

**tinyRTX User Manual**

**(MSP430)**

***tiny******R****eal-****T****ime e****X****ecutive*

*Microchip MSP430Family*

*MSP430G2553*

*Including Demo Application*

*For TI MSP-EXP430G2 LaunchPad*

Table of Contents

Section Page

1. Overview 3

1.1. tinyRTX Features – System Space 3

1.2. User App Features – User Space 4

2. tinyRTX Theory of Operation 4

2.1. tinyRTX Tasks 5

2.2. SRTX – System Real-Time eXecutive 5

2.3. SISD – System Interrupt Service Director 6

2.4. SUSR – System User Application Interface 6

2.5. STRC – System Trace Facility 6

3. Demo APplication 6

3.1. SUSR – System User Application Interface 6

3.2. UAPP – User Application 7

3.3. UADC – User Analog/Digital Conversion 7

**Revision History:**

12 Sep 14 SHiggins@tinyRTX.com Release 1.0.0 – Converted from PIC18 document.

# Overview

The tiny Real-Time eXecutive, or tinyRTX, is a small (therefore fast) flexible real-time kernal for embedded processors. tinyRTX is a periodic non-preemptive executive supporting multiple prioritized user tasks, and encourages minimizing the amount of processing performed in interrupt service routines. This design results in more predictable system operation, fewer disrupted control sequences, less resource contention and deadlock, and less incoherent shared data.

Future development on tinyRTX will implement binary and counting semaphores, which can be used to support mutual exclusion and synchronization. (The exisiting demo user application uses simple techniques for inter-task communication which do not require semaphores.)

tinyRTX does not natively support task preemption, reentrancy, dynamic task priorities, dynamic task creation, priority inversions, message mailboxes or queues, or dynamic RAM allocation. These or other features may be added in the future provided they can be easily omitted by the user if desired, to retain the existing tinyRTX advantages of simplicity, speed, and small code and RAM footprints.

This version of tinyRTX is for the PIC18F452 family of Texas Instruments Ultra Low Power controllers. It is written in C. It provides a non-preemptive multi-tasking real-time kernel and some common device drivers. Used as the starting point of the embedded software for a TI MSP430 design, it greatly reduces software development time, while providing a robust, tested, measurable operating framework.

## tinyRTX Features – System Space

tinyRTX and user application routines have been separated into modules based on function. These modules each conceptually reside in independent subsystems of “system space” and “user space.” This technique makes the code easy to learn, navigate, and maintain. Regular naming conventions have been followed.

System space modules begin with the letter “S”, and user space modules begin with the letter “U”. In general, most developers should seldom need to modify “S” modules.

**SRTX – System Real-Time eXecutive**

* Contains Scheduler and the Dispatcher
  + Scheduler runs by receiving timer interrupts and moving tasks to “ready” state
  + Dispatcher detects the ready state of the task and invokes it.

**SISD – System Interrupt Service Director**

* Contains Reset and Interrupt handlers

**SUSR – System User Application Interface**

* Primary interface between SRTX and user tasks
  + Two stage Power-On Reset for user applications
  + Timebase task for the task timer interrupt
  + Tasks for A/D conversion and I2C communication
  + General user tasks 1 through 3 scheduled by elapsed time

**STRC – System Trace Facility**

* Real-time tracing and throughput analysis
  + Macros provided invoke the Trace function
  + Macros store unique identifier in trace buffer
  + Configurable to also store timing information in trace buffer
    - Allows throughput calculations
  + Designed to be easily removed for production software

## User App Features – User Space

The included demo application is written completely in “user space.” Features include:

**UADC – User Analog-Digital Conversion App**

* Initializes and triggers the Microchip A/D Converter, which is wired to the potentiometer
* Converts the raw A/D value to 0 – 5.0V, convert to BCD and then to ASCII

# tinyRTX Theory of Operation

The tiny Real-Time eXecutive embodies a design philosphy of embedded software. The challenge is always to create software which executes quickly, but still can be changed relatively easily as requirements change. On the one hand, a full-featured off-the-shelf real-time operating system (RTOS) may be robust and flexible at the expense of high overhead (and licensing fees). So instead, many embedded designs use a “hand-rolled” custom “executive”, largely interrupt-driven with an often unused (or underused) background loop. This executive is built from scratch for each project to meet a set of static requirements, but is hard to understand, maintain, and change for updated requirements. Additionally, an interrupt-driven executive can be difficult to test and debug, because of the many interactions possible due to interrupts. These unexpected interactions can disrupt desired control sequences, cause deadlock due to resource contention, or mismanage shared data leading to use of data which is not coherent to a single point in time.

tinyRTX sets up a minimalist framework of how software modules cooperate, encourages prioritized user tasks run on a periodic basis, and suggests how functions should transfer control and data to each other during execution.

tinyRTX is a periodic non-preemptive executive supporting multiple prioritized user tasks, and encourages minimizing processing performed in interrupt service routines (ISR’s). Minimizing the usage of ISR’s reduces the interaction of non-synchronized tasks, reducing the possibilities of resource contention and task context and data disruption. Instead, interrupts are largely used to schedule periodic tasks, or indicate to periodic tasks that a hardware event has occurred. In well-contained problem spaces where the amount of processing and inter-task communication is minimal (such as I2C handshaking) ISR’s are encouraged as effective and fast.

## tinyRTX Tasks

tinyRTX provides by default three tasks, though of course this can be reduced or increased by the individual user. Task 1 has the highest priority; Task 3 has the lowest priority. In effect, this means Task 3 will not run if Task 1 (and/or Task 2) is scheduled to run. Instead, Task 3 will have to wait until both Task 1 (and/or Task 2) complete.

tinyRTX for the MSP430 is non-preemptive. This was done so the program is extremely similar to tinyRTX for PIC16 and PIC18, and also by not introducing complexity, the program is easier to understand. However, the features of the MSP430 appear sufficient for a pre-emptive executive, and that version may be implemented if there is enough interest.

Being non-premptive means that once Task 3 starts, should a timer interrupt occur and cause Task 1 (and/or Task 2) to be scheduled, after the timer interrupt Task 3 will resume and run to completion. After Task 3 completes, the highest priority task will be invoked by the Dispatcher.

tinyRTX places no restrictions on when tasks start relative to other tasks, or how often a task runs. These design parameters are set by the user.

## SRTX – System Real-Time eXecutive

The SRTX module contains the Scheduler and the Dispatcher. (See Figure 1.) The Scheduler runs by receiving timer interrupts at regular intervals specified by the user. After the user-specified number of timer interrupts occurs, the Scheduler communicates to the Dispatcher that a task is ready to run. The Dispatcher, which is the background idle loop, detects the ready state of the task and invokes it. If more than one task is ready to run at the same time, the Dispatcher invokes the highest priority task first.

## SISD – System Interrupt Service Director

The SISD module contains the Reset and Interrupt handlers. (See Figure 1.) The Interrupt Director saves the current processor context, determines the source of the interrupt, and can be configured to immediately execute a user interrupt routine. The user interrupt routine may optionally schedule a user task (along with servicing the interrupt.) The Interrupt Director may also be used to simply schedule a user task to later handle the event signaled by the interrupt. In these cases, after the Interrupt Director completes, and after the task that was interrupted also subsequently completes, the task associated with the interrupt will be invoked by the Dispatcher.

## SUSR – System User Application Interface

The SUSR module is the primary interface between SRTX and user tasks. (See Figure 1.) It provides a single location through which all user tasks are invoked. Tasks include user application power-on reset initializations for both early and late stages, the Timebase task for the Timebase timer interrupt, tasks for A/D conversion and I2C communication, and general user tasks 1 through 3 scheduled by elapsed time.

## STRC – System Trace Facility

The STRC module provides real-time tracing and throughput analysis of embedded program execution by storing trace information in a trace buffer. Macros invoking the Trace function are embedded in the user code or in the SUSR application interface, before or after critical program operations. Each macro causes a unique identifier to be stored in the trace buffer. When the trace buffer is full, the user can review the order of program execution by reviewing the stored trace information. The STRC module is designed to be easily removed for production software.

# Demo APplication

A demo application has been included to illustrate using the tinyRTX package. The demo application requires tinyRTX to run. Notice how the various application functions have been separated into modules based on function for improved maintanance.

## SUSR – System User Application Interface

The SUSR module is also listed above in SRTX Module Descriptions.

## UAPP – User Application

UAPP contains the main User Application. It contains routines for application main initialization, and timer initialization to generate interrupts for the SRTX Scheduler. For the demo timebase, Timer1 is used to generate interrupts every 100ms.

## UADC – User Analog/Digital Conversion

UADC contains the user application code to initialize and trigger the Microchip A/D Converter, which on the PICDEM 2 Plus board is wired to the potentiometer. UADC also contains a routine to convert the raw A/D value to 0 – 5.0V. The first step is to rescale the A/D input from 0-0x3FF to engineering units of 1 mV. The second step is to convert from mV in hex to mV in Binary Coded Decimal (BCD). The third step is to convert BCD to ASCII. Finally, the ASCII result is stored in the user application LCD buffer.

A/D conversion is such a straightforward process that no efficiency may be gained by implementing any part of it in SRTX. Most of the work lies in converting the raw A/D result, and the math routines used are included with the rest of the demo application.

Fig01 SRTX.wmf