# 8 SanctOS – a Round-Robin RTOS

This chapter explains the very simple round-robin RTOS called SanctOS, where each task (or function) works for a specified amount of time before passing on the processor time to the next task.

This RTOS is a direct adaptation of the home-brew round-robin ParrOS (Paul Round-Robin Operating System) RTOS assembly language program described in great detail in the appendix. This is the improved version written in the C language so as to make it more versatile and more easily portable to other micro-controllers. The name SanctOS is an acronym for Small ANd CompacT Operating System, and before proceeding further, it would be very advantageous if the ParrOS program is understood by reading the appropriate appendix.

Most of the commands are exactly the same as those for the ParrOS RTOS, (with the additional OS_ prefix), and the settings regarding the number of tasks, tick time and stack size can be set in the PARAMETERS.H file shown below.

There are some very immediate advantages in using C to write the RTOS. Parameters can be easily changed from char to integer or long types and the routines would automatically reflect the changes when they are compiled. An example here would be the OS_CREATE_TASK(parameter list) command where in the A51 version, the slot time parameter was of type integer (0-65535). If we had to change the parameter to long in order to be able to accommodate longer wait periods, we would have had to re-write the routines so as to increment or decrement double words (32-bit) rather than words (16-bit). In the C version this could be done fairly easily simply by changing the type declarations.

Naturally there are some memory space and speed penalties to pay for this versatility. However the improvements more than outweigh the penalties, especially as far as student understanding of the RTOS is concerned. Here we now list the RTOS commands, this time for the C version. The full SanctOS RTOS source program listing can be found in Appendix C.

## 8.1 SanctOS System Commands

The following are the only RTOS system calls available :

- OS_INIT_RTOS (uchar iemask); // Initialises all RTOS variables
- OS_RTOS_GO (void); // Starts the RTOS
- OS_CREATE_TASK (uchar task_num, uint task_add, uint slot_time);
  // Creates a task, allocating a slot_time during which it can execute

These commands are more fully explained in section 8.2 and its sub-sections.

## 8.2 Variations from the A51 version

The C version of the RTOS provides some variations and additional commands from ParrOS, which can be implemented easily, after using the program for a while.

### 8.2.1 OS_INIT_RTOS (uchar iemask)

This system command is used only once in the main program and its function is to initialise all the RTOS variables. It sets up the RTOS timer (the so called tick timer, which generates the regular the critical interrupt which calls the Interrupt Service Routine that handles the slot time counter and task swapping) and enables the required interrupts according to the iemask parameter given within the command.

An example of the syntax used for this command is:

```
OS_INIT_RTOS(0x01);
```

This would initiatilise the RTOS, enabling the external interrupt 0 since the iemask contents correspond to the interrupts shown in Table 8-1. The timer to be used for the RTOS tick timer (say Timer 0) and its corresponding interrupt would be enabled automatically (irrespective of the iemask setting), depending on the TICK_TIMER value declared in the SanctOS_Param.h header file. It should be noted here that this tick timer interrupt is therefore used by the RTOS and cannot be used by the user program.

| Interrupt | | IE MASK | | Notes |
|---|---|---|---|---|
| No: | Name | Binary | Hex | |
| 0 | External Int 0 | 00000001 | 01 | |
| 1 | Timer Counter 0 | 00000010 | 02 | Default RTOS timer for 8051 |
| 2 | External Int 1 | 00000100 | 04 | |
| 3 | Timer Counter 1 | 00001000 | 08 | |
| 4 | Serial Port | 00010000 | 10 | |
| 5 | Timer 2 (8032 only) | 00100000 | 20 | Default RTOS timer for 8032 |

**Table 8-1** IEMASK Parameter (SanctOS)

### 8.2.2 OS_RTOS_GO(void)

This system command is also used only once in the main program, when the RTOS would be required to start supervising and scheduling the individual processes or tasks. It does not take any parameters.

An example of the syntax used for this command is:

```
OS_RTOS_GO();
```

This would start the RTOS ticking, at a reference time signal of TICKTIME milliseconds as set in the parameters header file SanctOS_Param.h. This TICKTIME in milliseconds would then become the basic reference unit for other system commands which use any timeout parameter, such as the OS_CREATE() function.

### 8.2.3 OS_CREATE_TASK (uchar task_num, uint task_add, uint slot_time)

This system command creates the tasks by setting the appropriate variables corresponding to the task number (usually starting from 0), the function name (which actually corresponds to the address location where the task function actually starts in the program code area and the slot-time required for this task.

An example of the syntax used for this command gives some further explanation of its function and purpose.

```
OS_CREATE_TASK(0,Task_Zero_Routine,25);
```

This would create a task which refers to the function or sub-routine Task_Zero_Routine, having a task number 0 to be handled by the RTOS. The slot-time given for this task is 25 ticktimes. The value of TICKTIME milliseconds, declared in the SanctOS_Param.h would be used as the basic reference unit for this slot-time. Thus if TICKTIME was declared as 1 (meaning one millisecond), then the above task would run for 25ms each time it is given the go-ahead to run, that is this task would run in bursts of 25ms duration, stopping after 25ms when the next task would run for its own specified slot time and so on until all the tasks would have run and the turn for Task_Zero_Routine comes up again.

### 8.2.4 Other add-on MACROS

These macros (#define statements) add some more basic commands and flexibility.

```
OS_PAUSE_RTOS()          // Disable the RTOS
OS_RESUME_RTOS()         // Re-enable the RTOS

OS_CPU_IDLE()            // Sets the microprocessor in idle mode
                         // This is usually used in the main program endless loop after
                         // initialising and starting the RTOS.
OS_CPU_DOWN()            // Sets the microprocessor in power-down mode
```

These #define statements are simply substitutions for some instructions which might seem meaningless if they are just written in the normal way. For example, using these 2 definitions

```
#define OS_CPU_IDLE()          PCON |= 0x01 // Sets the microprocessor in idle mode

#define OS_CPU_DOWN()          PCON |= 0x02 // Set microprocessor in power-down mode
```

It would make the program much easier to understand if we use

```
OS_CPU_IDLE();
```

rather than just writing

```
PCON |= 0x01;
```

This SanctOS operating system is very simple to use and is ideal for situations where we have totally independent tasks. That is we have various jobs to do which do not rely on any input or event from some other task. If our particular requirement stipulates that we need to do all jobs together rather than sequentially, than this RTOS can be our solution. The tasks would all appear to be running simultaneously although in fact they would be alternating and using the processor time for a few milliseconds each.

## 8.3 SanctOS example program

A simple example would help to explain how the OS works. The programs or modules required, apart from the main example program are:

SanctOS_Startup.a51, SanctOS_A01.A51, SanctOS_V01.c using the header files SanctOS_V01.h and SanctOS_param.h

This example program creates 255 tasks, which happen to be practically all the 'same' task just to make it simpler to program. Each task simply outputs the task number to port P1. Thus the first task would output a zero and the last task would output 254 to this port. The number 255 refers to the main() function. The variable *Running* is actually used in the SanctOS RTOS to refer to the task number of the currently executing task and is declared as an external variable so that it can be used in the example or application program too.

Since the tasks, when created, are allocated a slot time of 50 and the parameter TICKTIME is given a value of 1, then when the RTOS program is executing, it would first start task 0, thus P1 would be 0 for 50 ms.

Then task 1 would be invoked, and P1 would be 1 for another 50ms, then task 2 and so. Thus effectively, P1 would be counting and showing 0 to 254 in 50ms steps!! So after Task 0 executes, it would have to wait for the other tasks and the main() program(255 other routines) to each execute in turn for 50ms before it can continue executing again. Because of the large number of tasks in this particular example, this delay which works out to 12.75s might not be acceptable for the particular application. If on the other hand we have fewer tasks (say 10) and you allocate a slot time of say 2ms per tasks, then the waiting period for each task between successive executions would only be 20ms which might be more acceptable. This can easily be checked in the example program, simply by changing the NOOFTASKS parameter in the SanctOS_Param.h file and the slot-time in the OS_CREATE_TASK() command in the main program.

One can compare this round-robin RTOS with the Chinese juggler spinning plates on those long sticks in some circus. Each stick (task) is 'touched' in turn, making sure that each plate (task) is visited before it is too late. Depending on our application, we can determine the slot-time which we require for each task, remembering that only one task would actually be executing at any particular moment.

It should also be mentioned at this point, that most applications can be written without using any RTOS, making use instead of the various interrupt service routines. However, the use of an RTOS can most of the time help us to write a more user-friendly programme which is neater and simpler to maintain. Some time is needed to get familiar with the RTOS commands and the way it is initialised, but once this is mastered, it should be relatively simple to implement our project using the RTOS.

**Example01.c**

```
/*****************************************************************************/
/*                                                                           */
/*      Example01.c: Demo                                                    */
/*              SanctOS demo                                                 */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/

#include <reg52.h>     /* special function registers 8052                   */
#include <absacc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "SanctOS_V01.h"      /* SanctOS RTOS system calls definitions       */

/*****************************************************************************/
/*      Task X:                                                              */
/*****************************************************************************/
        // This variable 'Running is declared in SanctOS_V01.h as
        // 'extern data unsigned char Running; '
        // and contains the number of the currently running task
        // which can therefore be used in the main program if required
void OutPort(void){    /* Output Task Number on to Port P1                   */
   while(1)
        {
                P1 = Running;
        }
   }

/*****************************************************************************/

/*****************************************************************************/
/* Main: Initialise and CREATE tasks */
/*****************************************************************************/

void main (void)       {       /* program execution starts here             */
unsigned char i;
        OS_INIT_RTOS(0x20);    /* initialise RTOS variables and stack       */
                               /* using Timer 2 interrupts                  */
                               /* OS_INIT_RTOS(0x00); would also be correct  */
                               /* since the tick-timer interrupt            */
                               /* will be set automatically                 */

        for(i=0;i<NOOFTASKS;i++) OS_CREATE_TASK(i, OutPort, 50);

        P1 = 0;
        OS_RTOS_GO();          /* start SanctOS RTOS                        */

        while (1) OS_CPU_IDLE();
}

/*****************************************************************************/
```

```
SanctOS_Param.H
/*
********************************************************************************
*                      SanctOS_Param.H --- RTOS KERNEL HEADER FILE
*
* For use with SanctOS_V01.C - Round-robin RTOS written in C by Ing. Paul P. Debono
*                      for use with the 8051 family of microcontrollers
*
* File          : Parameters_V01.H
* Revision      : 8
* Date          : February 2006
* By            : Paul P. Debono
*
*                       B. Eng. (Hons.) Elec. Course
*                       University Of Malta
*
********************************************************************************
*/
#ifndef __SANCTOS_PARAM_H__
#define __SANCTOS_PARAM_H__

/*
********************************************************************************
* RTOS USER DEFINITIONS
********************************************************************************
*/
#define STACKSIZE       0x10    // size of stack for each task - no need to change
#define CPU             8032    // set to 8051 or 8032
#define TICK_TIMER      2       // Set to 0, 1 or 2 to select which timer to use as the RTOS tick
timer
#define TICKTIME        1       // Length of RTOS basic tick in ms
#define NOOFTASKS       255     // Number of tasks used in the application

/*
********************************************************************************
*/

#endif
```

**Example 2**

The second example shows a 3-task application, each task having a 5ms time-slot. Each task toggles a different pin on port P1 every 1s, 1.5s and 3s respectively and these timings are worked out by a Timer 0 interrupt service routine, running independently from the RTOS interrupt. Timer 0 is set up to overflow every 50ms and a simple counter can be used to determine the number of overflows so as to get the correct pin-toggling timings. The RTOS this time uses Timer 2 as the tick-timer source, which is used to give each task a 5ms time-slot in which to run.

When executing, this program really gives the impression that all the three tasks are running simultaneously. The timing of the LEDs is a bit approximate here since the task might not be actually running when the LED toggle time expires. In the worst case scenario, the task might have to wait up to 10 milliseconds for the other two tasks to use their slot time before it would notice (when its time-slot comes up) that the toggling time has passed.

**Example02.c**

```
/*****************************************************************************/
/*                                                                           */
/* Example02.c: Demo                                                         */
/* SanctOS demo                                                              */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/

#include <reg52.h> /* special function registers 8052                        */
#include <absacc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "SanctOS_V01.h" /* SanctOS RTOS system calls definitions             */

sbit Led0 = P1^0;
sbit Led1 = P1^1;
sbit Led2 = P1^2;

bit Sec, OneSecFive, ThreeSecFive;

/* set up timer 0 as a 16-bit timer                                          */
/* Overflows every 50 milliseconds with 11.0592 MHz clock                    */
/* Timer needs to count 46080 before it overflows                            */
void SetUp_Timer0 (void)
{
    TMOD &= 0xF0;             // clear timer 0 control bits only
    TMOD |= 0x01;             // 16-bit
    TH0 = (65536-46080)/256;
    TL0 = (65536-46080)%256;  // Overflows every 50 milli seconds
    TR0 = 1;                  // Timer ON
    ET0 = 1;                  // Enable TF0 interrupt
}

void TF0_ISR (void) interrupt 1 using 2
{
    static unsigned int data overflow_count;
    TH0 = (65536-46080)/256;
    TL0 = (65536-46080)%256;  // Overflows every 50 milli seconds
    overflow_count = (overflow_count + 1) % 420;             // 420 is the LCDM of 20,30 and 70
    if (overflow_count%20UL == 0) Sec = 1;                   // 20 overflows = 1s
    if (overflow_count%30UL == 0) OneSecFive = 1;            // 30 overflows = 1.5s
    if (overflow_count%70UL == 0) ThreeSecFive = 1;          // 70 overflows = 3.5s
}

/*****************************************************************************/
/* Task 0: */
/*****************************************************************************/
```

```
void ToggleLed0(void){        /* Toggle LED 0                                  */
               while(1)
                       {
                       Led0 = ~Led0;
                       while(Sec == 0);
                       Sec = 0;
               }
 }

/*****************************************************************************/

/*****************************************************************************/
/*      Task 1:                                                              */
/*****************************************************************************/

void ToggleLed1(void){        /* Toggle LED 1                                  */
               while(1)
                       {
                       Led1 = ~Led1;
                       while(OneSecFive == 0);
                       OneSecFive = 0;
               }
 }

/*****************************************************************************/

/*****************************************************************************/
/* Task 2: */
/*****************************************************************************/

void ToggleLed2(void){        /* Toggle LED 2                                  */
               while(1)
                       {
                       Led2 = ~Led2;
                       while(ThreeSecFive == 0);
                       ThreeSecFive = 0;
               }
 }

/*****************************************************************************/

/*****************************************************************************/
/* Main: Initialise and CREATE tasks */
/*****************************************************************************/

void main (void)       {      /* program execution starts here                */
       OS INIT_RTOS(0x22);    /* initialise RTOS variables and stack          */
                              /* using Timer 0 & Timer 2 interrupts */
       OS CREATE_TASK(0, ToggleLed0, 5);
       OS CREATE_TASK(1, ToggleLed1, 5);
       OS CREATE_TASK(2, ToggleLed2, 5);
       P1 = 0;
       SetUp_Timer0(); /* Timer 0 interrupts are once again enabled here !!   */
       OS_RTOS_GO();          /* start SanctOS RTOS                           */
       while (1) OS_CPU_IDLE();
}

/*****************************************************************************/
```