MySTC_B BSP_Ver3.6

“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

2022年6月30日更新

更新说明

■ 2022 年 6 月 30 日版本更新
修正 ADC 初始化函数原型的说明

■ 2022 年 5 月 2 日版本更新（相对于 2021 年 11 月 8 日版本）

1. 串口 2 在 RS485 端口时，增加“ModBus 协议”支持（更新“异步通信模块函数库 Uart2.lib”）。使用方法见相应章节说明或“uart2.h”
2. 优化串口 1、串口 2 接收，在数据包超出定义的最大接收字节数时，后续接收字节数丢弃（与 IR 接收一致）

■ 2021 年 11 月 8 日版本更新（相对于 2021 年 9 月 4 日版本）

1，红外模块。改进了函数：

```
void SetIrRxd(void *RxdPt,unsigned char RxdNmax)
```

该函数较以前增加了一个参数 RxdNmax（定义接收数据包最大字节数），并修改了其中的解码程序。改进后，可避免接收数据过多而越界、无效的 0 字节数据包两处不当之处。

更新 STCBSP.lib 后，原应用程序中的 SetIrRxd 函数使用应作相应变动，详见 Ir.h。

2，修改 ADC 模块存在的一处 BUG，避免以前版本可能出现的 ADC 模块失灵现象。
本修改不影响原应用程序使用方法。

3，更正了 Uart1、Uart2 设置 band=115200bps 时的 BUG。
本修改不影响原应用程序使用方法。

目录

/****************************** STCBSP_V3.6 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(char);          //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）的数据包或一个 ModBus 数据包 （2022 年 5 月 2 日新增）

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数据包事件
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数据包事件
或一个 ModBus 数据包（2022 年 5 月 2 日新增）
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，非阻塞型；

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: ADCincEXT: 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)
ADCexpEXT: 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 V2.0 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 long band)：Uart1 模块初始化函数。
函数参数：unsigned long 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

(2) 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
当收到的数据大于 Nmax 后，将被丢弃
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，用户回调函数返回才接收下一个数据包
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：串口（1 和 2）上收到的两个数据包之时间间隔要求不小于 1mS（原因：系统内部调度方法限制）

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 11 月 8 日更新

*/

```
#ifndef _uart1_H_
#define _uart1_H_

extern void Uart1Init(unsigned long 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 V2.0 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 long band,unsigned char Uart2mode)：Uart2 初始化函数
函数参数：unsigned long 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）
Uart2Usedfor485ModBus —— 串口 2 用于 485 ModBus 协议收发
函数返回值：无

(2) 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
当收到的数据大于 Nmax 后，将被丢弃
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 事件后，用户回调函数返回才接收下一个数据包
函数返回值：无

(3) 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）

(4) 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。

当串口 2 工作于： Uart2UsedforEXT 或 Uart2Usedfor485 方式时，

Uart2 接收事件为：同串口 1 性质一致，

即：数据包头匹配、数据包大小一致

当串口 2 工作于： Uart2Usedfor485ModBus 方式时，

Uart2 接收事件为：收到一个 ModBus 数据帧

（数据包内字节间间隔<4 字节收发时间），

且帧内包头内容与设定内容匹配

但：1，数据帧内未进行 CRC 校验。

建议用户在回调函数中，

为了验证数据包正确性，

应对数据包进行 CRC 校验

2，未返回 ModBus 数据帧有效字节数。

用户可从收到的指令类型（第 2 字节）

及对应帧内数据（一般第 5、6 字节）

判断帧字节数

补充说明：串口（1 和 2）上收到的两个数据包之时间间隔要求不小于 1mS（原因：系统内部调度方法限制）

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 11 月 8 日更新

*/

```
#ifndef _uart2_H_
#define _uart2_H_


extern void Uart2Init(unsigned long band,unsigned char Uart2mode);     //串口 2 初始化
enum Uart2PortName    {Uart2UsedforEXT,       //串口 2 在 EXT 扩展插座上(TTL 标准串口)
                       Uart2Usedfor485,     //串口 2 用于 485 通信（
                       Uart2Usedfor485ModBus};   //  串口 2 用于 485 上 ModBus 协议
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

(2) 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。

(3) 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）

(4) void SetIrRxd(void *RxdPt,unsigned char RxdNmax)：

设置红外接收数据包存放位置、每个数据包最大字节数。

收到一个数据包（至少 1 字节数据）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）

unsigned char RxdNmax：指定每个数据包接收最大字节数。接收数据如果字节数超出，超出部分将被忽略。该值定义接收数据包最大值，以免超出而影响程序其它数据

函数返回值：无

(5) 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）

(7) 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 11 月 5 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,unsigned char RxdNmax);
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

a，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|0x8
0,0x6f|0x80 };
/* 带小数点    0       1       2       3       4       5       6       7       8       9       */

#endif
```

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!!!
```