

**tinyRTX User Manual**

**(PIC18 with C core)**

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

*Microchip PIC18F452 Family*

*PIC18F242 and /42A*

*PIC18F252 and /52A*

*PIC18F442 and /42A*

*PIC18F452 and /52A*

*Including Demo Application*

*For Microchip PICDEM 2 Plus Demo Board*

Table of Contents

Section Page

1. Overview 4

1.1. tinyRTX Features – System Space 4

1.2. User App Features – User Space 5

2. tinyRTX Theory of Operation 7

2.1. tinyRTX Tasks 7

2.2. SRTX – System Real-Time eXecutive 8

2.3. SISD – System Interrupt Service Director 8

2.4. SUSR – System User Application Interface 8

2.5. SLCD – System LCD Device Driver 8

2.6. SI2C – System I2C Device Driver 9

2.7. STRC – System Trace Facility 9

3. Demo APplication 9

3.1. SUSR – System User Application Interface 9

3.2. UAPP – User Application 9

3.3. ULCD – User LCD Formatting 10

3.4. UADC – User Analog/Digital Conversion 10

3.5. UI2C – User I2C Communication 10

3.6. MA16, E2BCD16, and FXM1616U – Math Routines 11

3.7. tinyRTX-PIC18F452.LKR – Linker Script 11

4. How to Assemble and run the Demo APplication 11

4.1. Required Hardware 11

4.2. Required Software 11

4.3. Unpacking and Assembling the Software 12

4.4. Running the Software 12

5. Software Diagrams and Call Trees 12

5.1. Figure 1 –Interrupts, Executive, and User Interface 12

5.2. Figure 2 –User and System LCD Handling 12

5.3. Figure 3 – User and System I2C Handling 12

5.4. Figure 4 – User I2C Implementation 12

5.5. Figure 5 – SISD\_Reset and SISD\_Interrupt 13

5.6. Figure 6 – SRTX\_Dispatcher 13

5.7. Figure 7 – SUSR\_Task3, SUSR\_TaskI2C and SUSR\_ISR\_I2C 13

**Revision History:**

(17 Aug 14 SHiggins@tinyRTX.com Release 1.0.0 – Converted from PIC16 document.)

(1 Sep 14 SHiggins@tinyRTX.com Release 1.1.0 – Updated for moving final I2C processing into scheduled task.)

3 Sep 14 SHiggins@tinyRTX.com Release 1.1.0 – First release with C core, 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 Microchip PIC controllers. It is written in PIC18 assembly language with the exception of the core executive routines

SRTX – System Real-Time eXecutive, and  
SISD – System Interrupt Service Director

which are 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 Microchip PIC 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. The use of separate modules requires a .LKR file with detailed comments, which has been included.

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

**SLCD – System LCD Device Driver**

* Independent refresh of upper/lower lines of a standard 2x16 character LCD display
  + ASCII characters to two 16-byte RAM buffers
  + Refresh rates controlled by user

**SI2C – System I2C Device Driver**

* Implements I2C serial communication using a Microchip MSSP module
  + Each SI2C routine handles a simple MSSP function
  + Routines linked together using a state table
    - Implemented with a computed GOTO jump table
  + Routines may be configured to implement different I2C protocols
    - Write Data, Read Status Then Write Data, etc.
  + Reads and writes may be single or multiple bytes for any message
  + Slave ACK/NACK detection and error handling is supported

**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:

**ULCD – User Liquid Crystal Display App**

* Independent formatting of the upper and lower lines of the LCD display
  + Upper line contains a scrolling display with contact information
  + Lower line contains the 0-5.0 V voltage present at the potentiometer
  + Lower line also contains current digital temperature reading

**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

**UI2C – User Inter-Integrated Circuit App**

* Initializes the I2C module to communicate with the temperature probe
* Full I2C implementation using SI2C table-driven state machine, including “Msg Done”

# 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 PIC18 is non-preemptive. This was done so the program is extremely similar to tinyRTX for PIC16, and also by not introducing complexity, the program is easier to understand. However, the features of the PIC18 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.

## SLCD – System LCD Device Driver

The SLCD module provides independent refresh of the upper and lower lines of a standard 2x16 character LCD display. (See Figure 2.) The user simply writes desired ASCII characters to two 16-byte RAM buffers when convenient to the user application. The user application then invokes the SLCD routines to refresh each LCD line. Lines may be refreshed independently and at different rates.

## SI2C – System I2C Device Driver

The SI2C module provides a family of routines implementing I2C serial communication using a Microchip MSSP module. (See Figure 3.) Each SI2C routine handles a simple MSSP function, such as performing an I2C Bus Start Enable, or reading received data from the SSPBUF register. These simple routines are linked together using a state table implemented with a computed GOTO jump table. (See Figure 4.) Different routines may be mixed and matched to implement different I2C protocols, such as Write Data, or Read Status Then Write Data. Reads and writes may be single or multiple bytes for any message. Slave ACK/NACK detection and error handling is supported. Individual routines representing message states are invoked either by the user, or by MSSP-provided interrupts.

## 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.

## ULCD – User LCD Formatting

The ULCD module provides independent formatting of the upper and lower lines of the PICDEM 2 Plus 2x16 character LCD display. The upper line contains a scrolling display with contact information for Sycamore Software, Inc. The scrolling position is updated once each second, which updates the data in the ULCD Line1 display buffer. The SLCD routine to update the hardware LCD Line1 is also called once each second.

The lower line contains the 0-5.0 V voltage present at the potentiometer, and the temperature reading present at the TC74 digital temperature sensor. The ASCII value for the voltage is updated every 100ms, and the ASCII value for the temperature is updated every 7 seconds. Therefore, the ULCD Line1 display buffer is updated every 100ms. The SLCD routine to update the hardware LCD Line1 is also called once each second.

## 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.

## UI2C – User I2C Communication

UI2C contains the user application code to initialize the MSSP to communicate with the TC74 digital temperature probe on the PICDEM 2 Plus demo board. Also all software is included to fully use the SI2C routines to implement complex I2C messages which both read and write.

Like the SI2C routines, the UI2C code is a collection of simple routines which represent various states. The UI2C states represent 1) Send Status Request I2C msg to TC74 and get Status result, 2) Check TC74 Status result, and either resend Status Request because TC74 not ready, or proceed to next state, 3) Send Data Request I2C msg to TC74 and get Data result, and 4) Schedule I2C User Task to process the message data from the TC74. The SRTX Dispatcher will invoke the UI2C Task to process the message data when the task is scheduled.

The simple routines for the above states are linked together using a state table implemented with a computed GOTO jump table. Also notice the routine named UI2C\_MsgDone. This special routine is called from the SI2C services (through SUSR, of course) when the desired I2C message completes (ie, an MSSP I2C Bus Stop has completed.) This routine then triggers the next state routine in UI2C.

## MA16, E2BCD16, and FXM1616U – Math Routines

These modules contain math routines to support the A/D conversion for the potentiometer, and the I2C temperature data received from the TC74 thermometer.

## tinyRTX-PIC18F452-c.lkr – Linker Script

When an entire embedded software package is wrapped into a single source file, it becomes unwieldy and hard to maintain. Both tinyRTX code and user application code have been created in separate files to make the code easy to learn, navigate, and maintain. Therefore a linker script has been used to ensure proper code placement. This file includes extensive comments.

# How to Assemble and run the Demo APplication

Unzip the file into a new directory.

## Required Hardware

You will need a PICDEM 2 Plus demo board from Microchip. You will also need a Microchip ICD 2 (or later) in-circuit debugger connected to a PC, with the ICD 2 software drivers installed. A similar debugger or emulator may be used in place of the ICD 2.

## Required Software

You will need the Microchip MPLAB IDE v6.30 or later properly installed on your PC. The latest version tested is MPLAB IDE 8.92, available from www.microchip.com. This software allows you to assemble and link the demo software. It also allows you to control the ICD 2 (or whichever debugging device you are using).

You will also need the embedded software package which accompanies this document. If you received this document without the embedded software package, please contact Sycamore Software, Inc. at SHiggins@tinyRTX.com

## Unpacking and Assembling the Software

First create a new subdirectory. Copy the archive into the new directory. Then unzip the archive using WinZip or a similar program in the same directory.

Open the MPLAB IDE. Use FILE/OPEN WORKSPACE and select “tinyRTX PIC16F877.mcw” as the workspace. Use PROJECT/BUILD ALL to assemble and link the assembly language source files.

## Running the Software

Select the ICD 2 by using DEBUGGER/SELECT TOOL/MPLAB ICD 2. The OUTPUT window should confirm that the MPLAB ICD 2 is READY. (If the OUTPUT window does not confirm, please consult the ICD 2 documentation, and ensure you can use the ICD 2 with the Microchip-supplied PICDEM 2 Plus software.) Program the demo board by selecting DEBUGGER/PROGRAM. Run this application by using DEBUGGER/RUN.

# Software Diagrams and Call Trees

The Microchip PIC 16F family supports 8 stack levels. Using more than 8 levels of interrupts and/or subroutine calls will exhaust the stack and cause unreliable or failed program operation. It is important that the finished embedded software have a full call tree analysis to determine maximum stack depth required.

## Figure 1 –Interrupts, Executive, and User Interface

TBD.

## Figure 2 –User and System LCD Handling

TBD.

## Figure 3 – User and System I2C Handling

TBD.

## Figure 4 – User I2C Implementation

TBD.

## Figure 5 – SISD\_Reset and SISD\_Interrupt

Light grey boxes indicate an initial invocation of the tree, or the terminal leaf of the tree which does not call any additional routines. Light grey boxes with greyed text indicate optional task invocation, either by interrupt or or scheduled task. Dashed lines indicate program controlled is transferred by a GOTO statement, which will not require any stack usage.

## Figure 6 – SRTX\_Dispatcher

Light grey boxes indicate the terminal leaf of the tree which does not call any additional routines.

## Figure 7 – SUSR\_Task3, SUSR\_TaskI2C and SUSR\_ISR\_I2C

Dashed lines indicate program controlled is transferred by a GOTO statement, which will not require any stack usage. SUSR\_Task3 performs all its functions by using GOTO statements, as the UI2C implementation is based on states. The final routine in the tree SI2C\_StartEnable, an SI2C state implementation, does perform a RETURN which returns to SUSR\_Task3.

SUSR\_ISR\_I2C is a state-based implementation of MSSP control of the I2C bus. Many of the states are simple routines which perform only a single function. For example, SI2C\_StartEnable has the primay function of asserting the MSSP start enable bit. Likewise, SI2C\_SendWriteAddr places an I2C slave address in the SSPBUF. Most of the SI2C state implementations are placed at the top of Figure 7.

Notice state SI2C\_MsgDone, which passes control through SUSR\_TaskI2C\_MsgDone in the SUSR layer back to UI2C\_MsgDone in the application layer. This triggers the next application I2C state.

Fig01 SRTX.wmfC:\Users\Stephen\Documents\PICdem2plus\Fig02 SLCD.wmf

Fig03 SI2C.wmfFig04 SI2C State.wmf

Fig05 Call Tree Init.wmf

Fig06 Call Tree Dispatcher.wmf

Fig07 Call Tree I2C.wmf