Nerve RTOS 开发指南

编辑：阿顿

扫码，获开源代码

NERVEOS 系统的历史版本

1. NerveOS1.0.0
2. NerveOS1.1.0
3. NerveOS1.5.1
4. NerveRTOS1.5.2
5. NerveRTOS1.5.2.5
6. NerveRTOS1.5.3
7. NerveRTOS1.5.4
8. NerveRTOS1.6.0
9. NerveRTOS1.6.7
10. NerveRTOS1.6.8

NERVEOS 系统的开发人员

开发：阿顿

测试：阿顿

手册信息

编写日期	2021 年 12 月
手册版本	第一版修订版
编辑	阿顿
审核	阿顿

0.目录

1.了解

1.1 什么是实时操作系统

实时操作系统（RTOS）是指当外界事件或数据产生时，能够接受并以足够快的速度予以处理，其处理的结果又能在规定的时间之内来控制生产过程或对处理系统做出快速响应，调度一切可利用的资源完成实时任务，并控制所有实时任务协调一致运行的操作系统。提供及时响应和高可靠性是其主要特点。

1.2 NERVE RTOS 可以干什么

主要实现多任务的运行

1.3 NERVE RTOS 的主要结构

1.4 工程文件结构

```
listing -keil5信息(.M51 .lst等)
Objects -目标文件(.Hex .obj等)
Uvproj  -keil5工程文件(. uvproj . uvport等)
Readme  -参考文档、手册资料(.ppt .pdf等)
User    -用户层      -主要用来放置Main.c文件
App     -应用层      -用户自己开发的通过系统来运行的程序
        |-- App      -未启用应用层文件
OS      -操作系统层 -为操作系统程序 (Nerve RTOS  主要文件就在这)
        |-- OS       -未启用操作系统层文件
Func    -功能层      -通过软件或者硬件（再或者是软硬结合）实现一些功能的程序
        |-- Func     -未启用功能层文件
Drive   -驱动层      -直接驱动硬件(寄存器、IO口等)的程序
        |-- Drive    -未启用驱动层文件
ASM     -汇编语言    -内核启动、任务调度的程序
        |-- ASM      -未启用汇编语言 文件
```

1.5 系统的主要文件

```
System_Clock.h/.c    - 系统时钟文件 ，提供基本的系统时钟服务
System_Control.h/.c - 系统管理文件 ，提供系统初始化、系统控制等服务
System_Task.h/.c     - 系统任务文件 ，提供基本的任务管理、调度等服务
System_Uart.h/.c     - 系统串口通信文件 ，提供系统的串口通信服务
CPU.h/.c             - 任务系统内核，提供系统任务上下文切换
```

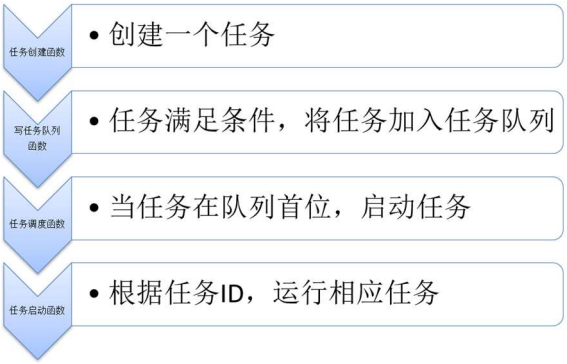
1.6 系统基本的外设支持

```
必备的:
    Timer_x : 提供系统时钟
可选的:
    UART_x : 提供串口通信
    WDT : 提供看门狗
    Power : 提供硬件系统管理
```

1.7 系统对内存的要求

存储空间（FLASH/ CODE）	随机存储器（RAM / DATA）	扩展随机存储器（SRAM / XDATA）
11Kbyte	50Byte	2200Byte

1.8 任务在系统中运行过程



1.9 任务的类型

类型	代号	说明
定时任务	0x00	定时触发
条件任务	0x01 ~ 0x1E	中断或异常触发
空任务	0xFF	手动触发

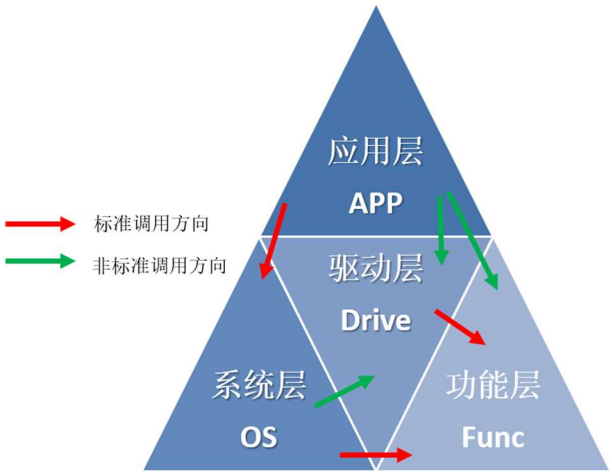
1.10 任务容量

0x00 ~ 0x20 (共 32 个任务)

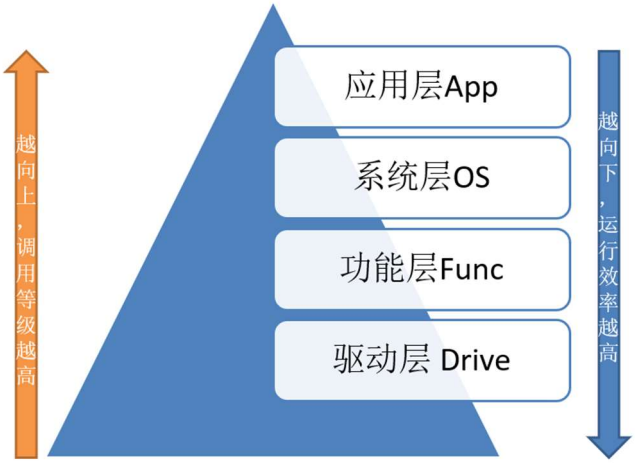
系统任务也占用了一些任务 ID

任务 ID	用途
30	重置看门狗
29	系统重启并下载模式
28	串口接收完成
27（临时占用）	上电测试
0	Main 函数

1.11 各部分程序的调用关系与架构



上图为各部分之间的程序的调用关系



上图为系统基本架构

2.开始

2.1 创建任务

创建任务，需要 System_Task.h/.c 文件中 TaskLoad 函数

函数原型：TaskLoad(u8 TaskId,u8 TaskType,u8 PY,u16 TaskM,u16 TaskPC)

参数名称	参数含义	参数范围
TaskId	任务 ID	0x00 ~ 0xFF
TaskType	任务的类型	0x00 ~ 0xFF
PY	任务的类型优先级	0x01 ~ 0xFF
TaskM	任务的参数	0x0000 ~ 0xFFFF
TaskPC	函数地址	0x0000 ~ 0xFFFF

示例： TaskLoad(1,0,1,10,test);//为 test 函数开启了，时间为 10ms，优先级为 1，ID 为 1 定时任务

为便于标记与使用，任务的类型与优先级都已使用宏定义，具体内容如下图所示

```
/*任务类型定义*/
//定时任务
#define Task_Time 0x00
//条件任务
#define Task_EX0 0x01
#define Task_UART1_RI 0x02
#define Task_UART1_TI 0x03
#define Task_I2C 0x04
#define Task_PCA 0x05
#define Task_PCA0 0x06
#define Task_PCA1 0x07
#define Task_PCA2 0x08
#define Task_PCA3 0x09
#define Task_ADC 0x0A
#define Task_LVDF 0x0B
#define Task_UART1_RIDAT 0x0C
#define Task_CMPL 0x0D
#define Task_CMPH 0x0E
#define Task_PWMCF 0x0F
#define Task_PWM0 0x10
#define Task_PWM1 0x11
#define Task_PWM2 0x12
#define Task_PWM3 0x13
#define Task_PWM4 0x14
#define Task_PWM5 0x15
#define Task_PWM6 0x16
#define Task_PWM7 0x17
#define Task_PWMFD 0x18
#define Task_UART2_RI 0x19
#define Task_UART2_TI 0x1A
#define Task_UART3_RI 0x1B
#define Task_UART3_TI 0x1C
#define Task_UART4_RI 0x1D
#define Task_UART4_TI 0x1E
#define Task_TIME0 0x1F
#define Task_TIME1 0x20
#define Task_TIME2 0x21
#define Task_TIME3 0x22
#define Task_TIME4 0x22
//空任务
#define Task_void 0xFF
```

任务类型

```
/*优先等级定义*/
#define Task_Class_L1 0x01//1级（用户级）
#define Task_Class_L2 0x02
#define Task_Class_L3 0x03
#define Task_Class_L4 0x04
#define Task_Class_L5 0x05
#define Task_Class_L6 0x06
#define Task_Class_L7 0x07
#define Task_Class_L8 0x08
#define Task_Class_L9 0x09
#define Task_Class_H1 0x0A//10级（系统级）
#define Task_Class_H2 0x0B
#define Task_Class_H3 0x0C
#define Task_Class_H4 0x0D
#define Task_Class_H5 0x0E
#define Task_Class_H6 0x0F
```

任务优先级

注：系统级与用户级无区别

使用宏定义，优化后的代码： TaskLoad(1,Task_Time, Task_Class_L1,10,test);

2.1.1 创建任务时，注意事项

1.对于周期循环定时型的任务，单次周期时长不要低于 10ms

2.对于实时性要求很高的程序，不建议通过系统来运行

2.2 注销任务

注销任务，需要 System_Task.h/.c 文件中 TaskRemove 函数

函数原型：TaskRemove(u8 TaskId)

参数名称	参数含义	参数范围
TaskId	任务 ID	0x00 ~ 0xFF

示例： TaskRemove (1);//注销 ID 为 1 的任务

2.3 手动启动任务

手动启动任务，需要 System_Task.h/.c 文件中 OpenTask 函数

函数原型：OpenTask(u8 TaskID)

参数名称	参数含义	参数范围
TaskID	任务 ID	0x00 ~ 0xFF

示例：OpenTask(1);//启动 ID 为 1 任务

2.4 指定优先级启动任务

指定优先级启动任务，需要 System_Task.h/.c 文件中 OpenTaskS 函数

函数原型：OpenTaskS(u8 TaskID,u8 Py)

参数名称	参数含义	参数范围
TaskID	任务 ID	0x00 ~ 0xFF
Py	任务的类型优先级	0x01 ~ 0xFF

示例：OpenTaskS (1, Task_Class_L2);//以优先级为 2 的方式启动 ID 为 1 任务

2.5 使用系统的延时功能

使用系统的延时功能，需要 System_Clock.h/.c 文件中 TaskDelayMs 函数

函数原型：TaskDelayMs(u16 TimeMs)

参数名称	参数含义	参数范围
TimeMs	延时时长（单位：毫秒）	0x00 ~ 0xFFFF

示例：TaskDelayMs(20);//延时 20 毫秒

注意：些延时与软件延时不同，当使用延时后，当前的任务将会挂起，等待指定延时时长后，才会被启动

2.6 使用系统的区域保护功能

2.6.1 开启任务保护

开启任务保护，需要 System_ Task.h/.c 文件中 EnableProtection 函数

函数原型：EnableProtection(u16 r1)

参数名称	参数含义	参数范围
r1	延时时长（单位：毫秒）	0x00 ~ 0xFFFF

示例：EnableProtection (20);//当任务保护时间设为 20 毫秒

2.6.2 取消任务保护

取消任务保护，需要 System_ Task.h/.c 文件中 DisableProtection 函数

函数原型：DisableProtection()

参数名称	参数含义	参数范围
void	空	空

示例：DisableProtection ();//取消任务保护时间设为 20 毫秒

2.7 条件任务触发接口

取消任务保护，需要 System_ Task.h/.c 文件中 TypeTaskStartupn 函数

函数原型：TypeTaskStartupn (u8 TYPE)

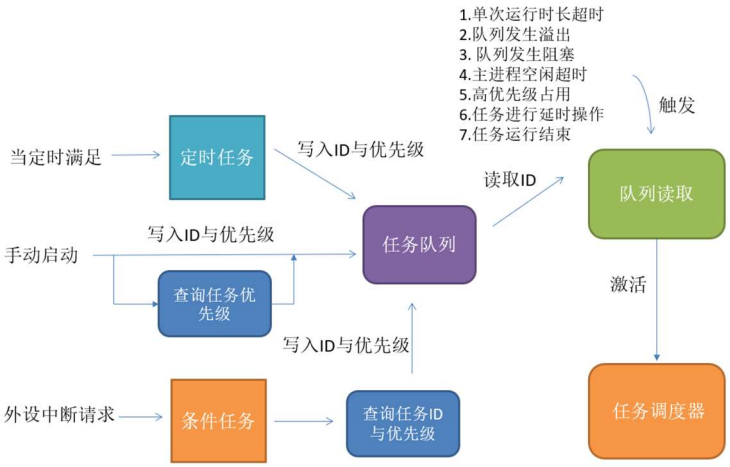
参数名称	参数含义	参数范围
TYPE	任务类型	空

示例：TypeTaskStartupn (ADC);//启动与 ADC 相关的任务

3.深入

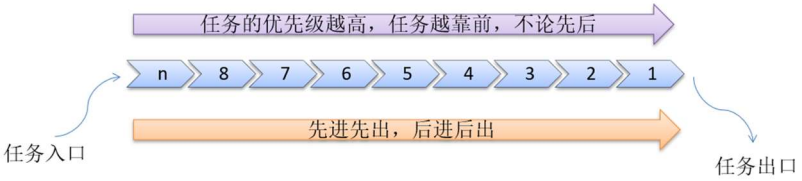
3.1 了解基本的

3.1.1 系统的任务系统



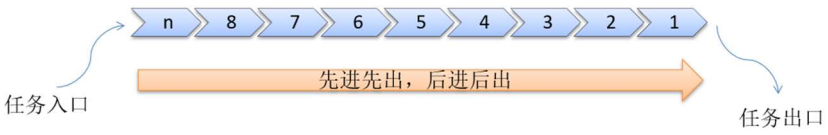
这张图，大概描述了任务系统，将任务运行的方式，分为了三大类，分为定时触发、手动触发和条件触发。任务系统的重点，主要在队列方面，而任务队列分为普通任务队列和中断型任务队列

普通型任务队列：主要对任务的时序与优先级管理，具体管理方式，如图所示。



•每次输入任务和输出任务前，系统自动对任务队列进行检查，修复异常

中断型任务队列：用于存放因为某些原因，被强制打断的任务，中断型任务队列的中任务永远比普通型任务队列的优先级要高。中断型任务队列内，没有优先级管理，只有时序管理。比普通型任务队列，运行起来要快的多。



•中断型队列中优先级高于普通任务队列任何任务的优先级

这两个队列的具体的定义在“System_task.h”文件中

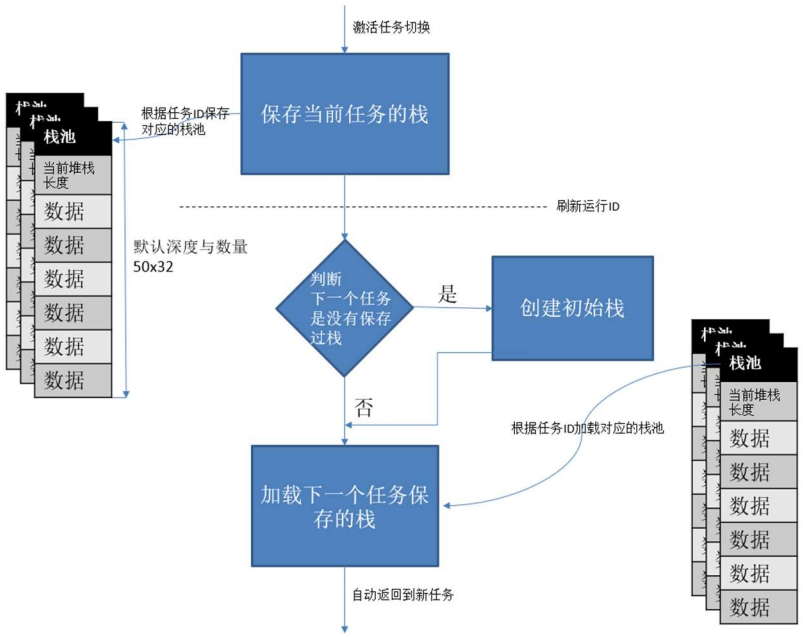
```
extern u8 xdata TaskInterruptQueue[TaskQueueMax + 2];//中断型任务队列

/*普通任务队列表*/
typedef struct
{
  u8 ID;//应用ID
  u8 Py;//优先级
}xdata taskqueue;
extern taskqueue TaskQueue[TaskQueueMax + 2];
```

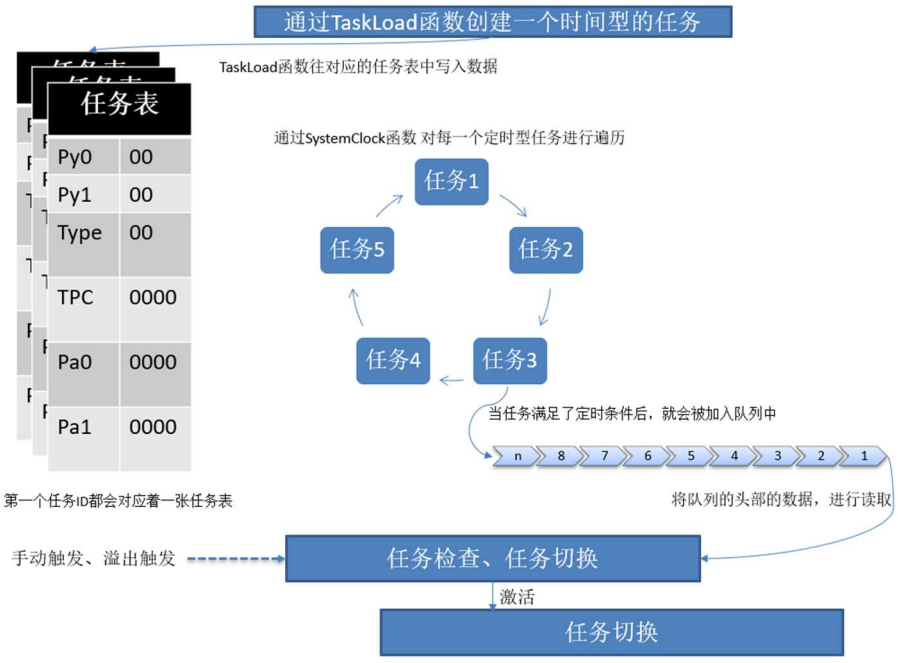
从定义中，我们可以看出中断型任务队列与普通型任务队列的长度是相同的，都由 TaskQueueMax 进行定义，那我们为什么对最大长度，还要加二呢？这么做，是为了保证队列的尾部有空隙，以免意外发生的溢出。

下面我们即将进入，系统的"大动脉"了，就是任务的上下文切换。

任务上下文切换，主要交给 TaskSwitch 函数来完成。其大概具体流程如上图所示。其函数存放在"CPU.c"文件中。

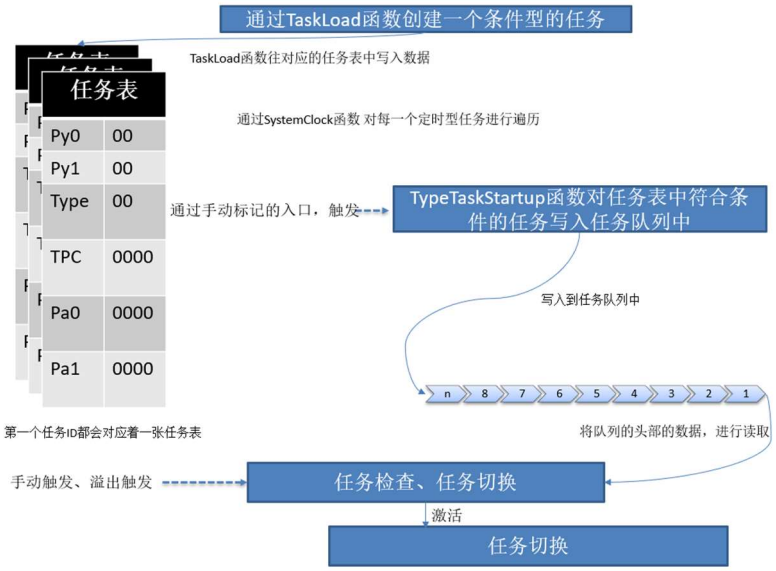


下面分别以时间型任务与条件型任务为例，讲述下，大概运行机制。



上图为时间型任务的运行大概的原理，其中的时钟系统为核心，具体内容请参考下一节的"系统的时钟系统"中内容。

下面，让我们看一下条件型条件的运行大概原理

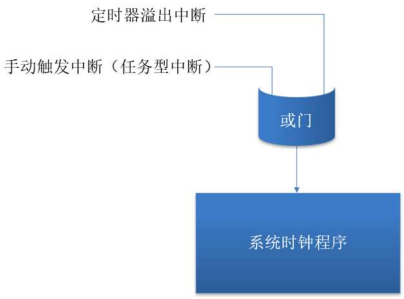


与时钟型任务的运行机制差不多，依赖于手动触发，在原有手动触发上加上任务对照检查。

注意：任务表默认开启弹性检查（弹性检查：根据占用最大数量，进行实时的计算，降低在进行遍历与查询操作的时间开销），如需要可在 System_Contlor.h 文件中将对其的宏定义注释即可。

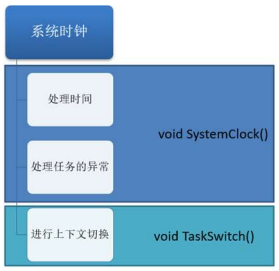
3.2 系统的时钟系统

本系统的时钟，分为两种触发中断方式，通过" TimerIRQTouchType"参数用于判断类型，只有参数为零时，才可判断为定时器溢出触发的中断。



系统时钟程序，不仅仅处理时间，还要处理任务系统的异常，进行任务的上下文切换，可谓是整个系统的核心。

系统时钟的程序，由"SystemClock"和"TaskSwitch"两个函数构成，其作用如下图所示。



除此以外，还有几个系统时钟的参数，需要留意。

```
extern u8 xdata AppRunTimer;//应用运行时间
extern u16 xdata SystemTime;//系统时钟
extern u32 xdata RUNTime;//连续运行时间
extern u16 xdata Task0Time;//任务0最大挂起时长
extern u16 xdata ProTime;//任务保护时长
```

其中 SystemTime、RUNTime 都可以直接使用。

3.3 系统的移植与剪裁

在进行移植之前，我要了解系统基本情况。

硬件方面：必须提供一个系统时钟。

Timer_x : 任意一个定时器，用于提供系统时钟（必选）。

UART_x　　: 任意一个串口，提供串口通信（可选）。

WDT　　: 看门狗一个，用于宕机复位（可选）。

Power　　: 提供硬件系统电源管理，用于系统对 MCU 复位，进入烧录模式等（可选）。

所有系统相关的软硬件，都会在 System_Init 函数中完成初始化操作。

在 System_Control.h 文件中，可指定用于硬件初始化函数入口。

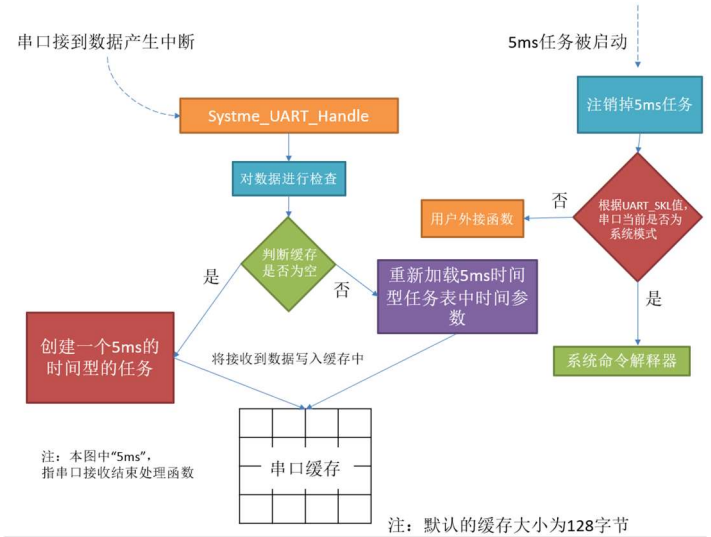
```
/*-------------------------------------函数-----------------------------------------*/
/*指定系统定时器初始化函数*/
#define System_Timer_Init {Timer0Init();}
/*指定系统串口初始化函数*/
#define System_UART_Init {UART1Init();}
/*指定系统看门狗初始化函数*/
#define System_WDT_Init {WDT_Init();}
```

在 System_Control.h 文件中，可配置启用的硬件功能。

```
/*-------------------------------------开关-----------------------------------------*/
/*配置是否使用串口功能*/
#define EnableSerialPortDebugging//启串口功能
/*配置是否使用看门狗功能功能*/
#define EnableWDT//启看门狗功能
```

一般在 System_Control.h 文件中，配置完成后，基本不用修改 System_Init 函数了。

3.4 系统的串口系统



上图系统自带的串口系统基本工作原理，串口系统的主要代码存放于 System_Uart.c/.h 文件中，如果不需要的话，可以在 System_Control.h 文件中，可配置启用的硬件功能。

负责串口系统主要由 Systme_UART_Handle 与 Systme_UART_Input 两大函数负责。

串口缓存的大小与串口响应时间，可以在 System_Uart.h 中进行配置。

```
#define UART_DAT_Max 128//串口缓冲区的数据最大长度
#define UART_TIMES 5//响应时间（ms）
```

3.5 绍系统串口的基本使用

系统串口在初始化以 115200bps 进行的初始化，使用时需要将串口调试软件的波特率调至 115200，才可正常使用。

下表是所有串口系统所有的命令以及其含义。

命令	命令含义
OU	使用串口输出字符串
RST	进位单片机复位
ISP	使单片机进入到 ISP 模式
OFT	关闭任务系统
PHI	使单片机进入高功耗模式
OFW	关闭看门狗
OPT	启动某个任务
QTS	查看任务表中所有的任务
QST	查看系统时间参数
QRT	查看系统运行时长
PPD	使单片机进入掉电模式
INFO	查看系统信息
OPTU	开启系统监视器
OFTU	关闭系统监视器
PIDL	使单片机进入掉电模式
PLOW	使单片机进入低功耗模式
HELP	列出所有串口命令

3.6 其他文件的介绍

3.6.1 Inc_Basic.h 文件

```
#include "STC8F.h"//STC8A8K64S4A12单片机.头文件
#include "Inc_DataType.h"//基本数据类型定义.头文件
#include "Inc_ActionValue.h"//操作值宏定义.头文件
#include "NOP.h"//空操作.头文件
```

用于存放与加载基本，通用的头文件

3.6.2 Inc_DataType.h 文件

```
/*--------------------------------无符号字符型--------------------------------------*/
typedef unsigned char       u8;
typedef unsigned char       uchar;
typedef unsigned char       UINT8;
typedef unsigned char       uint8_t;
```

用于基本数据类型的定义

3.6.3 Inc_ActionValue.h 文件

用于操作值宏定义

3.6.4 NOP.h 文件

```
extern void _nop_(void);
```

用于空指令函数定义

3.6.5 NOP.A51 文件

存放 NOP 函数的语句体

3.6.6 STARTUP.A51 文件

存放 8051 单片机的启动文件

4.结束

在开发之前，你要知道以下几点

1.我们已经完全适配了 STC8A8K64S4A12@22.1184MHz[1]

2.你的单片机至少要有 11KByte 的程序空间和至少 50Byte 的 RAM 以及 2200Byte 的 SRAM，这样系统才能正常运行 [2]

3.你的单片机最好支持 1T 或者更快 [3]

4.请使用 keil5 来开发

5.关于系统的详细内容请参考《NerveRTOS 开发指南》

注释：

1.目前只适配了 STC8A8K64S4A12@22.1184MHz，不保证 STC8A8K64S4A12 的库正确性、稳定性、完整性

2.以上的硬件配置只能保证系统正常运行，不预留用户程序的空间

3.6T/12T 单片机也可以运行系统，速度和稳定性可能较低

感谢您 Nerve RTOS 的支持，也希望 Nerve RTOS 对于您的学习或者开发有所帮助