

{ if(tempadj) if(tempadj<3) { tempadj++;SetBeep(1000,20);}}

if(GetAdcNavAct(enumAdcNavKeyLeft) == enumKeyPress)

{ if(tempadj) if(tempadj>1) { tempadj--;SetBeep(1000,20);}}

if(GetAdcNavAct(enumAdcNavKeyUp)== enumKeyPress)

{ if(tempadj == 1) { a=t.year/16\*10+t.year%16;

if((++a) >=99) a =0;

t.year=a/10\*16+a%10;

SetBeep(1000,20);

}

if(tempadj == 2) { a=t.month/16\*10+t.month%16;

if((++a) >=12) a =1;

t.month=a/10\*16+a%10;

SetBeep(1000,20);

}

if(tempadj == 3) { a=t.day/16\*10+t.day%16;

if((++a) >=31) a =1;

t.day=a/10\*16+a%10;

SetBeep(1000,20);

}

}

if(GetAdcNavAct(enumAdcNavKeyDown)== enumKeyPress)

{ if(tempadj == 1) { a=t.year/16\*10+t.year%16;

if((a--) == 0) a =99;

t.year=a/10\*16+a%10;

SetBeep(1000,20);

}

if(tempadj == 2) { a=t.month/16\*10+t.month%16;

if((a--) == 1) a =12;

t.month=a/10\*16+a%10;

SetBeep(1000,20);

}

if(tempadj == 3) { a=t.day/16\*10+t.day%16;

if((a--) == 1) a =31;

t.day=a/10\*16+a%10;

SetBeep(1000,20);

}

}

if(GetAdcNavAct(enumAdcNavKeyCenter)== enumKeyPress)

{ SetBeep(1000,20);

if(tempadj == 0 ) tempadj=3;

else { tempadj=0;

RTC\_Write(t);

}

}

break;

case Music: if(GetAdcNavAct(enumAdcNavKeyRight) == enumKeyPress)

{ if(Music\_PM <= 210) Music\_PM +=10;} //节凑最高限每分钟220拍

if(GetAdcNavAct(enumAdcNavKeyLeft) == enumKeyPress)

{ if(Music\_PM >= 60) Music\_PM -=10;} // 最低限 50

if(GetAdcNavAct(enumAdcNavKeyUp)== enumKeyPress)

{ if(Music\_tone < 0xFF) Music\_tone++;} //音调最高F调（显示：6）

if(GetAdcNavAct(enumAdcNavKeyDown)== enumKeyPress)

{ if(Music\_tone > 0xF9) Music\_tone--;} // 最低 A 显示：0

SetMusic(Music\_PM, Music\_tone, enumModeInvalid,

enumModeInvalid, enumModeInvalid); //调整播放节凑和音调

if(GetAdcNavAct(enumAdcNavKeyCenter)==enumKeyPress) //播放、暂停

if (GetPlayerMode() == enumModePlay)

SetPlayerMode(enumModePause);

else

{ if (GetPlayerMode() == enumModeStop)

SetMusic(Music\_PM, Music\_tone, &song, sizeof(song), enumMscDrvSeg7);

SetPlayerMode(enumModePlay);

}

break;

case FMradio: if(GetAdcNavAct(enumAdcNavKeyRight) == enumKeyPress)

{if(FM.frequency<1080) FM.frequency++;} //频率+

if(GetAdcNavAct(enumAdcNavKeyLeft) == enumKeyPress)

{if(FM.frequency>887) FM.frequency--;} //频率1

if(GetAdcNavAct(enumAdcNavKeyUp)== enumKeyPress)

{if(FM.volume<15) FM.volume++;} //音量+

if(GetAdcNavAct(enumAdcNavKeyDown)== enumKeyPress)

{if(FM.volume>0) FM.volume--;} //音量-

SetFMRadio(FM); //设置收音机新收音参数

if(GetAdcNavAct(enumAdcNavKeyCenter)== enumKeyPress)

{FM\_NVMwrite(FM); } //保存参数到NVM

break;

default: break;

}

if(GetAdcNavAct(enumAdcNavKey3) == enumKeyRelease)

//示例。按键3:(抬起时）向串口1发送二进制数据包：报告系统性能

{ SysPer=GetSysPerformance();

if((GetUart1TxStatus()==enumUart1TxFree)) Uart1Print(&SysPer, sizeof(SysPer));

SetBeep(1000,20);

}

}

void dealwithmykey()

{ if (GetKeyAct(enumKey2)== enumKeyPress)

//按键2:（按下时）切换显示和按键功能模式

{ SetBeep(1000,20);

if(++funcmode > Weight) funcmode= RTC\_YMD;

LedPrint(funcmode);

M24C02\_Write(0x00,funcmode);

if (funcmode == Music) Seg7Print(10,10,12,12,12,12,10,10);

if (funcmode == Music+1)

SetMusic(enumModeInvalid, enumModeInvalid, enumModeInvalid,

enumModeInvalid, enumMscNull); //离开音乐播放模式，关音乐显示

if (funcmode == Weight) EXTInit(enumEXTWeight);

if (funcmode == UltroSonic) EXTInit(enumEXTUltraSonic);

}

if (GetKeyAct(enumKey1)== enumKeyPress) //按键1:（按下时）向红外发送文本

{ IrPrint("大家好！\r\n", sizeof("大家好！\r\n"));

SetBeep(1000,100);

}

}

main.c文件：

//\*\*\*\*\*\*\* 用户程序段1：用户程序包含文件 \*\*\*\*\*\*\*\*\*\*\*\*\*//

#include "main.H" //必须。编写应用程序时，仅需改写 main.h 和 main.c文件

#include "song.c" //举例。song.c中编写了音乐（同一首歌）编码

//\*\*\*\*\*\*\* 用户程序段2：用户自定义函数声明 \*\*\*\*\*\*\*\*\*\*\*\*\*//

//\*\*\*\*\*\*\* 用户程序段3：用户程序全局变量定义 \*\*\*\*\*\*\*\*\*\*\*\*\*//

struct\_DS1302\_RTC t={0x30,0,9,0x06,9,1,0x21};

//举例。 实时时钟数据结构：秒、分、时、日、月、周、年。

//初值：2021年9月6日，周一，9：00：30

struct\_FMRadio FM; //举例。 FM收音机数据结构：收音频率、音量、GP1、GP2、GP3。

//初值：95.5MHz，音量5，灭、灭、亮

struct\_SysPerF SysPer; //举例。 系统性能数据结构：

//每秒主循环次数4字节、每秒轮询丢失次数1字节

struct\_ADC ADCresult; //举例。 热敏、光敏测量AD值

unsigned char Music\_tone,Music\_PM; //举例。 音乐播放音调、节凑（每分钟节拍数）

unsigned char rxd[8]; //举例。通信（串口1、串口2、红外共用）缓冲区8字节

unsigned char rxdhead[2]={0xaa,0x55}; //举例。通信（串口1、串口2）接收包头

//匹配字符2字节：（0xAA，0x55）

unsigned char funcmode; //举例。定义显示、按键功能模式

enum funcname {RTC\_YMD=1, //举例。功能模式命名。实时时钟：年月日

RTC\_HMS, // 实时时钟：时分秒

Rt\_Rop, // 热敏光敏测量

Music, // 音乐播放

FMradio, // FM收音机

UltroSonic, // 超声波测距

Weight}; // 电子秤

unsigned char tempadj; //举例。程序变量。

//调整时钟时用：=1 调年或时；=2 调月或分 =3 调日 或秒

//\*\*\*\*\*\*\* 用户程序段4：用户自定义函数原型 \*\*\*\*\*\*\*\*\*\*\*\*\*//

#include "function.c"

void myUart1Rxd\_callback() //举例。串口1收到合法数据包回调函数。

{ if ( GetUart2TxStatus() == enumUart2TxFree )

{ (\*(rxd+6)) += 1; //第7字节加1

Uart2Print(&rxd, sizeof(rxd)); //将数据包从串口2(485、或EXT扩展接口上)发送出去

}

}

void myUart2Rxd\_callback() //举例。串口2收到合法数据包回调函数。

{ if ( GetIrStatus() == enumIrFree )

{ (\*(rxd+6)) += 2; //第7字节加2

IrPrint(&rxd, sizeof(rxd)); //将数据包从红外发送出去

}

}

void myIrRxd\_callback() //举例。红外收到数据包回调函数。（NEC\_R05d编码）

{ dealwithIrRxd();

}

void my1mS\_callback() //举例。1mS事件回调函数

{

}

void my10mS\_callback() //举例。10mS事件回调函数

{

}

void my100mS\_callback() //举例。100mS事件回调函数

{ dealwithDisp();

}

void my1S\_callback() //举例。1S事件回调函数

{

}

void myADC\_callback() //举例。ADC事件回调函数

{

}

void myKN\_callback() //举例。导航按键事件回调函数

{ dealwithmyKN();

}

void mykey\_callback() // 按键（Key1、Key2）事件回调函数

{ dealwithmykey();

}

void myhall\_callback() //示例。有hall事件回调函数：发声报警

{ if(GetHallAct() == enumHallGetClose) SetBeep(1350,100);

}

void mySV\_callback() //示例: 振动事件回调函数：控制音乐播放/暂停

{ if(GetVibAct())

if (GetPlayerMode() == enumModePause) SetPlayerMode(enumModePlay);

else SetPlayerMode(enumModePause);

}

//\*\*\*\*\*\*\* main()函数 \*\*\*\*\*\*\*\*\*\*\*\*\*//

void main() { //主函数 main() 开始 //此行必须！！！

//\*\*\*\*\*\*\* 用户程序段5：用户main()函数内部局部变量定义 \*\*\*\*\*\*\*\*\*\*\*\*\*//

//\*\*\*\*\*\*\* 用户程序段6：用户main()函数（初始化类程序） \*\*\*\*\*\*\*\*\*\*\*\*\*//

//1,加载需要用的模块(由各模块提供加载函数)

Key\_Init(); //举例，需要用到的模块及其函数、方法，必须对其初始化

HallInit(); //举例

VibInit(); //举例

DisplayerInit(); //举例

BeepInit(); //举例

MusicPlayerInit(); //举例

AdcInit(ADCexpEXT); //举例，ADC模块初始化，有参数

StepMotorInit(); //举例，步进电机初始化

DS1302Init(t); //举例，DS1302初始化

IrInit(NEC\_R05d); //举例，红外接口设置

/\*\*\*\*\*\*\* 以下可4选1：加载EXT接口 \*\*\*\*\*\*\*\*/

// EXTInit(enumEXTWeight); //举例，EXT初始化成电子秤功能

// EXTInit(enumEXTPWM); //举例，EXT初始化成两路PWM功能

// EXTInit(enumEXTDecode); //举例，EXT初始化成增量式编码器解码功能

// EXTInit(enumEXTUltraSonic); //举例，EXT初始化成超声波测距功能

Uart1Init(1200); //举例，串口1初始化，有参数

/\*\*\*\*\*\*\* 以下可2选1：加载串口2 \*\*\*\*\*\*\*\*/

Uart2Init(2400,Uart2Usedfor485); //举例，串口2初始化到485接口（半双工）

// Uart2Init(2400,Uart2UsedforEXT); //举例，串口2初始化到EXT接口（TTL全双工）

//2,设置事件回调函数(由sys提供设置函数SetEventCallBack())

SetEventCallBack(enumEventKey, mykey\_callback); //举例

SetEventCallBack(enumEventSys1mS, my1mS\_callback); //举例

SetEventCallBack(enumEventSys10mS, my10mS\_callback); //举例

SetEventCallBack(enumEventSys100mS, my100mS\_callback); //举例

SetEventCallBack(enumEventSys1S, my1S\_callback); //举例

SetEventCallBack(enumEventHall, myhall\_callback); //举例

SetEventCallBack(enumEventVib, mySV\_callback); //举例

SetEventCallBack(enumEventNav, myKN\_callback); //举例，设置导航按键回调函数

SetEventCallBack(enumEventUart1Rxd, myUart1Rxd\_callback);

//举例，设置串口1接收回调函数

SetEventCallBack(enumEventUart2Rxd, myUart2Rxd\_callback);

//举例，设置串口2接收回调函数

SetEventCallBack(enumEventXADC,myADC\_callback); //扩展接口上新的AD值事件

SetEventCallBack(enumEventIrRxd,myIrRxd\_callback); //红外Ir上收到一个数据包

//3,用户程序状态初始化

SetDisplayerArea(0,7); //举例

SetUart1Rxd(&rxd, sizeof(rxd), rxdhead, sizeof(rxdhead));

//设置串口接收方式：数据包条件：

//接收数据包放置在rxd中，数据包大小rxd大小，

//数据包头需要与rxdhead匹配，匹配数量rxdhead大小

SetUart2Rxd(&rxd, sizeof(rxd), rxdhead, sizeof(rxdhead)); //举例

SetIrRxd(&rxd); //举例

//4,用户程序变量初始化

FM=FM\_NVMread(); //举例，FMRadio初始化。

FMRadioInit(FM);

Music\_PM=90;

Music\_tone=0xFC;

funcmode = M24C02\_Read(0x00);

LedPrint(funcmode);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MySTC\_OS 初始化与加载开始 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

MySTC\_Init(); // MySTC\_OS 初始化 //此行必须！！！

while(1) // 系统主循环 //此行必须！！！

{ MySTC\_OS(); // MySTC\_OS 加载 //此行必须！！！

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MySTC\_OS 初始化与加载结束 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*\*\*\*\*\*\* 用户程序段7：用户main()函数（主循环程序） \*\*\*\*\*\*\*\*\*\*\*\*\*//

} //主循环while(1)结束 //此行必须！！！

} //主函数 main() 结束 //此行必须！！！