Week 9

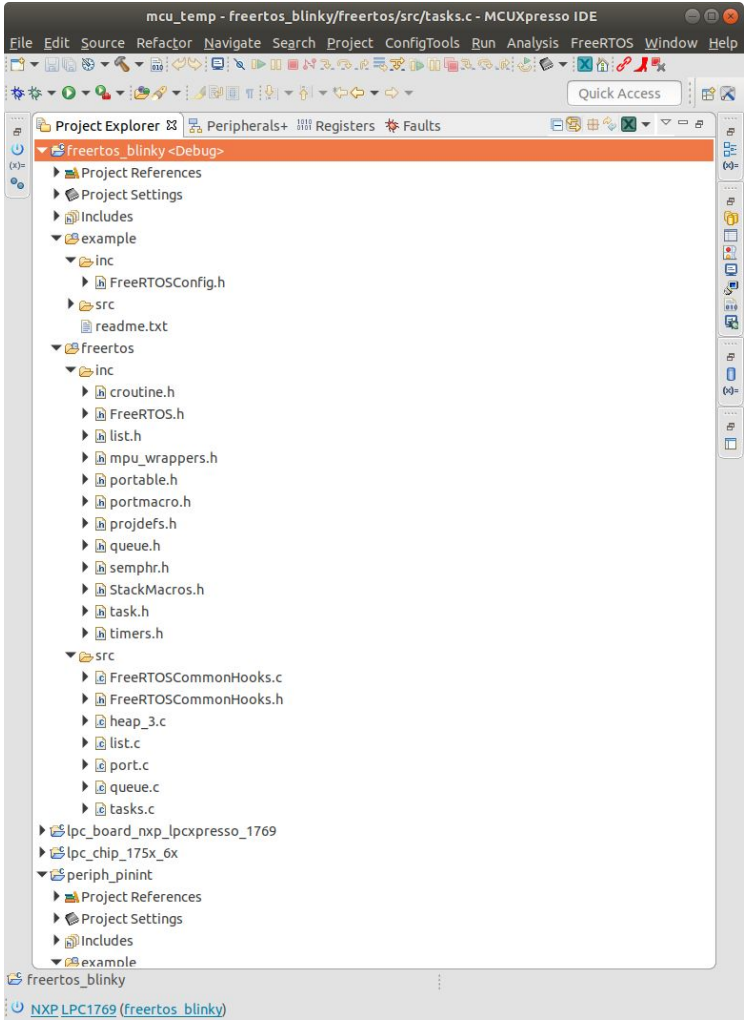
ESE3025

FreeRTOS Overview

- library components:
  - FreeRTOSConfig.h
    - a file containing macro definitions (#define) for many of the RTOS critical parameters, including: tick rate, preemption mode, etc.
  - FreeRTOS.h
    - must be included with any application that uses FreeRTOS, i.e., you must have #include<FreeRTOS.h>
  - API-specific header files:
    - task.h
    - queue.h
    - timers.h
    - list.h
    - projdefs.h
    - semphr.h

FreeRTOS Overview (2)

- API source files:
  - tasks.c
  - queue.c
  - list.c
  - port.c
- see the FreeRTOS Reference Manual provided here (to see a breakdown of API functions by API Group, e.g., Task and Scheduler API, Software Timer API, Semaphore API, etc.):
  - https://github.com/takiszourntos/teaching/blob/master/documentation/FreeRTOS_Reference_Manual_V10.0.0.pdf


mcu_temp - freertos_blinky/freertos/src/tasks.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run Analysis FreeRTOS Window Help
Quick Access
Project Explorer
Peripherals+
Registers
Faults
freertos_blinky <Debug>
Project References
Project Settings
Includes
example
inc
FreeRTOSConfig.h
src
readme.txt
freertos
inc
croutine.h
FreeRTOS.h
list.h
mpu_wrappers.h
portable.h
portmacro.h
projdefs.h
queue.h
semphr.h
StackMacros.h
task.h
timers.h
src
FreeRTOSCommonHooks.c
FreeRTOSCommonHooks.h
heap_3.c
list.c
port.c
queue.c
tasks.c
lpc_board_nxp_lpcxpresso_1769
lpc_chip_175x_6x
periph_pinint
Project References
Project Settings
Includes
example
freertos_blinky
NXP LPC1769 (freertos_blinky)

FreeRTOS Conventions

- see Appendix 1 in the FreeRTOS Reference Manual

APPENDIX 1: Data Types and Coding Style Guide

Data Types

Each port of FreeRTOS has a unique portmacro.h header file that contains (amongst other things) definitions for two special data types, TickType_t and BaseType_t. These data types are described in Table 3.

Table 3. Special data types used by FreeRTOS

Macro or typedef used	Actual type
TickType_t	This is used to store the tick count value, and by variables that specify block times. TickType_t can be either an unsigned 16-bit type or an unsigned 32-bit type, depending on the setting of configUSE_16_BIT_TICKS within FreeRTOSConfig.h. Using a 16-bit type can greatly improve efficiency on 8-bit and 16-bit architectures, but severely limits the maximum block period that can be specified. There is no reason to use a 16-bit type on a 32-bit architecture.
BaseType_t	This is always defined to be the most efficient data type for the architecture. Typically, this is a 32-bit type on a 32-bit architecture, a 16-bit type on a 16-bit architecture, and an 8-bit type on an 8-bit architecture. BaseType_t is generally used for variables that can take only a very limited range of values, and for Booleans.

Standard data types other than 'char' are not used (see below), instead type names defined within the compiler's stdint.h header file are used. 'char' types are only permitted to point to ASCII strings or reference single ASCII characters.

Variable Names

Variables are prefixed with their type: ‘c’ for char, ‘s’ for short, ‘l’ for long, and ‘x’ for BaseType_t and any other types (structures, task handles, queue handles, etc.).

If a variable is unsigned, it is also prefixed with a ‘u’. If a variable is a pointer, it is also prefixed with a ‘p’. Therefore, a variable of type unsigned char will be prefixed with ‘uc’, and a variable of type pointer to char will be prefixed with ‘pc’.

Function Names

Functions are prefixed with both the type they return and the file they are defined in. For example:

- v**Task**PrioritySet() returns a void and is defined within **task**.c.
- x**Queue**Receive() returns a variable of type BaseType_t and is defined within **queue**.c.
- v**Semaphore**CreateBinary() returns a void and is defined within **semphr**.h.

File scope (private) functions are prefixed with ‘prv’.

Formatting

One tab is always set to equal four spaces.

Macro Names

Most macros are written in upper case and prefixed with lower case letters that indicate where the macro is defined. Table 4 provides a list of prefixes.

Table 4. Macro prefixes

Prefix	Location of macro definition
port (for example, portMAX_DELAY)	portable.h
task (for example, taskENTER_CRITICAL())	task.h
pd (for example, pdTRUE)	projdefs.h
config (for example, configUSE_PREEMPTION)	FreeRTOSConfig.h
err (for example, errQUEUE_FULL)	projdefs.h

Note that the semaphore API is written almost entirely as a set of macros, but follows the function naming convention, rather than the macro naming convention.

The macros defined in Table 5 are used throughout the FreeRTOS source code.

Table 5. Common macro definitions

Macro	Value
pdTRUE	1
pdFALSE	0
pdPASS	1
pdFAIL	0

Rationale for Excessive Type Casting

The FreeRTOS source code can be compiled with many different compilers, all of which differ in how and when they generate warnings. In particular, different compilers want casting to be used in different ways. As a result, the FreeRTOS source code contains more type casting than would normally be warranted.

a note on: vTaskDelay()

- vTaskDelay() creates a delay with a specific number of TICKS
- but it also does something else important: while it is active, it places the task running it into BLOCKED state, allowing other tasks to run in that time
- if you want a delay without putting your task into the BLOCKED state, you can try a couple of things:
  - use a for(;;) loop
  - use taskENTER_Critical() and taskEXIT_Critical() --- suspends preemption

vTaskDelayUntil() versus vTaskDelay()