NUFR on the MSP430

Copyright © 2018, Bernie Woodland

All rights reserved.

Redistribution and use, with or without

modification, are permitted provided that the following conditions are met:

1. Redistributions must retain the above copyright notice, this

list of conditions and the following disclaimer.

THIS DOCUMENTATION IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE

DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;

LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Arm® is a trademark of Arm Holdings

Arm® Cortex® is a trademark of Arm Holdings

MSP430® is a trademark of Texas Instruments

Table of Contents

[Introduction 4](#_Toc528441192)

[Feature Set 4](#_Toc528441193)

[Make and Run Settings 5](#_Toc528441194)

[Using Interrupts 5](#_Toc528441195)

[Low Power Modes 6](#_Toc528441196)

[Tickless vs. Tickful Implementations 6](#_Toc528441197)

# Introduction

This document explains how to use NUFR on the MSP430 family of CPUs. The MSP430 is a 16-bit (traditionally, extended to 20-bit) CPU from Texas Instruments. The MSP430 consumes less power than most any CPU on the market, and has a well-thought out power savings strategy that permeates its rich set of peripherals. The MSP430 covers a large range of resource sizes; the top-end of the product line has enough resources for an MSP430 system developer to justify running a full-fledged RTOS, and not just a "scheduler".

NUFR adapts itself to the constraints of the MSP430, providing the advanced features of an RTOS in a frugal RAM footprint.

# Feature Set

NUFR’s flexibility allows for a wide range of configurations and usages of the real time environment. Furthermore, NUFR itself does not contain drivers or a board support package, as its philosophy is to give the system developer room to dictate things, rather than the OS. Nevertheless, a prototype MSP430 project is delivered with the Raging distribution as a skeleton for the most useful MSP430 NUFR usage. This project (the *Simple Project*) has the following features:

* Built with the most current (as of Aug/2018) GCC cross-compiler for the MSP430, which is the Somnium GCC compiler for the MSP430 (previously known as the *Red Hat Compiler)*
* Linux make environment; all free tools; Code Composer Studio not used; GDB capable
* Installation of MSP430 GCC compiler fully documented. Full documentation for setting up Ubuntu 18.04 LTS to run GCC toolchain.
* Support for MSP430 and MSP430x, both 16- and 20-bit models
* Support for all MSP430 variants through single-point makefile change
* NUFR used in a tickless OS configuration
* Ability for system developer to add OS tick capability according to his/her preferences, should it be needed
* Strategy for unlocking interrupts in interrupt handlers
* Straightforward power savings strategy implemented at app level, not in OS
* Proof-of-concept on MSP430F5529-based USB Launchpad

The features of NUFR and the Raging Distribution, which are fully documented in the user manual, distinguish NUFR from other OS’s that have been used on the MSP430:

* Preemptive multitasking and/or Round-Robin multitasking
* NUFR API calls from interrupt handlers; direct context switching from interrupt handlers
* Use of NUFR Service Layer (SL) App Timers, which allow timers to be set for timeouts in the range of 1 millisec to 3 days

RAM and FLASH consumption and CPU performance numbers are listed in *performance-numbers.xls.*

# Make and Run Settings

Instructions for making and running for the MSP430 are given in *getting-started.txt.*

To change the memory model and/or the target CPU type, modify the following section of the file *./makefiles/msp430-makefile.mk* and rebuild:

# \*\*\* Memory Model  
# Compile in either [A] or [B], but not both:  
#  
# [A] Small (16-bit) model  
#MEM\_MODEL=-msmall  
#MEM\_MODEL\_SWITCH="-D SMALL\_MODEL"  
# [B] Large (20-bit) model  
MEM\_MODEL=-mlarge  
MEM\_MODEL\_SWITCH="-D LARGE\_MODEL"  
  
# \*\*\*\* CPU Name  
# use lower case only for CPU\_NAME  
CPU\_NAME=msp430f5529  
CPU = -mmcu=${CPU\_NAME}

There are three memory models:

1. MSP430 16-bit mode. Code generated which will run in the legacy MSP430 mode. When a non-MSP430X is compiled, this is the only mode available.
2. MSP430X 16-bit mode. Code generated which will run in 16-bit mode on the MSP430X. When 16-bit mode is selected for an MSP430X target, this mode is automatically selected, rather than the MSP430 16-bit mode. Note that code generated in this mode is not compatible with the legacy MSP430 CPUs. Also note that, when this mode is used, only the FLASH and RAM which is mapped to addresses less than 0x10000 is available to the application.
3. MSP430X 20-bit mode. This is mode must be selected in order to use the FLASH and/or RAM on MSP430X chips which is mapped to addresses greater than 0xFFFF.

# Using Interrupts

Because of the limitations of the MSP430 interrupt facilities, the NUFR port to the MSP430 added workarounds, so that NUFR’s full feature set can be enjoyed on the 430. These workarounds are documented in comments in *./platform/MSP430/msp430-gcc/msp430-irq-entry-context-swith.h.* Failure to follow these guidelines will result in a crash.

# Low Power Modes

The Simple Project is a project intended to run on a battery or as an otherwise low-power system. To do this, one need simply add a reduced power mode function call to the background task. Here’s a hypothetical example:

int main(void)

{

nufr\_launch\_task(SOME\_TID, 0);

while (1)

{

some\_optional\_function\_call();

// Suspend execution by going into a low power mode

\_\_low\_power\_mode\_3();

}

}

NUFR converts the *main()* function, which is the codebase’s principal entry point, into the *Background Task* (*BG Task*): it runs infinitely without exiting *main()*. The BG Task only runs when no other task needs the CPU, as the purpose of the BG Task is to consume the CPU’s idle time. In a low-power system, this is the perfect location to place the CPU in a low-power mode state—to essentially turn the CPU off. In an MSP430, when an interrupt occurs, the low-power mode is cancelled, and the CPU resumes operation. Whether NUFR runs in a tickless or tickful mode, in either case, once the BG Task runs, a context switch out of the BG Task is always triggered by an interrupt. Therefore, the low power mode naturally gets cancelled in the sequence of events when context switching out of the BG Task to another task.

The problem occurs when context switching from a task to the BG Task. The switch out of the BG Task pushes a Status Register (SR) value which has the power mode bits set. Restoring this same SR will instantly cause the BG Task to go back into a sleep mode. In other words, the *while (1)* loop shown above won’t loop around, and *some\_optional\_function\_call()* won’t get to execute, each time the BG Task resumes.

To prevent this from happening, there is logic built into the NUFR MSP430 port’s context switch logic which clears all power savings bits in the SR before popping it off the stack.

# Tickless vs. Tickful Implementations

Although the NUFR philosophy is to avoid owning any hardware peripherals, drivers, etc., but rather letting the system engineer write code for any timer, etc. needed by NUFR and having him or her attach the timer to NUFR in the manner that he or she sees fit, the Raging distribution/Simple Project supplies a working MSP430 TimerA driver—otherwise, it would not be possible to demonstrate the Simple Project on the LaunchPad board or on any other platform for that matter. Plus, many end-users may not have a favorite TimerA driver available and will use the one supplied in the Raging distribution—perhaps just as a starting point or as reference code.

To understand tickless and tickful NUFR implementations on the MSP430, one must understand how part of the NUFR kernel (and by extension the app timer SL component) works, as it works differently than most other RTOSs. The NUFR kernel/SL exposes a few API calls that the surrounding code—such as the BSP, drivers, interrupt handlers, etc—call in accordance with a timing contract. These function calls are:

nufrplat\_systick\_handler()

nufrkernel\_update\_task\_timers()

nsvc\_timer\_expire\_timer\_callin()

The Simple Project does not call *nufrkernel\_update\_task\_timers().* By not calling *nufrkernel\_update\_task\_timers()*, the NUFR implementation becomes tickles. In a tickless NUFR implementation, the following NUFR APIs cannot be used:

nufr\_sleep()

nufr\_bop\_waitT()

nufr\_msg\_getT()

nufr\_sema\_getT()

…plus any SL API which calls any of the above

But to make NUFR tickful, one simply needs to call *nufrkernel\_update\_task\_timers()* once every *NUFR\_TICK\_PERIOD* milliseconds. That’s all, and NUFR becomes a tickful OS. So, anyone who wishes to can fix an MSP430 codebase, making NUFR tickful, by using a second timer to drive *nufrkernel\_update\_task\_timers().* Or, one could use the same timer to drive both, like the ARM code does in some of the *nufrplat\_systick\_handler()* functions.

But does running a tickless NUFR implementation present that many challenges and restrictions? Judicious use of app timers can compensate for the lack of availability of the NUFR tickful API calls, the ones listed above. It’s just a matter of having a good firmware architecture in place, for the most part, and for the few exceptions, figuring out how to work around them.