Power Management
Technologies for Networking Applications

FTF-NET-F0035

Li Yang | Linux Software Architect

APR. 2014
Agenda

- QorIQ Power Management features
- Linux PM solutions
- Benchmarking of the PM features
- PM feature and use cases
QorIQ Power Management features
Power Consumption

• CMOS Energy Consumption
  - Dynamic Energy Consumption
  - Static Energy Consumption

\[ E = \int_{0}^{t} \left( CV_{dd}^{2} f_{c} + V_{dd} I_{lkg} \right) \, dt \]
# Reduce Power Consumption

## Dynamic Power
- Reduce voltage
- Reduce frequency
- Clock gating
- Reduce active-load capacitance

## Static Power
- Reduce voltage
- Less leaky transistors (better process)
- Reduce working temperature
- Power gating
What’s Power Management?

• It is **NOT** about reducing *maximum* power consumption
• It is about **matching** the runtime *workload* requirements with runtime *performance/capacity*
• **Turn off** everything else as much as possible
• **PM** is all about providing ways to do so
# Core/Cluster states

<table>
<thead>
<tr>
<th>States</th>
<th>PH00</th>
<th>PH10</th>
<th>PW10</th>
<th>PH15</th>
<th>PH20</th>
<th>PW20</th>
<th>PH30</th>
<th>PCL10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old names</td>
<td>Run</td>
<td>Doze</td>
<td>Wait</td>
<td>Nap</td>
<td>Drowsy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instruction fetch</td>
<td>On</td>
<td>Stopped</td>
<td>Stopped</td>
<td>Stopped</td>
<td>Stopped</td>
<td>Stopped</td>
<td>Stopped</td>
<td>Stopped</td>
</tr>
<tr>
<td>Core clock</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td>Core Voltage</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Core L1 Cache</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>Snoop off, SW inval.</td>
<td>Snoop off, SW inval.</td>
<td>Snoop off, HW inval.</td>
<td>Snoop off, SW inval.</td>
<td>Snoop off, SW inval.</td>
</tr>
<tr>
<td>Cluster Clock</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>Cluster L2 Cache</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>Snoop off, SW inval.</td>
</tr>
</tbody>
</table>

New to e6500
# Device/system states

<table>
<thead>
<tr>
<th>SoC</th>
<th>LPM00</th>
<th>LPM10</th>
<th>LPM20</th>
<th>LPM30</th>
<th>LPM35</th>
<th>LPM40</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Run</td>
<td>Sleep</td>
<td>Deep Sleep</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core state</td>
<td>PH00</td>
<td>PH00+</td>
<td>PH20</td>
<td>PH20+</td>
<td>PH30</td>
<td>PH30</td>
</tr>
<tr>
<td>Core Timebase</td>
<td>On</td>
<td>On</td>
<td>Stopped</td>
<td>Stopped</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td>On-chip devices</td>
<td>On</td>
<td>On</td>
<td>Clock gated</td>
<td>Clock gated</td>
<td>Power off*</td>
<td>Off</td>
</tr>
<tr>
<td>Core Voltage</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>Off</td>
<td>Low</td>
</tr>
<tr>
<td>DDR controller</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>Power off*</td>
<td>Off</td>
</tr>
<tr>
<td>Exception devices</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>DDR memory</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>Self-refresh</td>
<td>unknown</td>
</tr>
<tr>
<td>On-board devices</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>Power off*</td>
<td>unknown</td>
</tr>
</tbody>
</table>
# T104x Deep Sleep

## Hardware
- SoC
  - LPM35
- Board
  - DDR self-refresh
  - Power rails control (FPGA/CPLD)

## Software
- RCW and PBI
- Secure Bootrom
- U-boot
- Linux kernel
- Device drivers
- User applications
Other features

- Dynamic Frequency Scaling
- Altivec Drowsy
- Cascade PM
- Auto-response
- Thermal diode/TMU
Linux PM solutions
Linux PM frameworks

User Space

PM control utilities(DPM)/cmdline

Sysfs

Linux device model
- pm_ops
- runtime_pm_ops
- wakeup source
- Clock
- Power domain
- Qos

PM core
- suspend
- hibernation
- Runtime PM
- Autosleep
- Wake lock
- Pm_qos
- devfreq

CPU mgmt
- CPU freq
- CPU idle
- CPU hotplug
- Power aware sched

Misc
- hwmon
- Thermal

Kernel Space
CPU Idle

- Dynamically put core in lightweight low power state, wakeup quickly
- `cpu_idle()` -- arch/powerpc/kernel/idle.c
  - A never-ending task on each CPU
  - Runs when there is nothing else to run
  - Calls `ppc_md.power_save()` when possible
  - Can be turned off by adding "powersave=off" to kernel bootcmd

- `e500_idle()` -- arch/powerpc/kernel/idle_e500.S
  - The `ppc_md.power_save()` for e500 platforms
  - Calls `wait` instruction for e500mc or later cores
  - Wait instruction enters PW10 or PW20 state depending on sysfs configure for e6500 -- `/sys/devices/system/cpu/cpuX/pw20_state`
  - Enters Doze or Nap state on old e500 platforms, use Nap state if configured through procfs -- `/proc/sys/kernel/powersave_nap`
  - Exits when there is an interrupt to the core
Multi-level CPU idle support

• New CPU idle framework to support multiple CPU idle states

• Governors
  – Ladder - Enter deeper state step by step
  – Menu – Calculate and select a state considering
    ▪ Energy break even point
    ▪ Performance impact
    ▪ Latency tolerance

• In plan to be supported on e6500
  – PW10
  – PW20
  – PCL10
CPU Hotplug

- Removes or plugs a CPU to the Linux system at runtime
- Build-time option:
  - CONFIG_HOTPLUG_CPU
- Runtime control through sysfs: /sys/devices/system/cpu/*
  - Unplugging
    
  ```
  #echo 0 > /sys/devices/system/cpu/cpuX/online
  ```
  - Plugging
    
  ```
  #echo 1 > /sys/devices/system/cpu/cpuX/online
  ```
CPU Hotplug Unplugging process – Common Part

- Send CPU_DOWN_PREPARE notification to all in-kernel interested modules
- Migrate all processes to other CPU(s)
- Migrate all interrupts to a new CPU
- Migrate timers/bottom half/tasklets to a new CPU
- Call an architecture specific routine __cpu_disable()
- Finally send CPU_DEAD notification to all in-kernel modules
CPU Hotplug Unplugging Process – Freescale Part

1. \texttt{\_\_cpu\_disable()} calls platform specific \texttt{cpu\_disable()}
   a) Clear bit in the online cpu mask
   b) Migrate IRQ

2. On the cpu to be unplugged
   a) \texttt{cpu\_idle()} will find its offload in \texttt{cpu\_mask}
   b) Calls platform specific \texttt{ppc\_md\_cpu\_die()} = \texttt{smp\_85xx\_mach\_cpu\_die()}
   c) Flush and disable L1
   d) Actually set the cpu to PH20 state for e6500 and Nap for earlier cores
CPU hotplug compatibility

• App/driver need to take action if depending on specific core
• Kernel: add to the notification chain using register_cpu_notifier()
  - CPU_ONLINE
  - CPU_UP_PREPARE
  - CPU_DOWN_PREPARE
  - CPU_DEAD
• User space:
  - Through uevent to get notifications
    ▪ udevd
    ▪ /sbin/hotplug
  - Process/thread automatically update affinity if original affinity setting can not be met
System Suspend

• Put system into low-power states on user command

• Suspend states:
  - standby – Sleep hardware state
  - mem – Deep sleep hardware state
  - disk - Suspend to disk/Hibernation

• Run-time control through sysfs
  
  #echo state > /sys/power/state

• Wakeup through interrupts
  - Different wakeup sources among SoCs
Wakeup sources for sleep/deep sleep

- MPIC timer
- External interrupt
- Ethernet magic packet
- Ethernet user defined packet
- USB plug/unplug events
- SD card plug/unplug events
- GPIO events
System Suspend process – Common Part

1. Sync file systems
2. Prepare
   a) Prepare console
   b) Send PM_SUSPEND_PREPARE notification
   c) Disable user-mode helper
   d) Freeze processes and kernel thread (freezable only, common kthreads are non-freezable)
3. Suspend console
4. Device PM suspend start
   1. Call .prepare() callbacks of device drivers (device_add order)
   2. Call suspend callbacks of device drivers with interrupt order (reverse order)
5. Platform-specific suspend prepare() operation
6. Device PM suspend end
   1. Call .late() callbacks (reverse order)
   2. Disable all interrupts except ones with IRQF_NO_SUSPEND in IRQ controller
   3. Call noirq() callbacks of device drivers with interrupt disabled (reverse order)
7. Disable all CPUs other than the booting CPU
8. Arch disable all interrupts (Clear EE)
9. Call suspend callbacks for system core devices (such as ktime)
10. Call platform-specific suspend_ops->enter()
System Suspend process – order of callbacks

- dpm_list created with device_add() order during bootup
- What’s the device_add() order?
  - Platform devices (added by device tree probe, order same as defined in device tree)
  - Bus devices (added by bus probe, order depends on probe timing)
    - Initcall level of the probing
    - Makefile order in the same level
- Orders of callbacks being called
  - Prepare() - dpm_list order
  - Suspend() - reverse dpm_list order
  - Suspend_late() - reverse dpm_list order
  - Suspend_noirq() - reverse dpm_list order
  - Resume_noirq() - dpm_list order
  - Resume_early() - dpm_list order
  - Resume() – dpm_list order
  - Complete() – reverse dpm_list order
System Suspend process – platform part: sleep

- E500: arch/powerpc/sysdev/fsl_pmc.c
  - pmc_suspend_enter()
- E500mc+: arch/powerpc/platforms/85xx/qoriq_pm.c
  - qoriq_suspend_enter()

1. Flush L1 dcache (e500/e500mc/e5500)
2. Flush L2 cache(e500mc/e5500)
3. Setting related bit in POWMGTCSR register
4. Hardware enter SLEEP state
5. Continue running when wakeup event is received
6. Back to common resume procedures
System Suspend process – PMC deep sleep

mpc85xx_enter_deep_sleep() - arch/powerpc/platforms/85xx/sleep.S

1. Enable SPE/FPU
2. Disable local interrupt
3. Backup important SPR registers in memory
4. Save current BPTR
5. Flush and disable L2 and L1 cache
6. Set resume entry point mpc85xx_deep_resume() to BPTR(boot page translate register)
7. Disable decrementer
8. Set deep sleep bit in power management unit
System Resume process – PMC deep sleep

1. Hardware wait for POWER_OK signal or voltage ramp up timer expire
2. Core reboot from page mapped by BPTR -- mpc85xx_deep_resume()
3. Enable L2 cache
4. Restore original BPTR and jump to normal memory address
5. Restore TLB1
6. Restore SPR registers
7. Return to common resume code (opposite to suspend sequence)
T104x Deep Sleep Software Flow chart

- **Active**
  - suspend
  - CPU hotplug
  - Driver suspend

- **Linux PM core**
  - suspend
  - resume
  - CPU hotplug
  - Driver resume

- **Platform support (RCPM driver)**
  - Cache ops
  - EPU, NPC, ...
  - Context switch
  - Code in SRAM
  - Backup/Restore states

- **Auto-response mode**
  - Deep sleep
  - Warm reset

- **U-boot**
  - DDR restore
  - HW init

- **AR driver**

- **AR Traffic**

- **AR Response**

- **Wakeup**
System Suspend process – T1040 deep sleep

1. Mask interrupts in RCPM
2. Backup content of DDR which will be damaged in DDR training
3. Setup resume pointer to SCFG_SPARERCR2 register
4. Enable warm reset in SCFG_DPSLPCR register
5. Setup state machine in the EPU
6. Backup core states into memory
7. Flush CPC cache
8. Setup environment for running code without RAM
9. Notify FPGA/CPLD to be ready for deep sleep entrance
10. Load the code into icache and start running the code in cache
11. Put DDR into self-refresh
12. Trigger state machine to take the SoC into LPM35 state
13. SoC deassert POWER_EN to notify the FPGA/CPLD to shutdown core and most of I/O power rails
14. Deep sleep is entered
System Resume process – T1040 deep sleep

1. EPU detects wakeup event, assert POWER_EN for the board to turn on power rails
2. Wait POWER_OK to be asserted by the board
3. Issues device warm reset
4. PBL loads RCW and PBI
5. Secureboot authentication is bypassed
6. Start running u-boot, and u-boot finds out this is deep sleep restore by checking CRSTSR[WDRFR]
7. Re-initialize DDR controller with by-pass mode
8. Bring DDR out of self-refresh
9. Restore DDR content damaged by DDR training
10. Get the kernel re-entry point from SCFG_SPARECR2 and jump to it
11. Setup non-booting CPUs and MMU
12. Restore core states from memory
13. Disable warm reset
14. Cleanup EPU state machine
CPUfreq

• Standard Linux interface for Dynamic Frequency Scaling
• Get available frequencies of cpuX
  - `# Cat /sys/devices/system/cpu/cpuX/cpufreq/scaling_available_frequencies`
  1199999 599999 299999 799999 399999 199999 1066666 533333 266666
• Change the core frequency to 533 MHz
  - `# echo 533333 > /sys/devices/system/cpu/cpuX/cpufreq/scaling_setspeed`
• Verify the frequency setting
  - `# cat /sys/devices/system/cpu/cpuX/cpufreq/scaling_cur_freq`
    533333
CPUfreq governors

- **Ondemand**
  - Measure the CPU load in sampling periods
  - Increase the frequency to maximum frequency when load is above a configurable threshold
  - Decrease the frequency to lowest frequency when the load is a configurable threshold

- **Conservative**
  - Increase/decrease the frequency gracefully (step by step)
Other PM Technologies (in plan)

- PM QoS
- Runtime PM
- Power aware scheduler
- Autosleep
PM QoS

• Provide interface to set application specific QoS expectation
  – CPU DMA latency
  – Network latency
  – Network throughput
  – Interface: /dev/[cpu_dma_latency, network_latency, network_throughput]

• Provide interface for device drivers
  – Maintain latency constrains of the device
  – Calls notifiers when aggregated value of constrains changed
Runtime PM

- More flexible for embedded systems from PC PM features
- Platform device drivers
  - Give drivers a single interface for clock and power domains.
  - Allows drivers to notify arch code of device idle state.
  - Provide drivers with PM callbacks for context save/restore

- Platform support
  - Track device driver idle state.
  - Let device idle state control power domains.
  - Adjust CPU sleep mode depending on idle state of devices.
Runtime PM APIs

• Used by users of the device
  – pm_runtime_enable(device);
  – pm_runtime_get_sync(device);
  – pm_runtime_put_sync(device);
  – pm_runtime_disable(device);

• Callbacks provided by the device driver
  – runtime_suspend(device);
  – runtime_resume(device);
Power Aware Scheduler

- Linux scheduling policy
- Sched_mc was removed from Linux kernel
- New Power Aware Scheduler framework is still being discussed in the Linux kernel community
- Need support of sched_domain
sched_domain

• **CPU topology**
  - SMT (e6500 dual threaded)
  - Multicore
  - CPU packages
  - NUMA
Autosleep

- User set intended sleep state in /sys/power/autosleep
- Application assert wakelock if it doesn’t want the system to go sleep
- System will automatically enter that system low power state when there is no wakelock being held
Benchmarking of the PM features
Goals of benchmarking

• Evaluate Linux PM features
  – Power consumption
  – Latency
  – Performance impact

• Optimization
  – Improve future hardware features
  – Improve software support

• Use case
  – Learn how to choose/use the features to best suit specific application
Understand side effects

- **Tradeoffs**
  - **Power** saving
  - **Latency** added
  - **Performance** impact
Latency analysis and optimizations

Hardware latency

Software latency

- enable_nonboot_cpus()
- dpm_resume_irqs()
- dpm_resume_early()
- dpm_resume()
- dpm_complete()
- resume_console()
- suspend_finish()
Power Measurement Solutions

- Internal measurement (using CPU to collect and process data)
- External measurement (using FPGA and server to collect and process data)
T4240 Core states benchmark

T4240QDS power consumption per core of hardware states

Per core consumption = 1V power supply consumption (total SoC power minus I/O power) / number of cores
T4240 PM benchmark overview

T4240 PM features test: power consumption, latency, performance

Rev1: 6GB DDR@800MHz, CCB@667MHz, CPU@1.67GHz
Rev2: 6GB DDR@933MHz, CCB@733MHz, CPU@1.67GHz

Power consumption (mW)

<table>
<thead>
<tr>
<th>State</th>
<th>T4240 Rev1</th>
<th>T4240 Rev2</th>
<th>Latency</th>
<th>CoreMarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully loaded</td>
<td>55001</td>
<td>34438</td>
<td>6.2</td>
<td>166891</td>
</tr>
<tr>
<td>Idle No PM feature</td>
<td>49257</td>
<td>31926</td>
<td>6.1</td>
<td>166681</td>
</tr>
<tr>
<td>Idle PW10</td>
<td>31978</td>
<td>15004</td>
<td>7.1</td>
<td>166787</td>
</tr>
<tr>
<td>Idle PW20</td>
<td>15153</td>
<td>11477</td>
<td>538</td>
<td>166891</td>
</tr>
<tr>
<td>Idle PW20, freq</td>
<td>11985</td>
<td>10101</td>
<td>6634</td>
<td>83006</td>
</tr>
<tr>
<td>Idle PW20, hotplug</td>
<td>15164</td>
<td>11488</td>
<td>811274</td>
<td>83587</td>
</tr>
<tr>
<td>Sleep</td>
<td>9906</td>
<td>9120</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

freq: half frequency using DFS
hotplug: unplug half cores
T1040 PM benchmark overview

<table>
<thead>
<tr>
<th>State</th>
<th>T1040QDS CPU</th>
<th>T1040QDS DDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully loaded</td>
<td>4875</td>
<td>550</td>
</tr>
<tr>
<td>Idle No PM feature</td>
<td>4682</td>
<td>546</td>
</tr>
<tr>
<td>Idle PW10</td>
<td>3248</td>
<td>544</td>
</tr>
<tr>
<td>Sleep</td>
<td>1153</td>
<td>157</td>
</tr>
<tr>
<td>Deep sleep</td>
<td>77</td>
<td>67</td>
</tr>
</tbody>
</table>

Power consumption (mW)
PM features and use cases
How to characterize your use case

- Load triggering mechanism
  - Application (predictable)
  - Direct User Input (predictable)
  - Traffic or service request from Network (non-predictable)

- Load profile
  - Sustained
  - Burst
  - Mixed

- Latency tolerance
  - Low latency (RT)
  - Medium latency (Networking)
  - High latency (Human interactive)
Who triggers the load

• Autonomous **application** running on the system
  - Characteristic: Load predictable
  - Example products: Monitoring (NVR), automatic control, consumer electronics

• **Human** through Human-machine interface
  - Example trigger: Buttons/switches, Graphic UI, Plug/unplug, Console
  - Characteristic: Load predictable
  - Example products: PC, consumer electronics, printing & imaging

• **Remote peer** connected through network
  - Example trigger: New packets, New transactions
  - Characteristic: Load Unpredictable
  - Example products: server, networking equipments
Know your load profile

- Load for a specific time
  - Fully Loaded
  - Partially Loaded
  - Partial Feature Used
  - Complete Idle
- Load for a time span
  - Burst load
  - Sustained Load
  - Mixed
Define your latency requirement

• **Definition:** the time between service request and react to that request

• **Response latency**
  – Time between request and starting to deal with the request or providing initial feedback

• **Speed up latency**
  – Time between request and the time when the performance/capacity has increased to the point with which the request can be service with QoS requirements met.
Static PM vs Dynamic PM

Use cases → Load Predictable

Load Predictable → Dynamic PM features

Load Predictable → Efficiency or Convenience

Efficiency or Convenience → Efficiency

Efficiency → Static PM features

Convenience → Dynamic PM features
Static PM features

- No PM
  - Partial Features
  - Fully Loaded
- CPU hotplug CPU freq
  - Partially Loaded
  - Complete Idle
- DEVDISR
  - Partial Features
  - Fully Loaded
- Sleep Deep Sleep
  - Complete Idle
  - Partially Loaded
Dynamic PM features

<table>
<thead>
<tr>
<th></th>
<th>I/O Load</th>
<th>Task Load</th>
<th>Mixed</th>
</tr>
</thead>
</table>
| **Burst Load** | Autosleep + WoL  
Autosleep + AR  
CPUidle  
CPUfreq(on-demand) | CPUidle  
CPUfreq(on-demand) | Autosleep + WoL  
Autosleep + AR  
CPUidle  
CPUfreq(on-demand) |
| **Sustained Load** | Cascade PM + CPUidle  
CPUfreq(conservative) | Power Aware scheduler  
CPUfreq(conservative) | Cascade PM + CPUidle  
Power Aware scheduler  
CPUfreq(conservative) |
| **Mixed** | Cascade PM + CPUidle  
CPUfreq(on-demand)  
CPUfreq(conservative) | CPUidle  
CPUfreq(on-demand)  
CPUfreq(conservative) | Cascade PM + CPUidle  
CPUfreq(on-demand)  
CPUfreq(conservative) |
Choose with response latency

<table>
<thead>
<tr>
<th>PM features</th>
<th>T4240 Latency (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No PM scheduling</td>
<td>6.200</td>
</tr>
<tr>
<td>CPU idle PW10</td>
<td>6.106</td>
</tr>
<tr>
<td>CPU idle PW20</td>
<td>7.098</td>
</tr>
<tr>
<td>sleep</td>
<td>811,274 (^1)</td>
</tr>
<tr>
<td>hibernation</td>
<td>12,601,000 (^2)</td>
</tr>
</tbody>
</table>

1. Variable with different device enabled
2. Variable with different harddisk and memory size
Choose with speed up latency

<table>
<thead>
<tr>
<th>PM features</th>
<th>T4240 Latency (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static CPUfreq</td>
<td>17 ~ 98</td>
</tr>
<tr>
<td>CPU hotplug</td>
<td>6,634</td>
</tr>
<tr>
<td>Dynamic CPUfreq ondemand</td>
<td>9,531 ~ 19,911</td>
</tr>
<tr>
<td>Dynamic CPUfreq conservative</td>
<td>9,531 ~ 209,911</td>
</tr>
</tbody>
</table>

1. Default settings, tunable
Introducing The QorIQ LS2 Family

Breakthrough, software-defined approach to advance the world’s new virtualized networks

New, high-performance architecture built with ease-of-use in mind
Groundbreaking, flexible architecture that abstracts hardware complexity and enables customers to focus their resources on innovation at the application level

Optimized for software-defined networking applications
Balanced integration of CPU performance with network I/O and C-programmable datapath acceleration that is right-sized (power/performance/cost) to deliver advanced SoC technology for the SDN era

Extending the industry’s broadest portfolio of 64-bit multicore SoCs
Built on the ARM® Cortex®-A57 architecture with integrated L2 switch enabling interconnect and peripherals to provide a complete system-on-chip solution
QorIQ LS2 Family

Key Features

SDN/NFV Switching

Data Center

Wireless Access

Unprecedented performance and ease of use for smarter, more capable networks

High performance cores with leading interconnect and memory bandwidth

- 8x ARM Cortex-A57 cores, 2.0GHz, 4MB L2 cache, w Neon SIMD
- 1MB L3 platform cache w/ECC
- 2x 64b DDR4 up to 2.4GT/s

A high performance datapath designed with software developers in mind

- New datapath hardware and abstracted acceleration that is called via standard Linux objects
- 40 Gbps Packet processing performance with 20Gbps acceleration (crypto, Pattern Match/RegEx, Data Compression)
- Management complex provides all init/setup/teardown tasks

Leading network I/O integration

- 8x1/10GbE + 8x1G, MACSec on up to 4x 1/10GbE
- Integrated L2 switching capability for cost savings
- 4 PCIe Gen3 controllers, 1 with SR-IOV support
- 2 x SATA 3.0, 2 x USB 3.0 with PHY
See the LS2 Family First in the Tech Lab!

4 new demos built on QorIQ LS2 processors:

- Performance Analysis Made Easy
- Leave the Packet Processing To Us
- Combining Ease of Use with Performance
- Tools for Every Step of Your Design