Real-Time Linux on Embedded Multi-Core Processors

Andreas Ehmanns,
Technical Advisor Embedded Software Systems
MBDA Deutschland GmbH
Content of Presentation

Single-Core + ? Multi-Core

RT

RT

Chip

Chip
Agenda

- Motivation
- Linux and Real Time
- Latency Measurements
- From Single-Core to Multi-Core
- Effects on Multi-Core Systems
- Hardware Architecture(s)
- Summary
• Combination of Vanilla Linux kernel and RT-Preempt patch is well established on embedded single-core systems.
• Semiconductor industry is driving an evolution towards multi-core processors even in the embedded area.

How can the combination of vanilla linux kernel and RT-Preempt patch be migrated to multi-core hardware?

• This presentation will outline one possible way to migrate from a single-core to a multi-core processor system.
Basis for migration:
Single-core system, 6U VME PowerPC card, 74xx

• Hardware becomes more and more obsolete
• Can (several) boards be replaced by new multi-core based boards?
• Advantages (depending on CPU type):
  • More computing power/board
  • Less power consumption/board
  • Less heat dissipation/board
• Disadvantages:
  • Shared resources (e.g. Caches, RAM, I/Os, …)
  • Possible interferences
Software Setup

- Vanilla Linux Kernel 4.4.3 from kernel.org
- RT-Preempt patch 4.4.3-rt9
- General kernel configuration: Full RT, Tickless System, High Resolution Timer, ...
- Disabled all features with negative impact on realtime behaviour, e.g. power management, dyn. frequency scaling, hotplugging, ...
- Installed tool „cyclicertest“ (part of rt-tests package, v0.89) for latency measurements

Note: Scope of presentation is the migration to multi-core and NOT a discussion of preempt patch or its tools.
• … measures latency of response to a stimulus.
• … sleeps for a defined time
• … measures actual time when woken up
• … calculates difference of actual and expected time

```c
while (!shutdown) {
    clock_nanosleep(&next);
    clock_gettime(&now);
    diff = calcdiff(now, next);
    ...
    next += interval
}```
Effects of RT-Preempt patch

- Latency measurement with „cyclictest“
- High load on Ethernet and RapidIO interface
- Long-Term measurement
- Unpatched version shows outliers up to 5ms!
- Note: Logarithmic scale on y-axis
- Curve shape is hardware dependant
Using Freescale/NXP QorIQ evaluation boards T2080RDB und T4240RDB

T2080RDB

T4240RDB
### Multi-Core Hardware and Software

<table>
<thead>
<tr>
<th></th>
<th>T2080RDB</th>
<th>T4240RDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>QorIQ T2080</td>
<td>QorIQ T4240</td>
</tr>
<tr>
<td>Core</td>
<td>e6500</td>
<td>e6500</td>
</tr>
<tr>
<td># cores</td>
<td>8</td>
<td>24</td>
</tr>
<tr>
<td>clock</td>
<td>1.8GHz</td>
<td>1.8GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>4GB</td>
<td>12GB</td>
</tr>
</tbody>
</table>

- Step of more than 3 PowerPC Generations from 74xx (G4)
- Boot images built with NXP SDK 2.0, based on Yocto
- Kernel 4.1.8, RT-Preempt patch 4.1.8-rt8
First Attempt

- Let Linux kernel handle all cores (SMP)
- Change kernel configuration to new CPU and I/O hardware.
- No further modifications
- „cyclictest“ to start one thread on each core (bound with affinity)
- Note: Outliers will always be marked in diagrams.

Let’s see what „cyclictest“ will report …
• Idle (no load)
• Curve shape can change from core(s) to core(s) each new run
• Are we done?

Additional investigation necessary
First Results T2080RDB - Load

- Under Load (CPU, Ethernet, Serial)
- Peaks can change from core(s) to core(s) each new run
- Different range on x-axis (250 µs instead of 100 µs)
• Similar behaviour than T2080 but 24 cores/graphs
  
  T2080 will be discussed only on the next slides
Scheduler decides on which core tasks run.
Tasks can be migrated dynamically.

Bind all tasks to one core, e.g. core 0

```bash
# Migrate all possible tasks to core 0
for PROCESS in `$(ls /proc);` do
    if [ -x "/proc/$PROCESS/task/" ]; then
        taskset -ac 0 $PROCESS
    fi
done
```
Effects of Task Migration

- Only two cores with latencies > 25µs
- Results vary from run to run.
- Still some tasks running on non-0 core
root@t2080rdb:~# cat /proc/interrupts

CPU0  CPU1  CPU2  CPU3  CPU4  CPU5  CPU6  CPU7
36:  2    31     2     3     3    46     3  6708  OpenPIC  36 Level  serial

Interrupts are handled by different cores

# Migrate IRQs to core 0
for IRQ in $(ls /proc/irq); do
    if [[ -x "/proc/irq/\${IRQ}\" && \${IRQ} != "0" ]]; then
        echo 1 > /proc/irq/\${IRQ}/smp_affinity
    fi
done

# Set default affinity for new IRQs
echo 1 > /proc/irq/default_smp_affinity
Effects of IRQ and Task Migration

- Less entries in region 150-250µs
- Still two cores with entries >25µs
- Core(s) vary from run to run.
• **Isolate CPUs/cores from the kernel scheduler.**

• **Use boot parameter isolcpus**
  - isolcpus= cpu_number [, cpu_number ,...]
  - Remove the specified CPUs, from the general kernel SMP balancing and scheduler algorithms.
  - Use “taskset” to assign applications to cores.

• **Idea:**
  - Reserve core 1-7 for user applications
  - Let core 0 handle all kernel and OS load
Effects of „isolcpus“

- Core 0 handles kernel and OS
- Core 1-7 reserved for user application.
- No outliers
- Max Latency for core 1-7: 13µs

Latency Measurement on T2080RDE

<table>
<thead>
<tr>
<th>Core</th>
<th>Min Latencies</th>
<th>Avg Latencies</th>
<th>Max Latencies</th>
<th>Histogram Overflows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 0</td>
<td>00005</td>
<td>00005</td>
<td>00005</td>
<td>00000</td>
</tr>
<tr>
<td>Core 1</td>
<td>00005</td>
<td>00005</td>
<td>00005</td>
<td>00000</td>
</tr>
<tr>
<td>Core 2</td>
<td>00005</td>
<td>00005</td>
<td>00005</td>
<td>00000</td>
</tr>
<tr>
<td>Core 3</td>
<td>00005</td>
<td>00005</td>
<td>00005</td>
<td>00000</td>
</tr>
<tr>
<td>Core 4</td>
<td>00005</td>
<td>00006</td>
<td>00005</td>
<td>00000</td>
</tr>
<tr>
<td>Core 5</td>
<td>00005</td>
<td>00005</td>
<td>00005</td>
<td>00000</td>
</tr>
<tr>
<td>Core 6</td>
<td>00005</td>
<td>00005</td>
<td>00005</td>
<td>00000</td>
</tr>
<tr>
<td>Core 7</td>
<td>00005</td>
<td>00005</td>
<td>00005</td>
<td>00000</td>
</tr>
</tbody>
</table>
Use a „Real Application“ for Testing

- Latency measurements are necessary
- But: Best test is your „Real Application“
  - Migration: Use existing code on new hardware
  - New project: Try to do a reference implementation of critical code (sections).

- Writing a reference implementation
  - Is your application/system able to run „in parallel“?
  - Be careful with time measurement
  - Take caching effects into account
  - Simulate or implement messaging if necessary
  - Does your application I/O?
  - Run long-term measurements to find unfrequent outliers
  - Check that applications executes as expected
  - …
• Let’s do a demonstration with an existing real time critical algorithm in 2 versions:
  • Pure C/C++ code
  • Usage of Altivec instructions (SIMD vector unit of PowerPC family)

• Algorithm: Up to 95% of code can run in parallel.
• Usage of big Lookup-Tables reduces caching effects
• Simulates storage of data in hardware
• Main code parts are parallelized.
• Factor of 9.9 between 1 and 8 cores
• Factor of 1.9 between 4 and 8 cores
• Speedup can be described by Amdahl’s law.
• Good perf. improvement up to 5 cores (factor 4.2).
• No real speedup for 6 and 7 cores.
• Reduced performance with 8 cores.
• Faktor of 1.03 between 5 and 8 cores.

What is the root cause?
- Timing values shown before are average
- 1-4 cores: < 2% jitter
- 5-8 cores: 50-100% jitter
• `/proc/cpuinfo` and „top“ report 8 cores.

• T2080: 4 physical cores, 8 virtual cores

• „Dual-Threaded Cores“

• Most hardware elements of a physical core are available twice.

• One Altivec unit per physical core.

• Only 4 Altivec units on T2080.

Taken from „QorIQ T2080 Family Reference Manual“, Document Number: T2080RM, Rev. 1, 05/2015, Copyright NXP Semiconductors
• Check OS core numbering scheme
• Pin only one Altivec applications to one physical core
• E.g. use core 0, 2, 4, 6 instead of 0, 1, 2, 3
More things to know:

- One L1 cache per physical core
- Two „Core Threads“ share L1
- One L2 cache for all cores
- CoreNet is the main interconnect of CPU, RAM, I/O, ...
- QMan, BMan, FMan for higher throughput and less interferences.
- Check for possible I/O interferences!

Taken from „QorIQ T2080 Family Reference Manual“, Document Number: T2080RM, Rev. 1, 05/2015, Copyright NXP Semiconductors
• Similar to T2080 but:
  • T4240 contains 3 T2080 units.
  • 3 L2 Caches, one per each T2080 unit.
• L2 interferences can be seen with test algo:

<table>
<thead>
<tr>
<th>Run on core</th>
<th>Time in µs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 2</td>
<td>1220</td>
</tr>
<tr>
<td>0, 8</td>
<td>774</td>
</tr>
</tbody>
</table>

Taken from „QorIQ T4240 Reference Manual“, Document Number: T4240RM Rev. 2, 06/2015, Copyright NXP Semiconductors
Cache Interferences – Basic Example

- 1, 2 and 4 Threads: Distributed to physical cores
- 8 Threads: Using all virtual cores
- Caches sizes are visible
- L2 cache is shared between all cores
- L1 cache is shared inside physical core

![Ramspeed on T2080RDB](chart)

- Throughput in MB/s
- Block Size in KB
- L1
- L2
- RAM
• Cache and RAM interferences are more complicated for T4240
• CoreNet internal architecture unknown
• Same applies to QMan (Queue Manager), FMan (Frame Manager) and BMan (Buffer Manager)
• T2080 and T4240 provide numerous (high-speed) interfaces (e.g. SATA, PCIe, sRIO, SerDes, SPI, I²C, SD/MMC, UART, …)
• Interferences between them and/or Cache/RAM might occur.
• Consider DMA transactions.

Depending on application and its I/O, possible interferences should be considered and investigated profoundly.
• A vanilla Linux kernel with applied RT-Preempt patch can be used on multi-core systems with realtime requirements.

• Linux provides configuration parameters and tools to adapt system and core behaviour to own needs.

• System and software engineers need a good knowledge of processor hardware architecture.

• Depending on processor architecture, deployment of application to dedicated cores has to be considered carefully (including processing power, caches, I/O interferences).
• RT Linux on (embedded) multi-core systems is not something „magic“.

• Shown setup is just an example for a dedicated combination of hardware and software.

• Your system may look different and the solution may be different.

• Use the content of this presentation as a suggestion …

Give it a try. It‘s fun …