How to Measure RTOS Performance

Colin Walls

colin_walls@mentor.com

with acknowledgement to Faheem Sheikh & Dan Driscoll

mentor.com/embedded
Agenda

Introduction
RTOS Metrics
Memory Footprint
Interrupt Latency
Scheduling Latency
Timing Kernel Services
Conclusions
Agenda

Introduction
RTOS Metrics
Memory Footprint
Interrupt Latency
Scheduling Latency
Timing Kernel Services
Conclusions
Introduction – Why?

Desktop computers
- Infinite CPU power
- Infinite memory
- Costs nothing

Embedded systems
- Enough CPU power – just
- Adequate memory – none to spare
- Power consumption an issue
- Cost normally critical
Introduction – Choosing an RTOS

- RTOS solutions
  - Proprietary [in-house; home brew]
  - Commercial
    - At least 200 product available
  - Open source

- Asking the right questions is important

- Understanding the answers is critical
  - Pitfalls with misinterpretation
Agenda

Introduction

RTOS Metrics

Memory Footprint

Interrupt Latency

Scheduling Latency

Timing Kernel Services

Conclusions
RTOS Metrics

- Three common categories:
  - Memory footprint
    - Program and data
  - Latency
    - Interrupt and scheduling
  - Services performance

- No real standardization
  - Embedded Microprocessor Benchmark Consortium (EEMBC) not widely adopted
    - Oriented towards CPU benchmarking
Agenda

Introduction
RTOS Metrics

Memory Footprint
Interrupt Latency
Scheduling Latency
Timing Kernel Services
Conclusions
Memory Footprint

- RAM and ROM requirements of RTOS
  - On a specific platform

- ROM size:
  - Kernel code
  - Read-only data
  - Runtime library code
  - Maybe in flash; possibly copied to RAM

- RAM size:
  - Kernel data structures
  - Global variables
  - May need accommodate "ROM"
Memory Footprint – Dependencies

- Key factors affect the footprint calculation

- CPU architecture
  - Huge effect on number of instructions

- Software configuration
  - Which kernel components are included
  - Scalability ...

- Compiler optimization
  - Reduces code size, but may adversely affect performance
Memory Footprint – Scalability

RTOS

Service #1
Service #2
Service #3
Service #4
... Service #271

Application

... rtos_call_1();
... rtos_call_3();
... rtos_call_157();
...

Memory

Application Code

RTOS core
Service #1
Service #3
Service #157
Memory Footprint – Measurement

- Making measurement need not be difficult
- Two key methods:
  - Memory MAP file
    - Generated by standard linkers
    - Quality and detail varies between tools
  - Specific tool
    - Shows footprint information for selected executable image
    - Example: objdump
Memory Footprint – Importance

- Limited memory availability
  - Small on-chip memory
  - No external memory option
  - Application code is priority

- Larger systems
  - Kernel performance a priority
  - Place in on-chip memory
  - Lock into cache

- Using a bootloader
  - Non-volatile memory and RAM space used
Memory Footprint – Pitfalls

- Vendor data can be readily misinterpreted

- Look at minimum configuration definition
  - It may be a tiny, impractical subset

- Runtime library functions are often not included

- RAM/ROM sizes should have a min/max range
  - RAM is likely to be application dependent
  - ROM driven by kernel configuration
    - Need minimum “useable” size
    - Also maximum measure with all services
  - Scalability variable – may be different kernel versions
Memory Footprint – Example

- Nucleus RTOS kernel is fully scalable
- Example: ARM Cortex A8 in ARM mode
- Built with Mentor Sourcery CodeBench toolchain
- Full optimization for size

- ROM = 12-30 Kbytes
- RAM = 500 bytes
Memory Footprint – Example

- Min has essential kernel services [dynamic memory, threads, semaphores, events, queues]
  - Runtime library excluded
- Max includes all kernel services

- Compiling for Thumb-2 mode reduces ROM by 35%
  - So Nucleus kernel can use 7.8 Kbytes on a Cortex-M based controller
Agenda

Introduction
RTOS Metrics
Memory Footprint

**Interrupt Latency**
Scheduling Latency
Timing Kernel Services
Conclusions
Interrupt Latency

Definitions of interrupt latency
Interrupt Latency – Definition

- Different definitions

- System: Time between interrupt assertion and the instant an observable response happens

- OS: Duration of when the CPU was interrupted until the start of the corresponding interrupt service routine (ISR)
  - This is really OS overhead
  - Many vendors refer to this as the latency
  - Hence often report zero latency
Interrupt Latency – Measurement

Interrupt response is the sum of two distinct times:

\[ \tau_{IL} = \tau_H + \tau_{OS} \]

where:

- \( \tau_H \) is the hardware dependent time
  - depends on the interrupt controller on the board as well as the type of the interrupt
- \( \tau_{OS} \) is the OS induced overhead
  - Best and worst case scenarios
  - Worst when kernel disables interrupts
Interrupt Latency – Measurement

- Best approach is to record time between interrupts source and response
  - Use an oscilloscope
  - For example:
    - One GPIO pin can generate an interrupt
    - Another pin toggled at the start of the ISR
Interrupt Latency – Importance

- Specific types of designs rely on this metric:
  - Time critical
  - Fault tolerant

- If high I/O bandwidth is needed, measure the latency of a particular interrupt

- Most systems can tolerate interrupt latency of tens of microseconds
Interrupt Latency – Pitfalls

- Main danger is interpretation of published figures

### Hardware configuration
- Which platform?
- CPU speed?
- Cache configuration?
- Timer frequency?
- Interrupt controller type?

### OS configuration
- Where is code running from? [Flash, SDRAM ...]
- Which interrupt used?
- Code optimized for speed?
- Is metric best or average case?
Interrupt Latency – Example

- Nucleus RTOS
  - ARM Cortex A8
  - 600MHz
  - Running from SRAM

- Average interrupt latency of less than 0.5 microseconds
Agenda

Introduction
RTOS Metrics
Memory Footprint
Interrupt Latency

**Scheduling Latency**

Timing Kernel Services
Conclusions
Scheduling Latency – Definition

- Performance of RTOS thread scheduler
- Very wide variation in measurement technique and interpretation
- Two distinct related quantities:
  - Context switch time
  - Scheduling overhead
Scheduling Latency – Definition

Context switch time

Event → OS → Task A → OS → Response → Task B → OS → Event

Context Switch Latency
Scheduling Latency – Definition

Scheduling overhead

Event → OS → Task B → OS → Response → Scheduling Overhead
Scheduling Latency – Measurement

Scheduling latency is the maximum of two times:

\[ T_{SL} = \text{MAX}(T_{SO}, T_{CS}) \]

where:

- \( T_{SO} \) is the scheduling overhead
  - End of ISR to start of task schedule
- \( T_{CS} \) is the time taken to save and restore thread context
Scheduling Latency – Importance

- Most systems that have stringent interrupt latency demands also need low scheduling latency
- Broadly, systems that are:
  - Time critical
  - Fault tolerant
Scheduling Latency – Pitfalls

- Ignoring initial system state
  - If system is idle, there is no time taken saving context

Hardware configuration
- Which platform?
- CPU speed?
- Cache configuration?
- Timer frequency?
- Interrupt controller type?

OS configuration
- Where is code running from? [Flash, SDRAM ...]
- Which interrupt used?
- Code optimized for speed?
- Is metric best or average case?
Scheduling Latency – Example

- Nucleus RTOS
  - ARM Cortex A8
  - 600MHz
  - Running from SRAM

- Scheduling latency is 1.3 microseconds
Agenda

Introduction
RTOS Metrics
Memory Footprint
Interrupt Latency
Scheduling Latency

**Timing Kernel Services**

Conclusions
Timing Kernel Services

- RTOS may have a great many API calls
- Timing of keys ones may be of interest
  - Focus on frequently used API calls
- Four key categories:
  - Threading services
  - Synchronization services
  - Inter-process communication services
  - Memory services
Timing Kernel Services

Threading Services

- Control of fundamental kernel functionality:
  - Create thread
  - Start thread
  - Resume thread
  - Stop thread
- Many multi-taking applications make heavy use of these calls
Timing Kernel Services

Synchronization Services

- Services to synchronize between contexts, like an ISR and a thread
- Also protection of critical code sections from concurrent access
  - e.g. Semaphore may be used by Ethernet ISR to write data into shared memory that is also used by a task
- Timing is important if the application has numerous shared resources or peripherals
Timing Kernel Services

Inter-process Communication Services

- Services to share data between multiple threads
- Examples:
  - FIFOs
  - Queues
  - Mailboxes
  - Event flags
- Many multi-taking applications make heavy use of this type of service
Timing Kernel Services

Memory Services

- In a multi-threaded context, kernel is used to manage dynamic memory
- Allocation and de-allocation times are important
  - Applications with large data throughput are particularly sensitive
Timing Kernel Services – Pitfalls

- Hardware configuration
  - Which platform?
  - CPU speed?
  - Cache configuration?

- OS configuration
  - Where is code running from? [Flash, SDRAM ...]
  - Code optimized for speed?
  - Is metric best or average case?
  - Is kernel configured for reduced error checking?
## Timing Kernel Services – Example

<table>
<thead>
<tr>
<th>Nucleus RTOS Kernel Service</th>
<th>Time in µS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task resumption</td>
<td>0.3</td>
</tr>
<tr>
<td>Task suspension</td>
<td>0.3</td>
</tr>
<tr>
<td>Obtaining a semaphore</td>
<td>0.5</td>
</tr>
<tr>
<td>Set an event</td>
<td>0.4</td>
</tr>
<tr>
<td>Send message to queue</td>
<td>0.9</td>
</tr>
<tr>
<td>Allocate memory</td>
<td>0.2</td>
</tr>
<tr>
<td>De-allocate memory</td>
<td>0.7</td>
</tr>
<tr>
<td>Allocate partition</td>
<td>0.4</td>
</tr>
</tbody>
</table>
Agenda

Introduction
RTOS Metrics
Memory Footprint
Interrupt Latency
Scheduling Latency
Timing Kernel Services

Conclusions
Conclusions

- RTOS performance data provided by vendors can be useful
- It can also be misleading if it is misinterpreted
- Need a thorough understanding of:
  - Measurement techniques
  - Terminology
  - Trade-offs
to conduct fair comparison
- Recommendation is to rely on holistic or application-oriented measurements
Thank you

Colin Walls
colin_walls@mentor.com
http://blogs.mentor.com/colinwalls