

Using the FreeRTOS™ Real Time Kernel

NXP LPC17xx Edition

Richard Barry

Contents

List of Figures	vi
List of Code Listings	viii
List of Tables	xi
List of Notation.....	xii
Preface	
FreeRTOS and the LPC17xx	1
Multitasking on an LPC17xx Cortex-M3 Microcontroller	2
An Introduction to Multitasking in Small Embedded Systems	2
A Note About Terminology	2
Why Use a Real-time Kernel?	3
The LPC17xx Port of FreeRTOS.....	4
Resources Used By FreeRTOS	5
The FreeRTOS, OpenRTOS, and SafeRTOS Family.....	6
Using the Examples that Accompany this Book.....	9
Required Tools and Hardware	9
Opening the Example Workspaces	10
Building the Examples	12
Chapter 1 Task Management.....	15
1.1 Chapter Introduction and Scope.....	16
Scope	16
1.2 Task Functions.....	17
1.3 Top Level Task States	18
1.4 Creating Tasks.....	19
The xTaskCreate() API Function	19
Example 1. Creating tasks	22
Example 2. Using the task parameter	25
1.5 Task Priorities	27
Example 3. Experimenting with priorities.....	28
1.6 Expanding the 'Not Running' State.....	31
The Blocked State.....	31
The Suspended State	32
The Ready State	32
Completing the State Transition Diagram.....	32
Example 4. Using the Blocked state to create a delay.....	33
The vTaskDelayUntil() API Function	36
Example 5. Converting the example tasks to use vTaskDelayUntil()	38
Example 6. Combining blocking and non-blocking tasks	39
1.7 The Idle Task and the Idle Task Hook.....	42

Idle Task Hook Functions	42
Limitations on the Implementation of Idle Task Hook Functions	43
Example 7. Defining an idle task hook function	43
1.8 Changing the Priority of a Task	45
The vTaskPrioritySet() API Function	45
The uxTaskPriorityGet() API Function	45
Example 8. Changing task priorities	46
1.9 Deleting a Task	51
The vTaskDelete() API Function	51
Example 9. Deleting tasks	52
1.10 The Scheduling Algorithm—A Summary	55
Prioritized Pre-emptive Scheduling	55
Selecting Task Priorities	57
Co-operative Scheduling	57
Chapter 2 Queue Management	61
2.1 Chapter Introduction and Scope	62
Scope	62
2.2 Characteristics of a Queue	63
Data Storage	63
Access by Multiple Tasks	63
Blocking on Queue Reads	63
Blocking on Queue Writes	64
2.3 Using a Queue	66
The xQueueCreate() API Function	66
The xQueueSendToBack() and xQueueSendToFront() API Functions	67
The xQueueReceive() and xQueuePeek() API Functions	69
The uxQueueMessagesWaiting() API Function	72
Example 10. Blocking when receiving from a queue	73
Using Queues to Transfer Compound Types	77
Example 11. Blocking when sending to a queue or sending structures on a queue	79
2.4 Working with Large Data	85
Chapter 3 Interrupt Management	87
3.1 Chapter Introduction and Scope	88
Events	88
Scope	88
3.2 Deferred Interrupt Processing	90
Binary Semaphores Used for Synchronization	90
Writing FreeRTOS Interrupt Handlers	91
The vSemaphoreCreateBinary() API Function	91
The xSemaphoreTake() API Function	94
The xSemaphoreGiveFromISR() API Function	95
Example 12. Using a binary semaphore to synchronize a task with an interrupt	97

3.3	Counting Semaphores	102
	The xSemaphoreCreateCounting() API Function	105
	Example 13. Using a counting semaphore to synchronize a task with an interrupt.....	107
3.4	Using Queues within an Interrupt Service Routine	109
	The xQueueSendToFrontFromISR() and xQueueSendToBackFromISR() API Functions	109
	Efficient Queue Usage	111
	Example 14. Sending and receiving on a queue from within an interrupt	111
3.5	Interrupt Nesting	116
Chapter 4	Resource Management	121
4.1	Chapter Introduction and Scope.....	122
	Mutual Exclusion.....	124
	Scope	125
4.2	Critical Sections and Suspending the Scheduler	126
	Basic Critical Sections	126
	Suspending (or Locking) the Scheduler	127
	The vTaskSuspendAll() API Function.....	128
	The xTaskResumeAll() API Function	128
4.3	Mutexes (and Binary Semaphores).....	130
	The xSemaphoreCreateMutex() API Function.....	132
	Example 15. Rewriting vPrintString() to use a semaphore	132
	Priority Inversion	135
	Priority Inheritance	136
	Deadlock (or Deadly Embrace)	137
4.4	Gatekeeper Tasks.....	139
	Example 16. Re-writing vPrintString() to use a gatekeeper task.....	139
Chapter 5	Memory Management.....	145
5.1	Chapter Introduction and Scope.....	146
	Scope	147
5.2	Example Memory Allocation Schemes	148
	Heap_1.c	148
	Heap_2.c	149
	Heap_3.c	151
	The xPortGetFreeHeapSize() API Function	151
Chapter 6	Trouble Shooting	155
6.1	Chapter Introduction and Scope.....	156
	printf-stdarg.c.....	156
6.2	Stack Overflow.....	157
	The uxTaskGetStackHighWaterMark() API Function	157
	Run Time Stack Checking—Overview	158
	Run Time Stack Checking—Method 1	158
	Run Time Stack Checking—Method 2	159

6.3	Other Common Sources of Error	160
	Symptom: Adding a simple task to a demo causes the demo to crash	160
	Symptom: Using an API function within an interrupt causes the application to crash....	160
	Symptom: Sometimes the application crashes within an interrupt service routine	161
	Symptom: Critical sections do not nest correctly	161
	Symptom: The application crashes even before the scheduler is started.....	161
	Symptom: Calling API functions while the scheduler is suspended causes the application to crash	162
	Symptom: The prototype for pxPortInitialiseStack() causes compilation to fail.....	162
6.4	Execution Visualization—Trace Hook Macros	163
	Available Trace Hook Macros.....	163
	Defining Trace Hook Macros	172
	Example 17. Using Trace Hook Macros	173
6.5	Execution Visualization—Run Time Statistics.....	178
	Run Time Statistics Time Base	178
	Configuring an Application to Collect Run Time Statistics	179
	The vTaskGetRunTimeStats() API Function.....	180
	Example 18. Generating Run Time Statistics	181
Chapter 7	FreeRTOS-MPU	187
7.1	Chapter Introduction and Scope	188
	Scope.....	188
7.2	Access Permissions	189
	User Mode and Privileged Mode	189
	Access Permission Attributes	189
7.3	Defining an MPU Region	191
	Overlapping Regions.....	191
	Predefined Regions and Task Definable Regions	191
	Region Start Address and Size Constraints.....	192
7.4	The FreeRTOS-MPU API	194
	The xTaskCreateRestricted() API Function	194
	Using xTaskCreate() with FreeRTOS-MPU	199
	The vTaskAllocateMPURegions() API Function	200
	The portSWITCH_TO_USER_MODE() API Macro.....	202
7.5	Linker Configuration	203
7.6	Practical Usage Tips	206
	Accessing Data from a User Mode Task	206
	Intertask Communication from User Mode	207
	FreeRTOS-MPU Demo Projects	207
Chapter 8	The FreeRTOS Download	209
8.1	Chapter Introduction and Scope	210
	Scope.....	210
8.2	Files and Directories.....	211

Removing Unused Source Files.....	212
8.3 Demo Applications	213
Removing Unused Demo Files.....	214
8.4 Creating a FreeRTOS Project	215
Adapting One of the Supplied Demo Projects	215
Creating a New Project from Scratch	216
Header Files	217
8.5 Data Types and Coding Style Guide	218
Data Types	218
Variable Names	219
Function Names.....	219
Formatting.....	219
Macro Names	219
Rationale for Excessive Type Casting.....	220
Appendix 1: Licensing Information.....	223
Open Source License Details.....	224
GPL Exception Text	225
INDEX	227

List of Figures

Figure 1. Locating the 'Change Target MCU' speed button within the LPCXpresso IDE (highlighted by the box with round corners).....	9
Figure 2. Block diagram of the LPC17xx	10
Figure 3. Selecting the option to import existing projects into the workspace	11
Figure 4. Selecting the .zip archive	11
Figure 5. The projects listed in the Project Explorer window of the LPCXpresso IDE	12
Figure 6. Locating the 'Debug' speed button within the LPCXpresso IDE (highlighted by the box with round corners).....	12
Figure 7. Top level task states and transitions.....	18
Figure 8. The output produced when Example 1 is executed	23
Figure 9. The execution pattern of the two Example 1 tasks	24
Figure 10. The execution sequence expanded to show the tick interrupt executing	28
Figure 11. Running both test tasks at different priorities.....	29
Figure 12. The execution pattern when one task has a higher priority than the other	30
Figure 13. Full task state machine.....	33
Figure 14. The output produced when Example 4 is executed	35
Figure 15. The execution sequence when the tasks use vTaskDelay() in place of the NULL loop.....	35
Figure 16. Bold lines indicate the state transitions performed by the tasks in Example 4	36
Figure 17. The output produced when Example 6 is executed	40
Figure 18. The execution pattern of Example 6	41
Figure 19. The output produced when Example 7 is executed	44
Figure 20. The sequence of task execution when running Example 8	49
Figure 21. The output produced when Example 8 is executed	50
Figure 22. The output produced when Example 9 is executed	53
Figure 23. The execution sequence for Example 9	54
Figure 24. Execution pattern with pre-emption points highlighted.....	56
Figure 25. An example sequence of writes and reads to and from a queue	65
Figure 26. The output produced when Example 10 is executed	77
Figure 27. The sequence of execution produced by Example 10	77
Figure 28. An example scenario where structures are sent on a queue	78
Figure 29. The output produced by Example 11	82
Figure 30. The sequence of execution produced by Example 11	83
Figure 31. The interrupt interrupts one task but returns to another.....	90
Figure 32. Using a binary semaphore to synchronize a task with an interrupt	93
Figure 33. The output produced when Example 12 is executed	100
Figure 34. The sequence of execution when Example 12 is executed	101
Figure 35. A binary semaphore can latch at most one event.....	103
Figure 36. Using a counting semaphore to 'count' events	104
Figure 37. The output produced when Example 13 is executed	108
Figure 38. The output produced when Example 14 is executed	115

Figure 39. The sequence of execution produced by Example 14	115
Figure 40. Constants affecting interrupt nesting behavior	118
Figure 41. Mutual exclusion implemented using a mutex	131
Figure 42. The output produced when Example 15 is executed	135
Figure 43. A possible sequence of execution for Example 15	135
Figure 44. A worst case priority inversion scenario	136
Figure 45. Priority inheritance minimizing the effect of priority inversion	137
Figure 46. The output produced when Example 16 is executed	143
Figure 47. RAM being allocated within the array each time a task is created	148
Figure 48. RAM being allocated from the array as tasks are created and deleted	150
Figure 49. The output produced when Example 17 is executed	177
Figure 50. The output produced when Example 18 is executed	184
Figure 51. The top-level directories—Source and Demo	211
Figure 52. The three core files that implement the FreeRTOS kernel	212
Figure 53. The source directories required to build an LPC17xx demo application	212
Figure 54. The demo directories required to build an LPC17xx demo application	214

List of Code Listings

Listing 1. The task function prototype	17
Listing 2. The structure of a typical task function	17
Listing 3. The xTaskCreate() API function prototype	19
Listing 4. Implementation of the first task used in Example 1	22
Listing 5. Implementation of the second task used in Example 1.....	22
Listing 6. Starting the Example 1 tasks	23
Listing 7. Creating a task from within another task after the scheduler has started.....	24
Listing 8. The single task function used to create two tasks in Example 2	25
Listing 9. The main() function for Example 2	26
Listing 10. Creating two tasks at different priorities	29
Listing 11. The vTaskDelay() API function prototype	34
Listing 12. The source code for the example task after the null loop delay has been replaced by a call to vTaskDelay()	34
Listing 13. vTaskDelayUntil() API function prototype	37
Listing 14. The implementation of the example task using vTaskDelayUntil().....	38
Listing 15. The continuous processing task used in Example 6.....	39
Listing 16. The periodic task used in Example 6.....	40
Listing 17. The idle task hook function name and prototype.	43
Listing 18. A very simple Idle hook function.....	43
Listing 19. The source code for the example task prints out the ulIdleCycleCount value	44
Listing 20. The vTaskPrioritySet() API function prototype.....	45
Listing 21. The uxTaskPriorityGet() API function prototype	45
Listing 22. The implementation of Task 1 in Example 8.....	47
Listing 23. The implementation of Task 2 in Example 8.....	48
Listing 24. The implementation of main() for Example 8.....	49
Listing 25. The vTaskDelete() API function prototype.....	51
Listing 26. The implementation of main() for Example 9.....	52
Listing 27. The implementation of Task 1 for Example 9	53
Listing 28. The implementation of Task 2 for Example 9	53
Listing 29. The xQueueCreate() API function prototype	66
Listing 30. The xQueueSendToFront() API function prototype	67
Listing 31. The xQueueSendToBack() API function prototype.....	67
Listing 32. The xQueueReceive() API function prototype	70
Listing 33. The xQueuePeek() API function prototype	70
Listing 34. The uxQueueMessagesWaiting() API function prototype	72
Listing 35. Implementation of the sending task used in Example 10.....	74
Listing 36. Implementation of the receiver task for Example 10.....	75
Listing 37. The implementation of main() for Example 10.....	76
Listing 38. The definition of the structure that is to be passed on a queue, plus the declaration of two variables for use by the example	79
Listing 39. The implementation of the sending task for Example 11.	80

Listing 40. The definition of the receiving task for Example 11	81
Listing 41. The implementation of main() for Example 11.....	82
Listing 42. The vSemaphoreCreateBinary() API function prototype	92
Listing 43. The xSemaphoreTake() API function prototype	94
Listing 44. The xSemaphoreGiveFromISR() API function prototype.....	95
Listing 45. Implementation of the task that periodically generates a software interrupt in Example 12.....	97
Listing 46. The implementation of the handler task (the task that synchronizes with the interrupt) in Example 12.....	98
Listing 47. The software interrupt handler used in Example 12	99
Listing 48. The implementation of main() for Example 12.....	100
Listing 49. The xSemaphoreCreateCounting() API function prototype	105
Listing 50. Using xSemaphoreCreateCounting() to create a counting semaphore.....	107
Listing 51. The implementation of the interrupt service routine used by Example 13.....	107
Listing 52. The xQueueSendToFrontFromISR() API function prototype	109
Listing 53. The xQueueSendToBackFromISR() API function prototype	109
Listing 54. The implementation of the task that writes to the queue in Example 14.....	112
Listing 55. The implementation of the interrupt service routine used by Example 14.....	113
Listing 56. The task that prints out the strings received from the interrupt service routine in Example 14.....	114
Listing 57. The main() function for Example 14	114
Listing 58. Using a CMSIS function to set an interrupt priority.....	116
Listing 59. An example read, modify, write sequence	122
Listing 60. An example of a reentrant function	124
Listing 61. An example of a function that is not reentrant.....	124
Listing 62. Using a critical section to guard access to a variable	126
Listing 63. A possible implementation of vPrintString().....	126
Listing 64. The vTaskSuspendAll() API function prototype.....	128
Listing 65. The xTaskResumeAll() API function prototype.....	128
Listing 66. The implementation of vPrintString().....	129
Listing 67. The xSemaphoreCreateMutex() API function prototype.....	132
Listing 68. The implementation of prvNewPrintString().....	133
Listing 69. The implementation of prvPrintTask() for Example 15	133
Listing 70. The implementation of main() for Example 15.....	134
Listing 71. The name and prototype for a tick hook function	140
Listing 72. The gatekeeper task	140
Listing 73. The print task implementation for Example 16	141
Listing 74. The tick hook implementation	141
Listing 75. The implementation of main() for Example 16.....	142
Listing 76. The heap_3.c implementation.....	151
Listing 77. The xPortGetFreeHeapSize() API function prototype.....	151
Listing 78. The uxTaskGetStackHighWaterMark() API function prototype.....	157
Listing 79. The stack overflow hook function prototype	158
Listing 80. The malloc() failed hook function name and prototype.....	160

Listing 81. The trace macros demonstrated by Example 17	175
Listing 82. The definition of the task created by Example 17	176
Listing 83. The vTaskGetRunTimeStats() API function prototype	180
Listing 84. Configuration of the timer 0 (TIM0) peripheral used in Example 18	181
Listing 85. Macros added to FreeRTOSConfig.h to enable the collection of run time statistics in Example 18.....	181
Listing 86. The definition of two tasks created in Example 18 to use up some processing time	182
Listing 87. The task that prints out the collected run time statistics in Example 18	183
Listing 88. Syntax required by GCC, IAR, and Keil compilers to force a variable onto a particular byte alignment (1024-byte alignment in this example)	192
Listing 89. Defining two arrays that may be placed in adjacent memory	192
Listing 90. The xTaskCreateRestricted() API function prototype	194
Listing 91. Definition of the structures required by the xTaskCreateRestricted() API function	195
Listing 92. Using the xTaskParameters structure	198
Listing 93. Using xTaskCreate() to create both User mode and Privileged mode task with FreeRTOS-MPU	200
Listing 94. The vTaskAllocateMPURegions() API function prototype.....	200
Listing 95. Using vTaskAllocateMPURegions() to redefine the MPU regions associated with a task.....	201
Listing 96. Defining the memory map and linker variables using GNU LD syntax.....	204
Listing 97. Defining the privileged_functions named section using GNU LD syntax.....	205
Listing 98. Copying data into a stack variable before setting the task into User mode.....	206
Listing 99. Copying the value of a global variable into a stack variable using the task parameter	207
Listing 100. The template for a new main() function	216

List of Tables

Table 1. Comparing the FreeRTOS license with the OpenRTOS license	7
Table 2. xTaskCreate() parameters and return value	19
Table 3. vTaskDelay() parameters	34
Table 4. vTaskDelayUntil() parameters	37
Table 5. vTaskPrioritySet() parameters	45
Table 6. uxTaskPriorityGet() parameters and return value	46
Table 7. vTaskDelete() parameters	51
Table 8. xQueueCreate() parameters and return value	66
Table 9. xQueueSendToFront() and xQueueSendToBack() function parameters and return value	67
Table 10. xQueueReceive() and xQueuePeek() function parameters and return values	70
Table 11. uxQueueMessagesWaiting() function parameters and return value	73
Table 12. Key to Figure 30	83
Table 13. vSemaphoreCreateBinary() parameters	92
Table 14. xSemaphoreTake() parameters and return value	94
Table 15. xSemaphoreGiveFromISR() parameters and return value	96
Table 16. xSemaphoreCreateCounting() parameters and return value	106
Table 17. xQueueSendToFrontFromISR() and xQueueSendToBackFromISR() parameters and return values	109
Table 18. Constants that affect interrupt nesting	117
Table 19. xTaskResumeAll() return value	128
Table 20. xSemaphoreCreateMutex() return value	132
Table 21. xPortGetFreeHeapSize() return value	152
Table 22. uxTaskGetStackHighWaterMark() parameters and return value	157
Table 23. Trace hook macros	163
Table 24. Macros used in the collection of run time statistics	179
Table 25. vTaskGetRunTimeStats() parameters	180
Table 26. MPU region access permissions	190
Table 27. xMemoryRegion structure members	195
Table 28. xTaskParameters structure members	196
Table 29. vTaskAllocateMPURegions() parameters	201
Table 30. Named linker sections required by FreeRTOS-MPU	203
Table 31. Linker variables required by FreeRTOS-MPU	203
Table 32. FreeRTOS source files to include in the project	217
Table 33. Special data types used by FreeRTOS	218
Table 34. Macro prefixes	220
Table 35. Common macro definitions	220
Table 36. Comparing the open source license with the commercial license	224

List of Notation

API	Application Programming Interface
CMSIS	Cortex Microcontroller Software Interface Standard
FAQ	Frequently Asked Question
FIFO	First In First Out
HMI	Human Machine Interface
IDE	Integrated Development Environment
IRQ	Interrupt Request
ISR	Interrupt Service Routine
LCD	Liquid Crystal Display
MCU	Microcontroller
MPU	Memory Protection Unit
RMS	Rate Monotonic Scheduling
RTOS	Real-time Operating System
SIL	Safety Integrity Level
TCB	Task Control Block
TIM0	The LPC17xx Peripheral Timer 0
UART	Universal Asynchronous Receiver/Transmitter