FreeRTOS - Common Task Design Patterns in Multi-tasking Applications

Richard Barry, Founder
Real Time Engineers Ltd.

Class ID: 9C11L
Introductions

- Real Time Engineers Ltd.
  - FreeRTOS™
  - FreeRTOS+™

- WITTENSTEIN high integrity systems
  - OpenRTOS®
  - SafeRTOS®

Richard Barry
Director, Real Time Engineers Ltd
Founder, the FreeRTOS project
Pre-requisites

- Working knowledge of the C programming language
- Appreciation of multi-tasking concepts

Learning Objectives

- See how an RTOS can simplify software design
- Learn how to use task and interrupt prioritisation to ensure determinism
- See how to use RTOS services to improve maintainability
- See when an RTOS allows more efficient CPU usage
- Get confidence in starting an RTOS based design
Pre-requisites

- Working knowledge of the C programming language
- Appreciation of multi-tasking concepts

Learning Objectives

- See how an RTOS can simplify software design
- Learn how to use task and interrupt prioritisation to ensure determinism
- See how to use RTOS services to improve maintainability
- See when an RTOS allows more efficient CPU usage
- Get confidence in starting an RTOS based design

In an hour! Scope is limited
Agenda

- Real Time Multitasking
  - The ‘RT’ in RTOS
  - FreeRTOS products and market position

- A Hypothetical Application
  - A closer look at a real time control task
  - Handling continuous, high frequency and periodic control

- Putting it all together
Agenda and Learning Objectives

- Real Time Multitasking
  - The ‘RT’ in RTOS
  - FreeRTOS products and market position

- A Hypothetical Application
  - A closer look at a real time control task
  - Handling continuous, high frequency and periodic control

- Putting it all together

In an hour!
Real Time Multitasking

Setting the scene
All available tasks appear to be executing ...

... but only one task is ever executing at any time.
The RT In RTOS

- Deterministic
  - Hard real time – “it absolutely must”
  - Soft real time – “it should”
Task States

- Suspended
- Ready
- Running
- Blocked

Event
Blocking API function called

Function called
RTOS, Kernel or Scheduler?
Professionally developed, quality controlled, robust and supported

Downloaded more than 170,000 times in 2010 & 2011 combined

Polled top in class for the questions “which kernel are you currently using?” and “which kernel are you most likely to use in your next project” in the 2011 and 2012 EETimes Embedded Market Surveys

is everywhere ....
31 architectures and 17 tool chains
Is It Free, Even For Commercial Use?

- Yes
  - Moderated open source
  - Proprietary code remains proprietary

FreeRTOS API

FreeRTOS Source

Application Source

Driver Source

Middleware Source

Closed Source

Open Source

Copyright 2012 Real Time Engineers Ltd - www.FreeRTOS.org
# FreeRTOS+ Ecosystem

<table>
<thead>
<tr>
<th>FreeRTOS+</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IO</strong></td>
<td></td>
</tr>
<tr>
<td>Add an open(), read(), write(), ioctl() peripheral interface to your application</td>
<td></td>
</tr>
<tr>
<td><strong>CLI</strong></td>
<td></td>
</tr>
<tr>
<td>Enable your application to efficiently process command line input</td>
<td></td>
</tr>
<tr>
<td><strong>Safety &amp; Certification</strong></td>
<td>A pre-certified kernel for microcontrollers, with a similar usage model to FreeRTOS</td>
</tr>
<tr>
<td><strong>Trace</strong></td>
<td>Get 15 graphically interlinked views of the trace, providing an unprecedented level of insight</td>
</tr>
<tr>
<td><strong>Nabto</strong></td>
<td>Exciting new technology that re-defines the web device, making connectivity simpler and safer than ever before</td>
</tr>
<tr>
<td><strong>TCP/IP</strong></td>
<td>Pre-built libraries for easy integration of TCP/IP and related protocols into cost-sensitive applications</td>
</tr>
<tr>
<td><strong>RTOS Training</strong></td>
<td>Expert instructor led RTOS training to maximise productivity – delivered online or on site</td>
</tr>
<tr>
<td><strong>SSL and TLS</strong></td>
<td>State of the art networking security for embedded systems</td>
</tr>
</tbody>
</table>
Problem Statement

Architecting a System
Hypothetical Application

- Elevator controller
  - Motor control
  - Connectivity
  - Safety functions
  - User interface
Categorising Requirements

Hard Real Time

- Motor commutation
- Power regeneration
- Safety functions
- Emergency stop
- Brakes
- Interlocks
- Audible signals/announcements
- Emergency call button

Soft Real Time

- Floor indicators
- Buttons in lift
- Buttons on floors
- Inter-elevator communication

Renesas DevCon October 2012

Copyright 2012 Real Time Engineers Ltd - www.FreeRTOS.org
Limited Scope: Motor Requirements

- Motor Control
  - 20kHz PWM
  - Commutation sequencing (~5kHz, ISR)
  - Servo function (PID, 250Hz)

- Communications
  - CANbus link to other motors, doors, buttons
Super Loop Solution

- Scan Keypad
- Decode Key
- Key Pressed

- Start conversion
- Scale result

- Initialize
- Read ADC
- Read Pot
- Read Motor position
- Perform control algorithm
- Drive Motor
- Function 3
- Function 4
- Function 5

- Function 1
- Function 2
- Function 3
- Function 4
- Function 5
Super Loop Solution

Will it scale?
## Scaling and Maintenance

- Interdependency between timing and functionality

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration of COTS components</td>
<td>😞</td>
</tr>
<tr>
<td>Code re-use</td>
<td>😞</td>
</tr>
<tr>
<td>Mixed processing requirements</td>
<td>😞</td>
</tr>
<tr>
<td>Concurrent team development</td>
<td>😞</td>
</tr>
<tr>
<td>Testability</td>
<td>😞</td>
</tr>
<tr>
<td>Mix of hard and soft real time</td>
<td>😞</td>
</tr>
<tr>
<td>Scalable (not susceptible to application change)</td>
<td>😞</td>
</tr>
<tr>
<td>Tolerant of hardware change</td>
<td>😞</td>
</tr>
</tbody>
</table>
High Level Design

Apportioning functionality
**Slow, Fast and Continuous Processing**

- **Outer loop PID control** (periodic, ms resolution)
  - HMI
  - Set points
  - State
  - Demand
  - CAN protocol
  - State
  - Link state
  - CAN (event driven)

- **Inner loop commutation** (µs resolution)
  - Motor Position
  - Demand
  - PWM values
  - PWM (continuous)

- **PWM**
  - Control

---

Renesas DevCon October 2012

Copyright 2012 Real Time Engineers Ltd - www.FreeRTOS.org
Inner Loop Control
Deterministic interrupt processing
Slow, Fast and Continuous Processing

HMI

Set points

State

Outer loop PID control
(periodic, ms resolution)

Demand

State

CAN protocol

Sync

PWM

Continuous

PWM/Time interrupt

Motor Position

Inner loop commutation
(µs resolution)

Demand

Motor Position

PWM values

PWM

Can (event driven)

Tasks

Interrupts

Peripherals

Renesas DevCon October 2012

Copyright 2012 Real Time Engineers Ltd - www.FreeRTOS.org
FreeRTOS RX600 Interrupt Nesting Model

Priorities 13, 14 and 15 are never disabled

- configKERNEL_INTERRUPT_PRIORITY
- configMAX_SYSCALL_INTERRUPT_PRIORITY

Priority 15
Priority 14
Priority 13
Priority 12
...  
Priority 4
Priority 3
Priority 2
Priority 1
Outer Loop PID Control

Techniques for periodic processing
Outer loop PID control
(periodic, ms resolution)

Inner loop commutation
(µs resolution)

PWM values

PWM (continuous)

HMI
Set points
State

Demand
State

CAN protocol
State

Motor Position

CAN
Link state

Sync

Demand

Tasks

Interrupts

Peripherals
Control Task Structure

- Delay (wait) until it is time to start the next control cycle
- Perform control function
- Output results

Note: It is a simple autonomous and sequential piece of code
/* Tasks always have the same prototype. */

void vOuterLoopPIDTask( void *pvParameters )
{
    for( ;; )
    {
        WaitNextControlCycle();
        PerformControlFunction();
        OutputResults();
    }

    /* A task cannot exit without first deleting itself. */
    vTaskDelete( NULL );
}
Creating The Control Task

```c
xTaskCreate(  /* A pointer to the task function. */
vOuterLoopPIDTask,

/* Textual name. */
"PID",

/* Dimensions of the task stack. */
configMINIMAL_STACK DEPTH,

/* Parameters passed into the task. */
(void *) 0,

/* Assign a high priority to the task. */
( configMAX_PRIORITIES - 1 ),

/* A handle for the task. */
NULL
);
while( ( xTaskGetTickCount() - xLastCycleTime ) < CONTROL_PERIOD );

- Non blocking (in RTOS terms) code
- Consumes CPU time while doing ‘nothing’
What would happen if Task2 used a null loop delay?
Execution pattern with Task2 using a null loop delay
WaitNextCycle() as System Call

- Time is specified in “ticks”
- Very efficient, only executes when there is work to be done

vTaskDelay(aTime);
vTaskDelayUntil(aReferencePoint, aTime);
Increase Accuracy with a Hardware Timer

Outer loop PID control (periodic, ms resolution)

Inner loop commutation (µs resolution)

PWM values

PWM (continuous)

HMI

Set points

State

CAN protocol

Demand

Link state

Sync

Tasks

Interrupts

Peripherals

Motor Position

State

Demand

H/W Timer

Renesas DevCon October 2012

Copyright 2012 Real Time Engineers Ltd - www.FreeRTOS.org
Interrupt Synchronisation

ISR

‘ISR’ Task

Task3

t1 t2 t3 t4

Task3 gets pre-empted!

Wait for semaphore (xSemaphoreTake())

PerformControlFunction()
OutputResults()
Interrupt Synchronisation

Give semaphore (xSemaphoreGiveFromISR())

Wait for semaphore (xSemaphoreTake())

Task3 gets pre-empted!

ISR

‘ISR’

Task3

t1 t2 t3 t4

PerformControlFunction()

OutputResults()
Interrupt Synchronisation

ISR

‘ISR’ Task

Task3

t1 t2 t3 t4

Task3 gets pre-empted!

Wait for semaphore (xSemaphoreTake())

PerformControlFunction()
OutputResults()
Interrupt Synchronisation

Task3 gets pre-empted!

ISR

‘ISR’ Task

Task3

t1 t2 t3 t4

Wait for semaphore (xSemaphoreTake())

PerformControlFunction()

OutputResults()
WaitNextControlCycle() as a Deferred Interrupt

- Time is specified in “ticks”
- Very efficient, only executes when there is work to be done

```c
xSemaphoreTake(xCtrlSem, portMAX_DELAY);

xSemaphoreGiveFromISR(xCtrlSem, &HigherPriorityTaskWoken);
```
What was it we were doing?

- Elevator controller
  - Motor control
  - Connectivity
  - Safety functions
  - User interface
Add In Connectivity

**HMI**
- Set points
- State

**Outer loop PID control**
- (periodic, ms resolution)

**Demand**

**CAN protocol**
- State

**Inner loop commutation**
- (µs resolution)

**Motor Position**

**PWM**
- (continuous)

**Tasks**
- Interrupts
- Peripherals

**PWM values**

**H/W Timer**
Tasks can communicate with each other, and with interrupts, using FreeRTOS queues.
Inter-task Communication

Message passing using queues
Delay (wait) until it is time to start the next control cycle

Perform control function

Output results

Read and process a value from queue (if any data is available)

Note: Read only a single item from the queue to ensure control cycle remains deterministic
Queues

- An ‘intelligent’ FIFO buffer
- Data is copied byte for byte the only truly flexible way
- Tasks/interrupts can access queues simultaneously
- Queues can hold a finite number of items
- The number and size of each queued item is set when the queue is created
/* The type of message passed to the PID control task. */
typedef enum
{
    eP_Value;
    eI_Value;
    eD_Value;
    eSetPointValue
} eParameterId;

/* The structure passed on the queue. How pvValue is interpreted by the PID control task depends on the eControlParameterId value. */
typedef struct xCTRL_PARAMETER_MESSAGE
{
    eParameterId;
    void *pvValue;
} xOurQueueMessage_t;
Creating a Queue

```c
xQueueHandle xQueue;
xQueue = xQueueCreate
    (  
      /* The length of the queue */
      3,
      /* The size of each item. */
      sizeof( xOurQueueMessage_t )
    );
```

---

Renesas DevCon October 2012

Copyright 2012 Real Time Engineers Ltd - www.FreeRTOS.org
Sending to a Queue

```c
xQueueSendToBack /* Or xQueueSendToFront() */
(
    /* The handle of the queue */
    xQueue,

    /* The item being sent */
    &xFirstMessage,

    /* The number of ticks to block if the queue is already full. */
    portMAX_DELAY
);
```
Sending to a Queue

xQueueSendToBack /* Or xQueueSendToFront() */
(
    /* The handle of the queue */
    xQueue,

    /* The item being sent */
    &xSecondMessage,

    /* The number of ticks to block if the queue is already full. */
    portMAX_DELAY
);

/* Or xQueueSendToFront() */
Receiving from a queue

xOurQueueMessage_t xReceived;

xQueueReceive
    (/* The handle of the queue */
     xQueue,

     /* Variable into which the item will be placed. */
     &xReceived,

     /* This time a block time of zero is specified as the control task must not block on the queue. */
     0
    );
uint32_t ulSetPoint;
uint16_t usP, usI, usD;

switch( xReceived.eParameterId )
{
    case eSetPointValue :
        ulSetPoint = *( ( uint32_t * ) xReceived.pvValue );
        break;

    case eI_Value :
        usI = *( ( uint16_t * ) xReceived.pvValue );
        break;

    /* Etc. */
}

Interpreting the Received Message
Putting It All Together

Outer loop PID control (periodic, ms resolution)

Inner loop commutation (µs resolution)

PWM (continuous)

HMI
Set points
State
H/W Timer

Demand

CAN protocol
State

Tasks

Interrupts

Peripherals

PWM values

Motor Position

Demand

Motor Position

CAN (event driven)
void vOuterLoopPIDTask( void *pvParameters )
{
  xOurQueueMessage_t xMessage;

  for( ;; )
  {
    xSemaphoreTake( xCtrlCycleSemaphore, portMAX_DELAY );
    PerformControlFunction();
    OutputResults();
    if( xQueueReceive( xQueue, &xMessage, 0 ) == pdPASS )
    {
      ProcessReceivedMessage( xMessage );
    }
  }
}
Summary
Discussed Implementation Across All Partitions and Interfaces

- HMI
  - Set points
  - State
- Outer loop PID control (periodic, ms resolution)
- Demand
  - State
- CAN protocol
  - Sync
  - Link state
- Tasks
- Interrupts
- Peripherals

- Inner loop commutation (µs resolution)
  - Motor Position
  - PWM values
- PWM (continuous)
## Scaling and Maintenance

- Decoupled, functionally cohesive code

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration of COTS components</td>
<td>😞</td>
</tr>
<tr>
<td>Code re-use</td>
<td>😊</td>
</tr>
<tr>
<td>Mixed processing requirements</td>
<td>😊</td>
</tr>
<tr>
<td>Concurrent team development</td>
<td>😊</td>
</tr>
<tr>
<td>Testability</td>
<td>😞</td>
</tr>
<tr>
<td>Mix of hard and soft real time</td>
<td>😊</td>
</tr>
<tr>
<td>Scalable (not susceptible to application change)</td>
<td>😊</td>
</tr>
<tr>
<td>Tolerant of hardware change</td>
<td>😊</td>
</tr>
</tbody>
</table>
Taking It Further

There is a lot more!
The FreeRTOS Project

Don’t let your RTOS solution lock you in - FreeRTOS is the professional grade, cross platform standard for microcontrollers. 31 architectures - 18 toolchains, millions of product deployments - just 1 market leading RTOS.

* Immediate Free Download * Feature Rich * Easy To Use Pre-configured Projects * Can be (and is) Used in Commercial Applications * Massive User Community * Free Forum Support * Optional Commercial Licensing/Support * Strict Coding Standard * Safety Critical Version Available * Tiny Footprint

FreeRTOS™ is a market leading RTOS from Real Time Engineers Ltd. that supports 31 architectures and receives 77500 downloads a year. It is professionally...
Questions?
Please Provide Your Feedback...

- Please utilize the ‘Guidebook’ application to leave feedback

or

- Ask me for the paper feedback form for you to use...
Thanks For Attending!

The FreeRTOS Real-time Kernel is a highly successful operating system for use in embedded microcontrollers. Downloaded more than 150,000 times in two years, it is a professional grade kernel which can be used in commercial applications completely free of charge.

This book is a concise, step by step, hands on tutorial guide that describes both general multitasking concepts and FreeRTOS specifics. It presents and explains numerous examples that are written using the FreeRTOS API.

The RX600 edition includes eighteen examples written specifically for the RX62N microcontroller using the Renesas compiler and the free High-performance Embedded Workshop IDE. The projects target the Renesas RX62N Starter Kit (RSK), but can be adapted easily for use on alternative RX62N based development hardware, such as the Renesas RX62N Demonstration Kit (RDK).

A complete API reference manual to complement this tutorial book can be obtained from http://www.FreeRTOS.org/Documentation.

About the author:

Richard Barry graduated with 1st Class Honors in Computing for Real Time Systems. He’s been directly involved in the start up of several companies, primarily working in the industrial automation and aerospace and simulation markets. Barry is currently a director of Real Time Engineers Ltd, owners of FreeRTOS and Head of Innovation at Wristchain High Integrity Systems.

Using The FreeRTOS Real Time Kernel

Renesas RX600 Edition

Richard Barry