SiRF Application Note Implementing User Tasks on the SiRFstarIII

Document Number **APNT3007**
Revision 1.0
1/20/05

Author: Young Lee

Table of Contents

LIST OF TABLES

1. INTRODUCTION

The purpose of this Application Note is to describe methods of implementing user tasks on the SiRFstarIII GSP3f and GRF3w chipsets using the embedded ARM7TDMI processor. User tasks implementation is limited to GSW3 software. It is not available to SLC3 users.

1.1. Background

The SiRFstarIII GSP3f ASIC integrates the GPS core, ARM7TDMI core, 4 Mbit flash, and 96 KB of SRAM. The SiRFstarIII GPS core performs satellite acquisition, measurement, tracking, and navigation. Running at 50 MHz, 20 - 25 % of CPU is being used under normal condition, which leaves enough CPU power for other applications. During the startup period, about 75 % of CPU is needed to acquire satellite signal.

The GSW3 software architecture is structured to allow the system designers to incorporate functionalities in the form of user tasks, thereby achieving a cost-effective solution by utilizing existing components.

2. APPLICABLE DOCUMENTS

1050-0053 GSW3 Software System Development Kit Reference Manual

3. USER TASKS OVERVIEW

In GSW3, there are three ways that a customer can implement specific user functions: two in the form of tasks and one by utilizing a callback function off the timer interrupt.

There are two user tasks available that customers can utilize: One is a periodic user task that gets scheduled off the 100 ms timer. 100 ms is the minimum interval that this user task can run. A different interval can be accommodated as long as it is in the multiple of 100 ms. This user task is suitable for the time-critical applications that demand predictable execution. The other user task is a non-periodic one that a user can schedule whenever it is desired. Since it has the lowest priority, it is appropriate for the task that requires a long execution time but not in a timely manner. The priorities of the user tasks are fixed and cannot be changed. Additionally, a callback function based off the 1 ms timer is available for very short and time-critical tasks.

SiRFstarIII employs different power saving methods for power-sensitive applications. See APNT3008, SiRF Application Note Power Saving Modes, for detailed information. While performing tracking and navigation, a receiver goes through power cycles where power is turned off and back on.

This affects how user tasks run: as this application note is written, user tasks are not available when any one of power saving methods is employed. Later software revision may alter this property.

3.1. Introduction to SiRF Tasks

The GSW3 software has a simple executive running a set of tasks with fixed priorities as shown in Table 1. The tasks are arranged according to their priority, from high priority at the top to low at the bottom.

SiRF tasks are preemptive. When a higher priority task is ready to run, it will preempt the current task and start to run. When that task is finished, the suspended task will resume.

Table 1 Basic SiRF Tasks

Tasks	Description	Comments
ATX Task	Determines and executes search strategies. Acquires, tracks and verifies satellites	Must run every 100 ms
RxC Task	Performs satellite prepositioning, visible list, state updates, navigation, power control, and message output (except some debug).	Runs 9 times a second (every 100 ms but one time it runs a navigation task which takes longer than 100 ms)
Serial Task	Handles serial input data	Runs whenever input data must be serviced.
Background Task	Computes satellite state and visible list once receiver has navigated	Runs every second.

3.2. Periodic User Task

The periodic user task is scheduled at a multiple of 100 ms, set during compilation. It has a high priority, just below the ATX task. Considering that the ATX task is critical to the GPS performance, the periodic user task should not interfere with the ATX task. Blocking interrupts for a long time must be avoided.

3.3. Non-periodic User Task

A user can call a function, MI_ScheduleUserTask (void), to schedule the non-periodic user task. This function call will add the non-periodic task to the pending task list. Additional calls to the function will have no effect until the task is actually set to run by the scheduler. At that time the task maybe again added to the pending list so that it will run again once the first request is complete. It has the lowest priority, below the background task. It is important for this task not to block other tasks from running.

3.4. User Callback Function

This function is called from the timer interrupt. The interval is limited to either 1 ms, 20 ms or 100 ms. Setting it to some other value will cause it to default of 100 ms. Its execution should be very short and should not block interrupts.

4. IMPLEMENTING USER TASK

User tasks are not enabled by default. In this section, how to enable user tasks and incorporate the functionalities are described. Sample code is provided. All the relevant code for user tasks resides in MiTasking.c which can be located in the MITasking folder under SDK.

4.1. Enabling User Tasks

1. User periodic and non-periodic tasks

   - File MiTasking.c, function MI_TaskOpen(): uncomment the call to MI_RegisterUserTask (). This registers both tasks.

   - Define the period for the periodic task. In the project file, Thumb C Compiler, Preprocessor tab, modify TASK_PERIOD=0 to the desired interval. Units are ms, and valid values are multiples of 100. for example, to have the periodic task run every 200 ms use the following:

     ```
     TASK_PERIOD=200
     ```

- Add preprocessor define USER_SDK
- Add a call to schedule the non-periodic task where desired. MI_ScheduleUserTask ();
- In the file MiTasking.c, fill in desired code in the stubs provided, MITask_UserTaskHigh100msPeriodic () and MITask_UserTaskLow () for periodic and non-periodic user task, respectively.

2. User callback function

- File MiTasking.c, function MI_TaskOpen(), define desired interval. Units are ms, valid values are 1, 20 and 100 ms. Any other value will cause the interval to be set to 100 ms.
- Add code to the stub, MITask_UserInterrupt (), in MiTasking.c. No interrupt blocking, make execution very short.

miTasking.c

```
EXTR MITask_Open (void)
{
     if(mod_getModuleStat(MOD_ID_MITASK) == MOD_OPEN)
        return MOD_ALREADY_OPEN;
     else
     {
         /*
         MI_RegisterUserTask this function registers the user tasks with the
         scheduler it does the registration of both the periodic and the
         non-periodic task. If this function call is disabled then the user
         tasks are not executed. We have the user tasks independent of the
         interrupt based functions. One could still execute the interrupt
         based functions without the user tasks.
         Commenting out the line below disables the user tasks.
         */
//         MI_RegisterUserTask ();
         /*
         The following code sets the interval for the interrupts to be
         generated. It is not essential to have this set to a particular
         value.
         It is important to note that if using the 1ms timer, one needs
         to take care of all the other interrupt based tasks depending
         upon this alarm. Care needs to be taken as regards the code
         execution in high speed interrupt functions.
         The value of Interval need to be either 1 or 20. if it is neither
         then the default 100ms is used.
         the second field is used to indicate whether the interrupt is to be
```

```
        enabled or disabled. TRUE = enable; FALSE = disable
        */
        interval = 20;
        MI_SetUserInterruptInterval (interval, USER_TASK_ENABLE);
        miTaskOpenFlag = TRUE;
        mod_setModuleStat(MOD_ID_MITASK, MOD_OPEN);
    }
   return (MOD_SUCCESS);
```

The next step is to add user task code:

In a file, miTasking.c, there exist three function stubs, and, for periodic user task, non-periodic user task, and user callback function, respectively.

4.2. Sample User Code

Sample code is supplied here.

miTasking.c

```
/*****************************************************************************
 *  FUNCTION NAME: miTask_100ms
 *
 *  Input: None
 *
 *  Output: The user task is executed every 100ms
 *
 *  Description: This function is a stub function for the 100ms task.
 *               All the function calls in this routine are executed
 *               every 100ms. Also one thing to keep in mind is that the
 *               priority of this task is greater than the serial task but
 *               less than the background task. DO NOT CHANGE THE NAME OF
THE
 *               FUNCTION

 *

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

void MITask_UserTaskHigh100msPeriodic (void)
{
   static int user100msCnt = 0;
   {
      // debug printf every 1 second.
      if ((user100msCnt++ % 10) == 0)
      {
         PRINTF("#!User Task 100ms %ld", user100msCnt);
      }

      // schedule non-periodic user task every 5 second.
```

```
      if ((user100msCnt % 50) == 0)
      {
         MI_ScheduleUserTask ();
      }
   }
}

/*****************************************************************************
 *  FUNCTION NAME: MITask_UserTaskLow
 *
 *  Input: None
 *
 *  Output: The user task is executed every 1Hz
 *
 *  Description: This function is a stub fucntion for the low priority non
 *               periodic user task. DO NOT CHANGE THE NAME OF THE FUNCTION
 *
*****************************************************************************/

void MITask_UserTaskLow (void)
{
   PRINTF("#!User Task Low");
}

/*****************************************************************************
 *  FUNCTION NAME: MITask_UserInterrupt
 *
 *  Input: None
 *
 *  Output: The user task is executed every timed interrupt 1ms or 20ms
 *
 *  Description: This function is a stub fucntion for the interrupt based
 *               tasks. DO NOT CHANGE NAME OF THE FUNCTION
 *
*****************************************************************************/

void MITask_UserInterrupt (void)
{
   static int userIntssCnt = 0;

   // print every 5 seconds.
   if ((userIntssCnt++ % 50) == 0)
      PRINTF("#!User Task Ints %ld", userIntssCnt);
}
```

4.3. Differences with Previous Products

For those customers who are already familiar with a user task in SiRFstarII products, differences are:

- More flexible user tasks implementation is achievable: while GSW2 allows one user task and XTrac 2.0 two user tasks, GSW3 offers the same number of user tasks as XTrac 2.0 but an additional user interrupt capability.
- User tasks can be enabled by adding preprocessor define, USER_SDK and changing the period in TASK_PERIOD.
- Running at 50 MHz, GSW3 can furnish more MIPs to user's functions. However, care should be taken so that it does not disrupt critical GPS tasks.
- The version at this time of writing does not allow user tasks to run during power management operation.

5. DOCUMENT MAINTENANCE

5.1. Required Approval for Changes

Changes to this document require the approval of Application Engineering, Program Manager, and Quality.

5.2. Revision History

Rev	Rev Date	CN Number	Description	Author/Editor
1	1/20/05	2663	Initial Release under CN Control	Young Lee