桂林电子科技大学
GUILIN UNIVERSITY OF ELECTRONIC TECHNOLOGY

单片机原理及应用
课程设计说明书

基于 51 自编小型实时操作系统

学　　院：计算机科学与工程学院

专　　业：计算机科学与技术

班　　级：11003102

姓　　名：罗盛明

学　　号：1100310218

指导教师：俸 皓

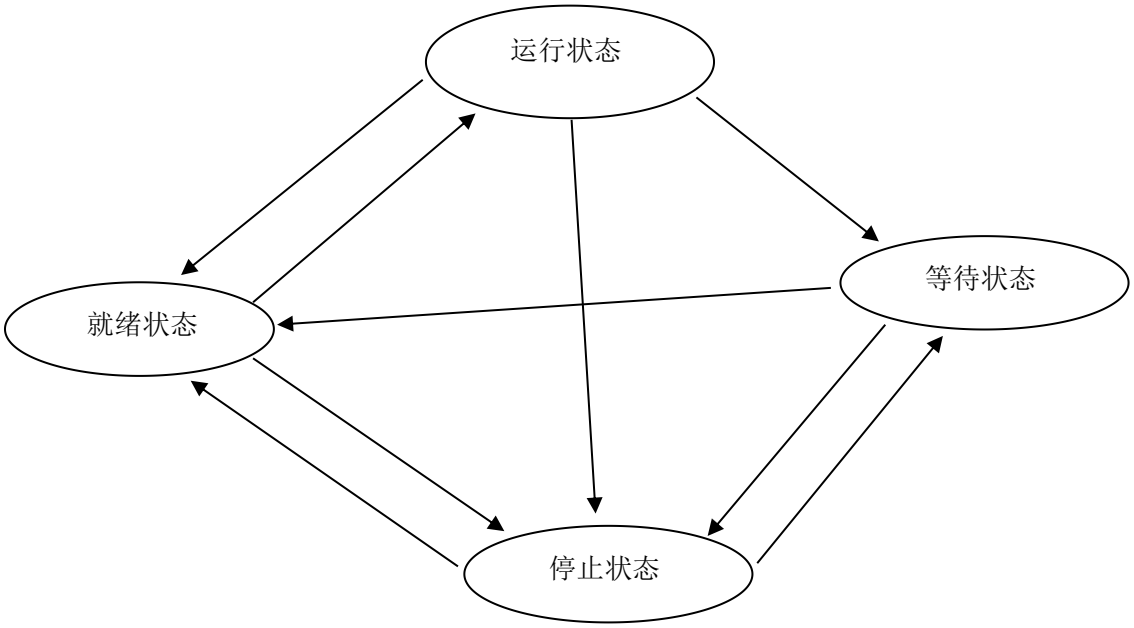
2014-6

简言：

本次的单片机课程设计我选择的是自编一个基于 51 的小型操作系统内核，我将之取名为--FMOS，该内核的特点有：多任务、具有实时性、时间片与优先级抢占结合的调度算法、任务管理、时间管理等，还具有良好的扩展性，根据需要可以对系统进行配置，当 FMOS 内核目前仅支持静态创建任务，由于 51 片内内存有限，在没有外部 RAM 的情况下能够支持 5~6 个任务，2～3 信号量。

FMOS 内核框架:

该内核框架里，进程一共具有四种运行状态，任何一个任务在任何时刻都只能处在这四个状态中的一种。



进程状态的切换只能通过两种方式进行，除此之外别无他法，一是任务级调度器，二则是中断级调度器，前者是一个系统调用，后者是嵌在系统时钟中断服务的一段代码。

就绪-->运行：进程得到运行的条件除 CPU 外其他都具备，在就绪队列中等待调度器调度，当被调度器选中后，保存上一个进程现场，加载选中的进程的上下文，进程获得 cpu 使用权，任务在其时间片内运行。

运行-->就绪：进程时间片用完后，但仍具有就绪运行状态，进程只需要再次获得时间片就能继续运行，这时中断级调度器将用完时间片的进程放就绪队列的尾部，同时从运行队列中选出新的就绪进程进入运行。

运行-->等待： 任务延迟等待或任务等待信号或等待中断的时间应该让出 cpu，让其他进程得以运行，提高 cpu 利用率，进入等待时，放弃自己的时间片，进入相应的等待队列，并且调度新的任务进入运行。

等待-->就绪：当延迟时间到后，或者等待的信号或中断来后，进程重新回到就绪队列，如果是延迟结束，进程就被放到普通就绪队列，如果是等待信号或中断，则进

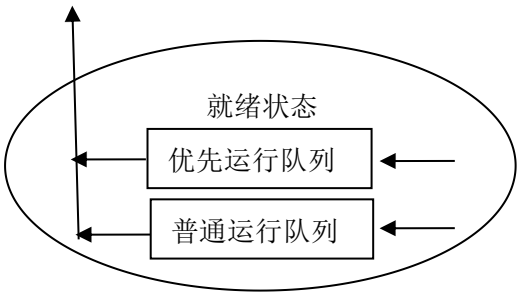
程被放入优先运行队列，优先得到运行。

运行-->停止：当进程自己杀死自己的时候，就从运行状态进到停止状态，除非是有其他进程将其激活，否则将永远得不到运行机会。

就绪-->停止/等待-->停止：如果进程已经在就绪运行队列里或在等待队列里，但是正在运行的任务（进程）将其放入停止状态，这个进程就不再调度器调度范围内，延迟计数也将停止计数，如果不被其他任务唤醒，则永远无法运行。

停止-->就绪/停止-->等待：正在运行的任务可以唤醒处在停止状态的进程，被唤醒的进程重新回到被停止之前的状态，即“从哪里来回到哪里去”，若之前是延迟等状态，则回到等待状态后，继续延迟计数。

时间片结合优先级抢占式的调度机制



FMOS 内核设计有两个就绪运行队列，分别是
--先运行队列--和--普通运行队列--；
顾名思义，优先队列中的进程优先级要比普通运行队列里要高，就是说，不管是任务级调度器还是中断级调度器都是先从优先运行队列中取出进程然后分配时间片进入运行，只有当优先运行队列没有进程时调度器才会到普通运行队列中取进程；队列内部是按时间片调度运行的，

抢占体现在：优先运行队列中没有优先进程，而此时运行的是普通进程，其时间未用完，如果这时有优先进程进入优先运行队列，那么在下一个系统时钟周期（时间片由若干个系统时钟周期组成）到来时，中断级调度器将从优先运行队列取出那个优先进程进入运行，并将当前运行的普通进程放优先运行队列的尾部，以保证当优先进程运行结束后继续运行原来那个普通进程。直到时间片用完。 抢占就在下一个系统时钟周期进行进程切换，而不是等当前进程时间片用完后进行优先调度。这种抢占方式保障了系统的实时效果。

什么样的进程才能被放进优先运行队列呢？

在 FMOS 内核中，有两种进程，这里先描述两个概念：优先级进程和普通级进程

*优先级进程：获得优先级标志，即优先寄存器中是优先级标志。
*普通级进程：没有获得优先级标志，即优先寄存器中是普通级标志。

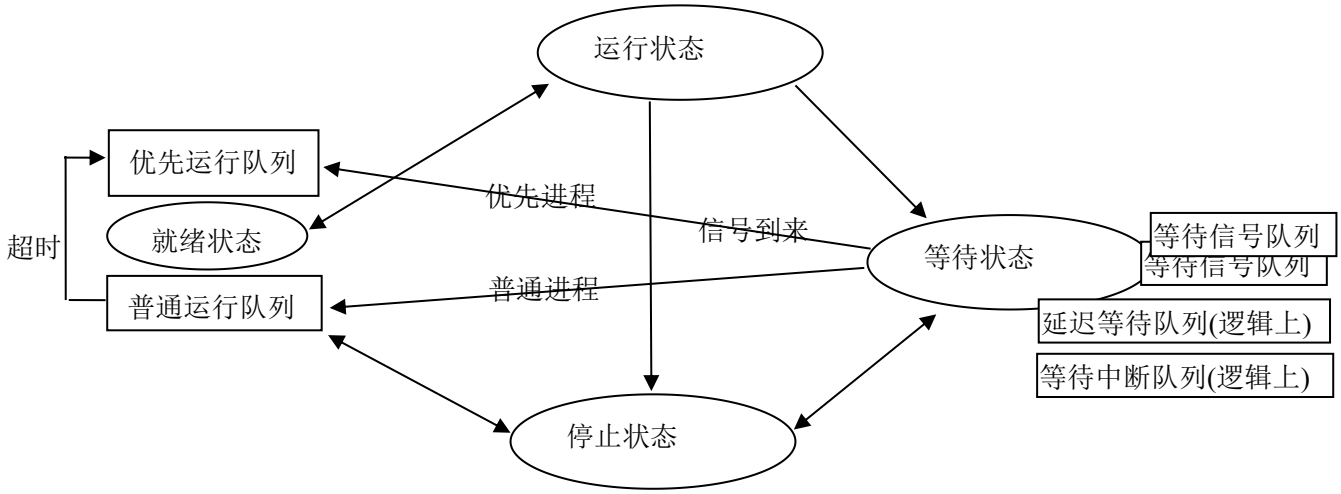
能够进入优先运行队列的进程以下几种进程：

（1）普通进程申请获得优先级变成优先进程，即实时性要求比较高的任务，可通过申请优先级标志，获得优先级的进程进入就绪队列时只会进到优先运行队列，永远不会进入普通运行队列，由于有了抢占机制，从而保证优先进程的实时要求。优先进程可放弃优先级标志变回普通进程。

（2）普通进程在一些条件下也能进入优先运行队列，但它并不是优先进程，比如：正在运行的普通进程被优先进程抢占，当由于时间没有用完，仍具有再次运行的条件，在优先进程运行时间片完后应当得以继续运行，但要保证这个被抢占的普通级进程 优先于普通运动队列里其他的进程，则将这个被优先进程抢占的普通暂时放到优先运行队列尾部中，当所有优先进程运行完后这个普通进程继续运行。

（3）等待信号或中断的进程不管是优先进程还是普通进程，一旦信号或中断产生，相应的进程都被放进优先运行队列，以保证实时要求。

（4）进入延时等待超时的普通任务（进程）；当一个普通进程进入延迟等待的时间结束后，被中断级调度器放入普通运行队列，但由于普通运行队列中前面的有多个进程都在等待或者优先运行队列一直有进程，导致延迟等待的进程超时，这时中断级调度器将延时超时的进程放入优先运行队列，保证不会超时太久。

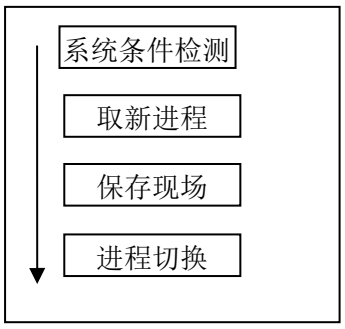


调度器结构

一、**任务级调度器**　在 FMOS 内核中声明为：void sys_dispatch(void);

任务级调度器实质上是一个仅供内核其他内核模块调用的接口(内核 API)，例如：用户 API 中的延迟方法 void delayBySysClockCycle(data16 time)；进程进入延迟后要放弃时间片并同时进行任务级调度，让新的进程得以运行。

任务级调度器不能被用户的应用程序直接使用，否则会导致系统紊乱。同样其他内核 API 只能在内核级别里使用。



任务级调度器功能有：

1）调度锁上锁和解锁，保证不会在进程调度时因中断而陷入混乱，但系统时钟服务仍可以更新延迟进程的等待时间，保障延迟精确。

2）条件检测，包括系统运行标志检查、系统运行模式检查、当前进程时间片检查等，当其中任何一个条件不满足是，任务级调度器立即退出进程调度。

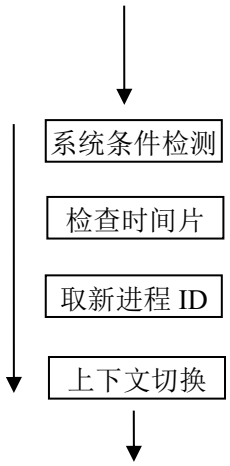
3）当上述条件都满足后，接下来是从就绪队列中取出新进程，优先运行队列中若

有优先进程则取出，否则从普通运行队列中取新进程，若两个队列都没有进程，则系统将运行系统的空任务进程。

4）取出新进程 ID 后，系统要给新进程分配时间片，并将旧进程的上下文压入其任务栈，更新任务控制块中的相应寄存器，接着将从新进程的任务栈中弹出，布置新进程的运行上下文，进程上下文切换完成，开始新的任务运行。

二、中断级调度器

中断级调度级是由嵌入在系统时钟中断服务里的一段代码构成，其所完成的功能和任务级调度器的功能相差不多，不同的是中断级调度器不能显示调用，它是系统时钟中断服务必不可少的部分之一，也是系统能够并发执行多个任务的本质所在，在系统时钟中断服务除了完成中断级调度器功能外还要完成系统时钟粒度更新功能、延迟等待进程的延迟时间的更新、监督时间片，监督信号量和中断等；



中断级调度级功能和任务级调度器功能差不多，不同的是它们一个是在普通函数（内核 API）中实现，而一个是在中断服务中实现，一个能被系统内核模块调用，另一个则不能，再者，中断级调度器的条件检查更加严格和复杂。当进程时间片用完后要取新进行，然后进行进程切换。

当条件检测不满足时同样不会进行进程调度，如果当前任务的时间片没有用完，则也不能进行调度，除非当前进程是普通进程，并且优先运行队列中有进程，这时要进行抢占普通进程运行，优先进程之间不能抢占，普通进程之间不能抢占，只有优先进程才能抢占正在运行的普通进程。

FMOS 内核解析 （结合代码）

要将内核解析，应该先从其各种量的命名规则开始，这对读其他系统内核解析都是必要的前提；理解系统的初始化过程对整个系统的机制

一）系统内核命名规则：

由于操作系统本身涉及到诸多个状态、标志、变量、方法等取名问题，统一的命名规则会给系统内核不管是设计阶段还是实现阶段都带来极大的好处，而且再后期扩展和维护过程当中极其体现了统一命名的优势。

FMOS 内核有文件命名规则、数据类型命名规则、标志宏命名规则、变量命名规则、函数命名规则、配置变量命名规则等。

前缀____

sys_　属于系统的变量或标志常量或 API 函数等

pro_　属于进程的变量或标志常量或函数等

sig_　需要信号的变量或标志常量或函数等

例如：PRO_RUN_STCO
进程状态中的：就绪运行标志

____后缀

_fi　(file) 表示文件

_st　(state) 表示状态量

_va　(value) 表示变量

_co　(connstant) 表示常量 或全大写

_cfg　(configure)　表示配置量

_mo　(mode) 表示模式量

_open　开

_clos　关

_lock　上锁

_unlock　解锁

sys_macro_def_fi.h　‘系统’宏定义‘文件’
注意：这些后缀是可以组合的　如：
SYS_RUN_STCO　‘系统’运行‘标志’‘常量’
SYS_THREAD_MOCO　‘系统’线程‘模式’‘常量’
process_number_cfg 任务总数‘配置’
sys_runstatus_stva 用来保存‘系统’运行‘状态’的‘变量’

二）FMOS 系统主要的数据结构

系统控制管理 数据结构

```
typedef struct sys_manage_control_structure
{
    data8 sys_runstatus_stva;        //用来保存系统运行状态标志变量
    data8 sys_runmode_mova;          //用来保存系统运行模式变量
    data8 sys_priorityrun_va;        //用来保存优先运行队列里优先进程数的变量
    data8 sys_waitqueue_va;          //用来保存等待队列里进程数变量
    data8 sys_dispatch_lock_siva;    //调度锁
    data8 sys_time_slice_va;         //时间片计数器
    data8 sys_currun_id;             //系统正在运行的进程 ID
    data8 sys_nexrun_id;             //系统新运行的进程 ID
};
```

进程控制块 数据结构

```
typedef struct pro_control_structure
{
    data8 pro_SP;                //用来保存任务栈的栈顶地址
    data8 pro_priority_stva;     //用来保存该进程的属性，是优先进程还是普通进程
#if overtime_enable > 0
    data8 pro_overtime_va;       //用来保存延迟等待进程允许超时的系统周期数
#endif
    data16 pro_delay_time_va;    //保存延迟时间
    data8 pro_status_stva;       //进程当前状态
    data8 pro_prestatus_stva;    //进程上一个状态
};
```

信号量控制块 数据结构

```
typedef struct signal_control_structure
{
    data8 sig_create_status;     //是否已经被创建
    data8 sig_type;              //信号量类型，是二进制型还是十进制型
    data8 sig_counter;           //信号量计数器
    data8 sig_have_wait;         //是否有进程在等待信号量
    data8 *sig_waitqueue;        //信号量等待队列
};
```

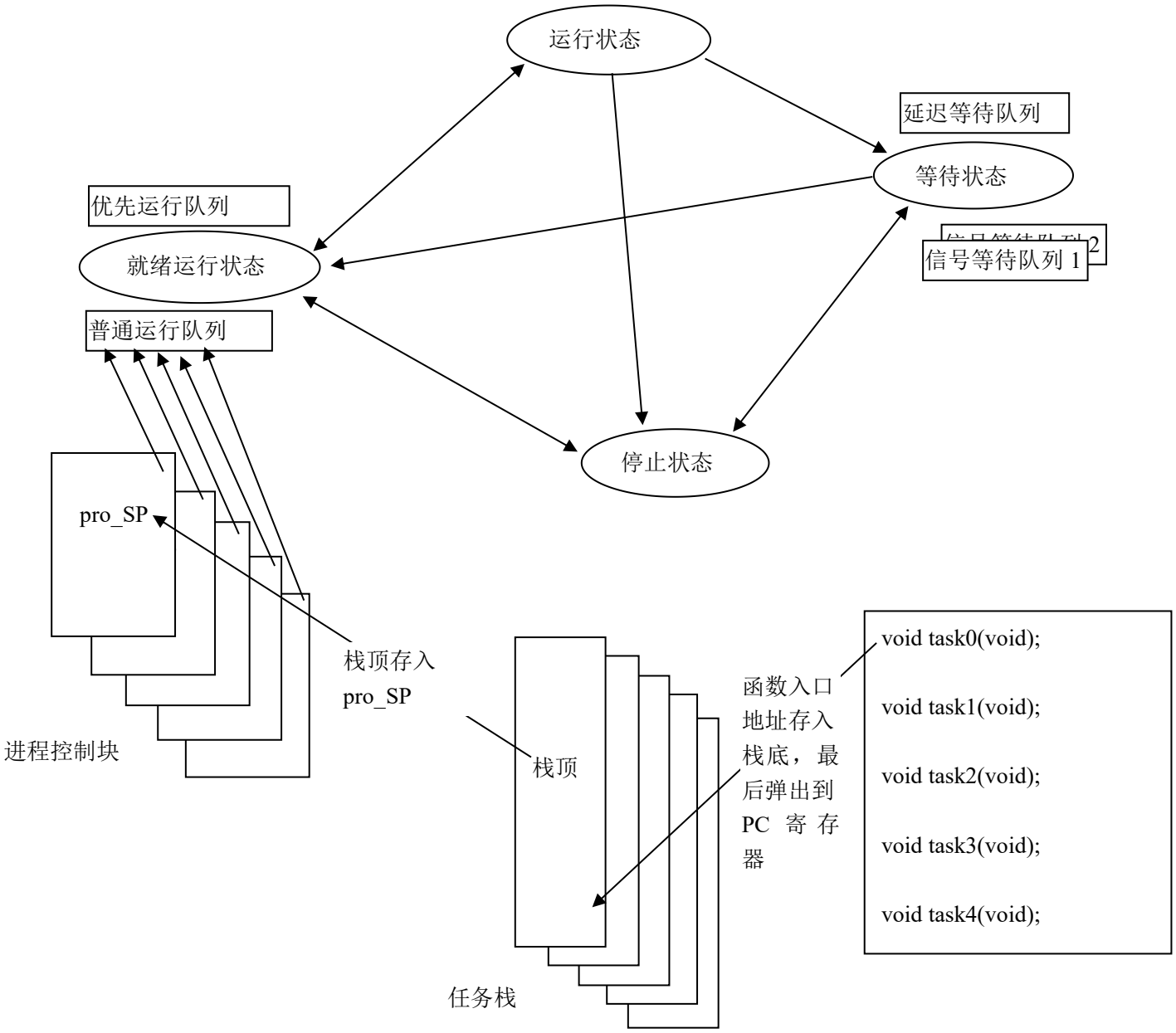
三）系统初始化过程

系统初始化 void OS_init(void) 过程包括：

普通运行队列的初始化	Que_com_initialQueue ();
优先运行队列的初始化	Que_pri_initialQueue ();
进程控制块的初始化	Pro_initialPCS();
系统管理控制块初始化	SYS_initialSMCS();
信号量控制块的初始化	SIG_initialSCS();
系统整体初始化	SYS_initialOS();

四）**系统启动**

系统初始化完成后 就启动系统 OS_start() 随之系统调度器开始调入进程进入运行，然后就进入了自动调度运行了。


运行状态
延迟等待队列
等待状态
优先运行队列
就绪运行状态
信号等待队列 1
普通运行队列
停止状态
pro_SP
栈顶存入
pro_SP
进程控制块
栈顶
函数入口
地址存入
栈底，最
后弹出到
PC 寄 存
器
void task0(void);
void task1(void);
void task2(void);
void task3(void);
void task4(void);
任务栈

系统初始化过程

附件　FMOS 系统主要文件代码

FMOS_Macro_Define_Fi.h　宏定义文件

```
/*************************************************************
*     author:fengming
*     date:2014-4
*************************************************************/
/*------------data type define-------*/
#ifndef datatype_redefine
#define datatype_redefine
typedef unsigned char data8;
typedef signed char sdata8;
typedef unsigned int data16;
typedef signed int sdata16;
#endif
/*-------for sys_manage_control_struct define status and signed constant------*/
/*sys_runstatus_stva: System status values.*/
#ifndef SYS_RUN_STCO
#define SYS_RUN_STCO    0x0f
#endif
#ifndef SYS_STOP_STCO
#define SYS_STOP_STCO    0xf0
#endif
/*sys_runmode_mova: System operating mode value.*/
#ifndef SYS_THREAD_MOCO
#define SYS_THREAD_MOCO 0x0f
#endif
#ifndef SYS_SERVICE_MOCO
#define SYS_SERVICE_MOCO 0xf0
#endif
/*sys_interr_ooc_siva: Interrupt is open?*/
#ifndef SYS_INTERR_OPEN
#define SYS_INTERR_OPEN 1
#endif
#ifndef SYS_INTERR_CLOS
#define SYS_INTERR_CLOS       0
#endif
/*sys_dispatch_lock_siva: Scheduling lock is locked?*/
#ifndef SYS_DISPATCH_LOCK
#define SYS_DISPATCH_LOCK 0x01
#endif
#ifndef SYS_DISPATCH_UNLOCK
#define SYS_DISPATCH_UNLOCK 0x00
#endif
```

```
/*------for pro_control_struct define status constant-------*/
/*pro_status_stva:*/
#define PRO_RUN_STCO 0x02
#define PRO_DELAY_STCO 0x04
#define PRO_WAITIN_STCO 0x08
#define PRO_WAITSIG_STCO 0x10
#define PRO_STOP_STCO 0x12

/**High priority tasks are running?**/
#define PRO_PRIRUN_STCO 0xff
#define PRO_COMRUN_STCO 0x00
/**Enter and exit the critical area critical areas.**/
#define SYS_ENTER_CRITICAL() {EA = 0;}
#define SYS_EXIT_CRITICAL() {EA = 1;}

/*semaphore*/
#define SIG_CREATED_STCO 0xff
#define SIG_NOCREATE_STCO 0x00

#define SIG_RELEA_SUCCE_STCO 0xff
#define SIG_RELEA_FAIL_STCO 0x00

#define SIG_APPLI_SUCCESS_STCO 0xff
#define SIG_APPLI_FAIL_STCO 0x00

#define SIG_TYPE_NONE_STCO 0x00
#define SIG_TYPE_LOCK_STCO 0x02
#define SIG_TYPE_DECI_STCO 0x0a
```

FMOS_Configure_Fi.h　系统配置文件

```
/*************************************************************
*     author:fengming
*     date:2014-4
*************************************************************/
/*--------configur the system process number and process stack deep-----*/
#define SYS_TIME_SLICE_cfg 5 /*Time slice*/

#define process_number_cfg 5      /*The total number of processes*/
#define pro_stack_deep_cfg 26      /*Task stack length*/

#define pro_comque_length_cfg 5        /*Ready queue length*/
#define pro_prique_length_cfg 4   /*Priority ready queue length*/

#define sys_clock_TH_cfg 0xD8          /*The system clock granularity*/
#define sys_clock_TL_cfg 0xF0
#define sys_clock_TMOD_cfg 0x01     /*Timer mode is used to configure the system clock.*/

#define overtime_enable 0
#define pro_overtime_cfg 3                    /*Wait timeout reference value*/

#define signal_number_cfg 2
#define signal_queue_length_cfg 4
```

FMOS_Hand_Fi.h　系统头文件

```
/*************************************************************
*     author:fengming
*     date:2014-4
*************************************************************/
#include "FMOS_Macro_Define_Fi.h"
#include "FMOS_Configure_Fi.h"
/*---------define the system control block --------*/
typedef struct sys_manage_control_structure
{
      data8 sys_runstatus_stva;
      data8 sys_runmode_mova;
      //data8 sys_interr_ooc_siva;
      data8 sys_priorityrun_va;
      data8 sys_waitqueue_va;
      data8 sys_dispatch_lock_siva;
      data8 sys_time_slice_va;
      data8 sys_currun_id;
      data8 sys_nexrun_id;
};

typedef struct pro_control_structure
{
      data8 pro_SP;
      data8 pro_priority_stva;
#if overtime_enable > 0
      data8 pro_overtime_va;
#endif
      data16 pro_delay_time_va;
      data8 pro_status_stva;
      data8 pro_prestatus_stva;

};
/*----------define the system run queue -------*/
typedef struct run_queue
{
            data8 queue_length;
            data8 queue_front;
            data8 queue_rear;
            data8 *queue_data;
};
```

```
typedef struct signal_control_structure
{
	data8 sig_create_status;
	data8 sig_type;
	data8 sig_counter;
	data8 sig_have_wait;
	data8 *sig_waitqueue;
};

/*---------system initial function declare------*/
void OS_start();
void OS_stop();
/*----------------------------------------------*/
data8 OSTaskCreate(data8 id, data8 *taskStack, void (*task));

/************************* BootLoad ***********************************/
void bootload(void);
/*------------sys_manage_control_struct operation function declare-----*/
void OS_init(void);
void SYS_initialOS(void);
void SYS_initialSMCS(void);
void Pro_initialPCS(void);

void Que_com_initialQueue (void);
void Que_pri_initialQueue (void);
/*****************************************************************************/
void Que_com_inQueue (data8 id);
void Que_pri_inQueue (data8 id);

data8 Que_com_ouQueue (void);
data8 Que_pri_ouQueue(void );

void Que_com_rmQueue(data8 id);
void Que_pri_rmQueue(data8 id);

data8 Que_check_ComQueue(data8 id);
data8 Que_check_PriQueue(data8 id);

/*****************************************************************************/
void SIG_initialSCS(void);

void sig_regist_WaitQueu(data8 sig_id,data8 pro_id);
void sig_delete_WaitQueue(data8 sig_id,data8 pro_id);
data8 sig_activate(data8 sig_id);
```

```
data8 sig_create(data8 sigtype,data8 initvalue);
data8 sig_apply_jam_time(data8 sig_id,data16 jam_time);
data8 sig_apply_jam(data8 sig_id);
data8 sig_apply_unjam(data8 sig_id);
data8 sig_release(data8 sig_id);
/*************************************************************************/
void sys_dispatch(void);

/*************************************************************************/
//void sys_systemClock_initial(void);
void sys_systemClock_on(void);
void delayBySysClockCycle(data16 time);
void sys_delay_return(data8 id);
/*************************************************************************/
void sys_stop_process(data8 id);
void sys_recover_process(data8 id);
void sys_wait_interr_signal(void);
void sys_recov_interr_signal(data8 id);

/**********************************************/
void apply_priority(void);
void apply_priority_byid(data8 otherid);

void giveup_priority(void);
void giveup_priority_byid(data8 otherid);
/*------task process declare----------------------------------*/
void task0(void);
void task1(void);
void task2(void);
#if process_number_cfg >3
void task3(void);
#endif
#if process_number_cfg >4
void task4(void);
#endif
#if process_number_cfg >5
void task5(void);
#endif
```

FMOS_Dispacth_Design_Fi.c　任务级调度器和中断级调度器

```
/*************************************************************
*     author:fengming
*     date:2014-4
*************************************************************/
#include "FMOS_Macro_Define_Fi.h"
#include "FMOS_Configure_Fi.h"
#include "FMOS_Hand_Fi.h"
#include <reg52.h>
  extern struct sys_manage_control_structure idata smcs;
  extern struct pro_control_structure idata pcs[process_number_cfg];
  extern struct signal_control_structure idata sigcs[signal_number_cfg];
  extern struct run_queue idata commom_queue;
  extern struct run_queue idata priority_queue;
/*******************************任务级调度器**********************/
void sys_dispatch(void)
{
      SYS_ENTER_CRITICAL();
      smcs.sys_dispatch_lock_siva = SYS_DISPATCH_LOCK;

      if (smcs.sys_runstatus_stva == SYS_STOP_STCO)
      {
            smcs.sys_dispatch_lock_siva = SYS_DISPATCH_UNLOCK;
            SYS_EXIT_CRITICAL(); return; }
      if (smcs.sys_runmode_mova == SYS_SERVICE_MOCO)
      {
            smcs.sys_dispatch_lock_siva = SYS_DISPATCH_UNLOCK;
            SYS_EXIT_CRITICAL(); return;  }
      if (smcs.sys_time_slice_va > 0 )
      {
            smcs.sys_dispatch_lock_siva = SYS_DISPATCH_UNLOCK;
            SYS_EXIT_CRITICAL(); return;  }

      SYS_EXIT_CRITICAL();

            if (smcs.sys_priorityrun_va > 0)
            {
                  smcs.sys_nexrun_id = Que_pri_ouQueue();
            }
            else
            {
                  smcs.sys_nexrun_id = Que_com_ouQueue();
            }
```

```
    SYS_ENTER_CRITICAL();
    smcs.sys_time_slice_va = SYS_TIME_SLICE_cfg;

    if (smcs.sys_currun_id != smcs.sys_nexrun_id)
    {
        __asm PUSH ACC
        __asm PUSH B
        __asm PUSH DPH
        __asm PUSH DPL
        __asm PUSH PSW
        __asm PUSH AR0
        __asm PUSH AR1
        __asm PUSH AR2
        __asm PUSH AR3
        __asm PUSH AR4
        __asm PUSH AR5
        __asm PUSH AR6
        __asm PUSH AR7

        pcs[smcs.sys_currun_id].pro_SP = SP;
        smcs.sys_currun_id = smcs.sys_nexrun_id;
        smcs.sys_nexrun_id = 0;
        pcs[smcs.sys_currun_id].pro_prestatus_stva = PRO_RUN_STCO;
        #if overtime_enable > 0
        pcs[smcs.sys_currun_id].pro_overtime_va = pro_overtime_cfg;
        #endif
        SP = pcs[smcs.sys_currun_id].pro_SP;

        __asm POP AR7
        __asm POP AR6
        __asm POP AR5
        __asm POP AR4
        __asm POP AR3
        __asm POP AR2
        __asm POP AR1
        __asm POP AR0
        __asm POP PSW
        __asm POP DPL
        __asm POP DPH
        __asm POP B
        __asm POP ACC

    }
smcs.sys_dispatch_lock_siva = SYS_DISPATCH_UNLOCK;
```

```
    SYS_EXIT_CRITICAL();
}
/*****************************中断级调度器**********************/
void sys_clockServe() interrupt 1
{

    TH0 = sys_clock_TH_cfg;
    TL0 = sys_clock_TL_cfg;

    if(smcs.sys_waitqueue_va > 0)
    {
        data8 i = 0;
        for(i=1; i<process_number_cfg; i++)
        {
            if(PRO_DELAY_STCO == pcs[i].pro_status_stva )
                {
                    pcs[i].pro_delay_time_va--;
                    if(0 == pcs[i].pro_delay_time_va)
                    {
                        pcs[i].pro_status_stva = PRO_RUN_STCO;
                        pcs[i].pro_prestatus_stva = PRO_DELAY_STCO;
                        smcs.sys_waitqueue_va--;

                        if(PRO_PRIRUN_STCO == pcs[i].pro_priority_stva)
                        {
                                Que_pri_inQueue (i);
                        }
                        else
                        {
                                Que_com_inQueue (i);
                        }
                    }
                }
            #if overtime_enable > 0
            else if((PRO_RUN_STCO == pcs[i].pro_status_stva)&&
                    (PRO_DELAY_STCO == pcs[i].pro_prestatus_stva)&&
                    (PRO_COMRUN_STCO == pcs[i].pro_priority_stva))
            {
                pcs[i].pro_overtime_va--;
                if(0 == pcs[i].pro_overtime_va )
                {
                    Que_com_rmQueue(i);
                    pcs[i].pro_overtime_va = pro_overtime_cfg;
                    Que_pri_inQueue (i);
```

```
                }
            }
            #endif
        }
    }


    if ((smcs.sys_runstatus_stva == SYS_RUN_STCO)&&
            (smcs.sys_runmode_mova == SYS_THREAD_MOCO)&&
            (smcs.sys_dispatch_lock_siva == SYS_DISPATCH_UNLOCK))
    {

        if(smcs.sys_currun_id==0)
            {
                smcs.sys_time_slice_va = 0;
            }

        if(smcs.sys_time_slice_va > 0)
            {
                smcs.sys_time_slice_va--;
            }
        else
            {
                    if((priority_queue.queue_length  >  0)||(commom_queue.queue_length  >
0))
                    {
                        if (smcs.sys_priorityrun_va > 0)
                        {
                                smcs.sys_nexrun_id = Que_pri_ouQueue();
                        }
                        else
                        {
                                smcs.sys_nexrun_id = Que_com_ouQueue();
                        }

                      if(PRO_RUN_STCO == pcs[smcs.sys_nexrun_id].pro_status_stva)
                        {
                            pcs[smcs.sys_currun_id].pro_SP = SP;
                                if(smcs.sys_currun_id != 0)
                                {
                                    if(PRO_PRIRUN_STCO                          ==
pcs[smcs.sys_currun_id].pro_priority_stva)
                                    {
                                        Que_pri_inQueue (smcs.sys_currun_id);
```

```
                        }
                    else
                        {
                            Que_com_inQueue (smcs.sys_currun_id);
                        }
                    }

                    smcs.sys_currun_id = smcs.sys_nexrun_id;
                    smcs.sys_nexrun_id = 0;
                    pcs[smcs.sys_currun_id].pro_prestatus_stva                =
PRO_RUN_STCO;
                    #if overtime_enable > 0
                    pcs[smcs.sys_currun_id].pro_overtime_va                   =
pro_overtime_cfg;
                    #endif
                    SP = pcs[smcs.sys_currun_id].pro_SP;
                }
            }
            smcs.sys_time_slice_va = SYS_TIME_SLICE_cfg;
        }
    }
}
```

FMOS_Init_Fi.c 系统初始化文件

```
/*************************************************************
*     author:fengming
*     date:2014-4
*************************************************************/
#include "FMOS_Macro_Define_Fi.h"
#include "FMOS_Configure_Fi.h"
#include "FMOS_Hand_Fi.h"
#include <reg52.h>

  struct sys_manage_control_structure idata smcs;/*instant systern control block*/
  struct pro_control_structure idata pcs[process_number_cfg];     /**/

  struct run_queue idata commom_queue;     /*instant commom run queue*/
  data8 idata pro_commom_queue[pro_comque_length_cfg];      /*define commom queue length*/

  struct run_queue idata priority_queue; /*instant prioryty run queue*/
  data8 idata pro_priority_queue[pro_prique_length_cfg];    /*define priority queue length*/

  data8 idata process_stack[process_number_cfg][pro_stack_deep_cfg];     /*process stack*/

  struct signal_control_structure idata sigcs[signal_number_cfg];
  data8 idata signal_wait_queue[signal_number_cfg];

void OS_init(void)
{
        Que_com_initialQueue ();
        Que_pri_initialQueue ();

        Pro_initialPCS();
        SYS_initialSMCS();
        SIG_initialSCS();
        SYS_initialOS();

}

  /*initial the system manage and control block*/
void SYS_initialSMCS(void)
{
        smcs.sys_runstatus_stva = SYS_STOP_STCO;
        smcs.sys_runmode_mova = SYS_SERVICE_MOCO;
        //smcs.sys_interr_ooc_siva = SYS_INTERR_CLOS;
        smcs.sys_priorityrun_va = 0;
        smcs.sys_waitqueue_va = 0;
```

```
    smcs.sys_dispatch_lock_siva = SYS_DISPATCH_LOCK;
    smcs.sys_time_slice_va = 0;
    smcs.sys_currun_id = 0;
    smcs.sys_nexrun_id = 0;
}

/*initial the process control struct */
void Pro_initialPCS(void)
{
    register data8 i = 0;
    for (i = 0;i<process_number_cfg;i++)
    {
        pcs[i].pro_SP = 0;
        pcs[i].pro_priority_stva = PRO_COMRUN_STCO;
        #if overtime_enable > 0
        pcs[i].pro_overtime_va = pro_overtime_cfg;
        #endif
        pcs[i].pro_delay_time_va = 0;
        pcs[i].pro_status_stva = PRO_STOP_STCO;
        pcs[i].pro_prestatus_stva = PRO_STOP_STCO;
    }

}
/*initial the run queue befor system run*/
void Que_com_initialQueue (void)
{
    register data8 i = 0;
    commom_queue.queue_length = 0;
    commom_queue.queue_front    = 0;
    commom_queue.queue_rear     = 0;

    for(i= 0;i<pro_comque_length_cfg;i++)
    {
        pro_commom_queue[i] = 0;
    }
    commom_queue.queue_data = &pro_commom_queue;
}
void Que_pri_initialQueue (void)
{
    register data8 i = 0;
    priority_queue.queue_length = 0;
    priority_queue.queue_front    = 0;
    priority_queue.queue_rear     = 0;
```

```
	for(i= 0;i<pro_prique_length_cfg;i++)
	{
		pro_priority_queue[i] = 0;
	}
	priority_queue.queue_data = &pro_priority_queue;
}
void SIG_initialSCS(void)
{
	data8 i = 0;
	for(i=0; i<signal_queue_length_cfg; i++)
	{
		signal_wait_queue[i]= 0;
	}

	for(i=0; i<signal_number_cfg; i++)
	{
		sigcs[i].sig_create_status = SIG_NOCREATE_STCO;
		sigcs[i].sig_type = SIG_TYPE_NONE_STCO;
		sigcs[i].sig_counter= 0;
		sigcs[i].sig_have_wait= 0;
		sigcs[i].sig_waitqueue=&signal_wait_queue[i];
	}
}

void SYS_initialOS(void)
{
		register data8 i = 0;
		//---initial process stack: put the function address to process stack.
		process_stack[0][1] = (data16)task0;
		process_stack[0][2] = (data16)task0 >> 8;

		process_stack[1][1] = (data16)task1;
		process_stack[1][2] = (data16)task1 >> 8;

		process_stack[2][1] = (data16)task2;
		process_stack[2][2] = (data16)task2 >> 8;
	#if process_number_cfg > 3
		process_stack[3][1] = (data16)task3;
		process_stack[3][2] = (data16)task3 >> 8;
	#endif
	#if process_number_cfg > 4
		process_stack[4][1] = (data16)task4;
		process_stack[4][2] = (data16)task4 >> 8;
	#endif
```

```
    #if process_number_cfg > 5
        process_stack[5][1] = (data16)task5;
        process_stack[5][2] = (data16)task5 >> 8;
    #endif

        for ( i=0; i<process_number_cfg; i++)
        {
            pcs[i].pro_SP = process_stack[i];
            pcs[i].pro_SP +=15;
            pcs[i].pro_priority_stva = PRO_COMRUN_STCO;
            #if overtime_enable > 0
            pcs[i].pro_overtime_va = pro_overtime_cfg;
            #endif
            pcs[i].pro_delay_time_va = 0;
            pcs[i].pro_status_stva = PRO_RUN_STCO;
            pcs[i].pro_prestatus_stva = PRO_STOP_STCO;
        }

        for(i=0; i<process_number_cfg; i++)
        {
            //pcs[i].pro_inWhereQue = PRO_IN_COMQUE_STCO;
            Que_com_inQueue(i);
        }
}
```

测试程序一共创建 5 个任务，任务一为空闲任务，任务二 ~ 四通过延迟等待获得所需要的时间间隔，这四个任务并发执行。