QorIQ™ Based Multicore LTE Layer 2 Software

Keith Shields
Freescale LTE System Enablement Overview: Software; Devices; AMC boards

Software Reference

LTE PHY Software
3GPP Rel 8 -LTE
Physical Processing

Transport Processing

LTE L2 Software
3GPP Rel 8 -LTE

X1 -S2

MAC/RLC

MSC8156
P20/30/40x

Enablement code developed to date is deployed in customer systems
Accelerating Customer Wireless QorIQ Development  
– It’s All About the Software

► Millions of lines of legacy LTE code need to be written in a parallel fashion to best utilize multi-core devices

► How to efficiently partition and run complex eNodeB functionality across a multi-core system
  • SMP vs AMP vs LWE
  • Scheduler, MAC, RLC, PDCP, IPSEC, GTP, SCTP……
  • Demonstrate Efficient and LTE SPECIFIC DPAA benefits

► Prove FSL QorIQ performance for LTE use-cases
  • Understand and remove system bottlenecks
  • Component benchmarks not sufficient, need full system functionality
LTE Overview
3G Evolution and Architecture

► Radio Side (LTE – Long Term Evolution)
  • Improvements in spectral efficiency, user throughput, latency
  • Simplification of the radio network
  • Efficient support of packet based services

► Network Side (SAE – System Architecture Evolution)
  • Improvement in latency, capacity, throughput
  • Simplification of the core network
  • Optimization for IP traffic and services
  • Simplified support and handover to non-3GPP access technologies
The main services and functions of PDCP for the **user** plane include:

- Header compression and decompression: ROHC
- Transfer of user data between RRC and RLC layers
- Ciphering

The main services and functions of PDCP for the **control** plane include:

- Ciphering and Integrity Protection
- Transfer of control plane data between RRC and RLC layers.
Protocol Stack Flow #1

Data from Network (01010101)

ROHC

Cipher (X%X$x)

PDPCP

RLC

RLC Header

Data

To MAC

Data

PDCP SN

Oct 1

Oct 2

Oct N-3

Oct N-2

Oct N-1

Oct N

MAC-I (cont.)

MAC-I (cont.)

MAC-I (cont.)

MAC-I (cont.)

MAC-I

...
Protocol Stack Flow #2

**RLC**

<table>
<thead>
<tr>
<th>D/C</th>
<th>RF</th>
<th>P</th>
<th>FI</th>
<th>E</th>
<th>SN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>LI₁</td>
<td></td>
<td>E</td>
<td>LI₂</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LI₁</td>
<td>E</td>
<td>LI₂</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>LIₖ₋₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIₖ₋₁</td>
<td>E</td>
<td>LIₖ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIₖ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MAC**

- Oct 1
- Oct 2
- Oct 3
- Oct 4
- Oct 5
- Oct [2+1.5*K-2]
- Oct [2+1.5*K-1]
- Oct [2+1.5*K]
- Oct [2+1.5*K+1]
- Oct N

**To Layer 1 Phy**
Freescale LTE Layer 2 Solution Overview
Software Support
## Freescale LTE Layer 2 Software Deliverables

<table>
<thead>
<tr>
<th>Category</th>
<th>Specification / Features</th>
</tr>
</thead>
</table>
| **RTOS Support**                      | • RTOS agnostic implementation  
• Example includes software ported to Linux user mode                                    |
| **API**                               | • Full software abstraction between data plane & control plane and data plane & scheduler through well-defined and documented APIs. |
| **Validation/Test**                   | • Software tested on unit, integration and system levels  
• Software test environment is part of the software delivery package.                         |
| **1. MAC - Medium Access Control Layer** | • Compatible to standard: 3GPP 36.321 (MAC) (V.8.3.0)  
• Includes sample downlink / uplink scheduler                                                 |
| **2. RLC - Radio Link Control**       | • Compatible to standard: 3GPP 36.322 (RLC) (V.8.3.0)                                        |
| **3. L2/L1 interface**                | • Implements an efficient L2/L1 interface designed for seamless integration with Freescale L1 solution (available today)  
• Easy L2/L1 interface, out-of-the-box experience through validated test cases.(over sRIO) |
| **4. Framework**                      | • Example integrated processing chain running under Linux (available today):  
  o Demonstrates integration of L2 modules  
  o Provides known development/test environment                                                |
| **5. PDCP - Packet Data Convergence Protocol** |  
  ▶ Header  
  ▶ ROHC & encryption  
  • Full header implementation (including HO) available now  
  • RoHC - Available through third party (available today)  
  • Provided with specific optimisations for Freescale architectures  
  • Air interface encryption (available today)  
  • Algorithm implemented on SEC Engine                                                                 |
| **6. IPSEC**                          | • Freescale Software (FastpathUTM), formerly Intoto                                         |
| **7. RRC**                            | • 3rd party or customer development                                                         |
- **Harness & Operating System**: System level test harness utilizing operating system timers and ethernet stack.
- **Scheduler**: Priority based round-robin scheduler.
- **Control**: API to facilitate configuration and execution of core modules.
- **PDCP**: Packet Data Convergence Protocol as specified by 3GPP 36.323 - utilises 3rd party ROHC implementation.
- **RLC**: Radio Link Control as specified by 3GPP 36.322.
- **MAC**: Medium Access Control as specified by 3GPP 36.321.
- **Common**: Generic functionality utilised by multiple modules e.g. linked list implementation, TTI event Timers etc.
- **IF1 interface**: Covers the protocol and LTE specific aspects of the DSP L1 interface.
LTE L2 Development Environment

Development Tools
- KDBG
- Insight
- DDD

Standard C Codebase
- Metrowerks
- GCC
- Binutils

Target Platforms
- Controller AMCs
  - 8548/8572/P2x/P4xAMCs
- DSP AMCs
  - 8144/8156AMCs
- Industry Standard Carriers
  - Pico/Micro TCA
- Proprietary Systems

Run-Time Environment
- 85xx
  - Linux
  - UBOOT
- x86
  - Linux
  - BIOS
- x86
  - Cygwin
  - BIOS
Freescale LTE Layer 2 QorIQ™
Solution Overview
Software Support
Multicore CPU's can permit a number of processing scenarios

We are benchmarking our LTE Layer 2 implementation to determine the optimal mix and fit for QorIQ architectures.
Multicore Simulation Environment

Applications

SMP/AMP Operating System

WIND RIVER
ENA
montavista
Green Hills Software

Optimized High-Speed Drivers

Hypervisor

virtutech
freescale

Functional

Cycle-Accurate

IDE (compiler / debugger / build tools)

Applications

SMP/AMP Operating System

WIND RIVER
ENA
montavista
Green Hills Software

Optimized High-Speed Drivers

Hypervisor

Freescale Multicore Silicon

API

Simulation to Hardware
Same Software

Freescale-supplied SDK items
DPAA Pool Channel Implementation

Linux Control Partition

Control Path

Lightweight Executive
PDCP/RLC/MAC/Sch

Doorbells
Shared Memory
INBAND – Via FQ’s

QMAN
Dedicated
Channel

QMAN
Pool Channel

Frame
Queues

Classify
Multicore Software Partitioning

Fixed Partitioning

- IPSEC
- PDCP
- MAC/RLC
- Scheduler
- Scheduler
- SMP Linux
- SMP Linux

Core #1
- Small fixed function LWE kernel with high icache hit ratio

Core #8

Dynamic Load Balancing

- Single LWE App
- SMP Linux
- SMP Linux
- Large Multifunction kernel with lower icache hit ratio but with dynamic load balancing

<table>
<thead>
<tr>
<th>Layer</th>
<th>RRC</th>
<th>PDCP</th>
<th>GTP</th>
<th>SCTP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MAC</td>
<td>IP</td>
<td>UDP</td>
<td>TCP</td>
</tr>
<tr>
<td></td>
<td>PHY</td>
<td>L2/L1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Traditional LTE L2 Downlink vs. DPAA solution

Traditional Implementation (Linux SMP)
• Potential Buffer Copy
• Synchronization/lock Point

DPAA LWE run to completion implementation
(objective: offload e500mc cores)
• Buffer copies are replaced by DPAA enqueue / dequeue operations
• Locking/synchronization is handled in hardware.
System Partitioning / Initialization

- DPAA partitioning and initialization is driven by the Linux device tree which is passed to both the hypervisor and guest OS’s on startup.

- The device tree is Power.org ePAPR compliant with extensions to support the new DPAA features.

- The device tree details
  - Partitioning of cores i.e., Linux/LWE.
  - Physical Memory Areas.
  - Allocation of all Physical resources – eg. Network ports, serial ports etc.
  - Portals
  - BMan Pools
  - FQ Allocation
  - etc …

- The Hypervisor parses the device tree and allocates resources as required.
Parallel Processing Elements
Cores

In order arrival of 3 packets

Packet order is preserved

P3 processing time is greater than P1 or P3

P3 is held until P2 is processed through the ORP
In order arrival of 3 packets

Packet order is preserved
In order arrival of 3 packets

Packet order is preserved
In order arrival of 3 packets

Packet order is preserved

Parallel Processing Elements
Cores
In order arrival of 3 packets

Packet order is preserved

Parallel Processing Elements Cores

P2 processing time is greater than P1 or P3

P3 is held until P2 is processed through the ORP
In order arrival of 3 packets

Packet order is preserved

Parallel Processing Elements
Cores
Software portals have 4 components
- Dequeue: Command registers + DQRR
- Enqueue: EQCR
- Messages: MR
  - Asynchronous error messages (e.g. enqueue rejections)
  - Management commands: command/response registers

Interrupts can be used to signal availability of data or space (in EQCR)

Rings provide finite size FIFOs
- Up to 16 entries for DQRR, 8 entries for EQCR and MR

Portal components are implemented inside QMan to reduce access latency
- Unlike traditional BD rings which are in “memory” and “registers”

QMan can “push” (stash) DQRR entries across Corenet into the appropriate core’s cache

PI and CI are the basic mechanisms used with rings but other forms of notification of data availability and data consumption are supported

When these other mechanisms are used QMan maintains PI/CI
PDCP QMan Stashing Example

- QMan can stash DQRR entries across Corenet into the appropriate core’s cache. The stash size 0 -> 3 Cache lines (64 bytes) can be set for the following components on FQ creation.

- Frame Data
  - Actual Packet Data

- FQ Context
  - Per Queue Context Data ie PDCP user context, Sequence Numbers, ROHC context, Cipher keys etc ….

- Frame Annotation
  - Per Packet Context Data ie Mapping of PDCP Bearer ID to internal structures.
Example DPAA main() for FQ Creation and Dequeue

Main()
{
  .. Init..
  fq = qm_new_fq(g_qm_portal,
      fq,
      channel,
      priority, pdcp_dl_context,
      0, 0, 0, MT_SHARED, 0);
  ..

  while (1) {
    if (entry = qm_dq_dqrr_entry(g_qm_portal)) {
      context = (struct lte_context *)(entry->contextB);
      context->handler(context, entry);
      qm_dqrr_cci_consume(g_qm_portal->p, 1);
    } else {
      idle_loop();
    }
  }
}

Frame Queue Creation sets the system connectivity. FQ's have a number of attributes which determine run time behaviour ie HELD ACTIVE, ORP/ODP

Processing is “data” driven - By the time data arrives at the core we do not need to parse channel, priority, FQ as data processing is driven by context
Comparison of example LTE Data handler()

```c
void pdcp_dl(struct pdcp_dl_context_t *context, struct qm_dqrr_entry *entry)
{
    SBL2_BUFFER_T buffer;
    struct qm_fd fd;

    buffer.length = fd.length20 - ETH_HLEN;
    buffer.data = (uint8_t *) ptov_dpa(fd.addr_lo);
    buffer.offset = fd.offset + ETH_HLEN;
    if (SBL2_PDCP_DL_DataTransfer(&context->bearer, &buffer))
    {
        fd.length20 = buffer.length;
        fd.offset = buffer.offset;
        qman_enqueue_performance(g_qm_portal, RLC_CHANNEL, &fd, 0, 0);
    } else {
        printf("PDCP DL Drop Packet.\n");
    }
}

void pdcp_dl(UINT16 bearer_id, UINT8 *sdu, UINT16 sdu_length)
{
    SBL2_PDCP_RADIO_BEARER_T *bearer = &SBL2_PDCP->radio_bearer[bearer_id];

    pdu = SBL2_GetBuffer();
    LOCK(&(bearer->mutex));
    MEMCPY(sdu, pdu, sdu_length);
    SBL2_PDCP_DL_DataTransfer(bearer, &pdu, sdu_length);
    UNLOCK(&(bearer->mutex));
}
```

**P4080 DPAA based PDCP Handler**

Abstraction of the bman buffer to SBL2_BUFFER_T type allows the same SBL2_PDCP_DL_DataTransfer code to be reused for Linux/SMPLinux/LWE/RTOS

**Linux SMP based PDCP Handler**

Although simpler the original code has a buffer allocation/copy and mutex lock around the data processing
LTE Layer 2 Summary
LTE Layer 2 Code Summary

- Evolving multicore code set derived from “mature” single core base.
- Code developed to date has the benefit of deployment “feedback”.
- Robust software development and management process established.
- Simulation environment facilitating early code development.
- Multicore code generation is underway.
Thank you for attending this presentation. We’ll now take a few moments for the audience’s questions and then we’ll begin the question and answer session.