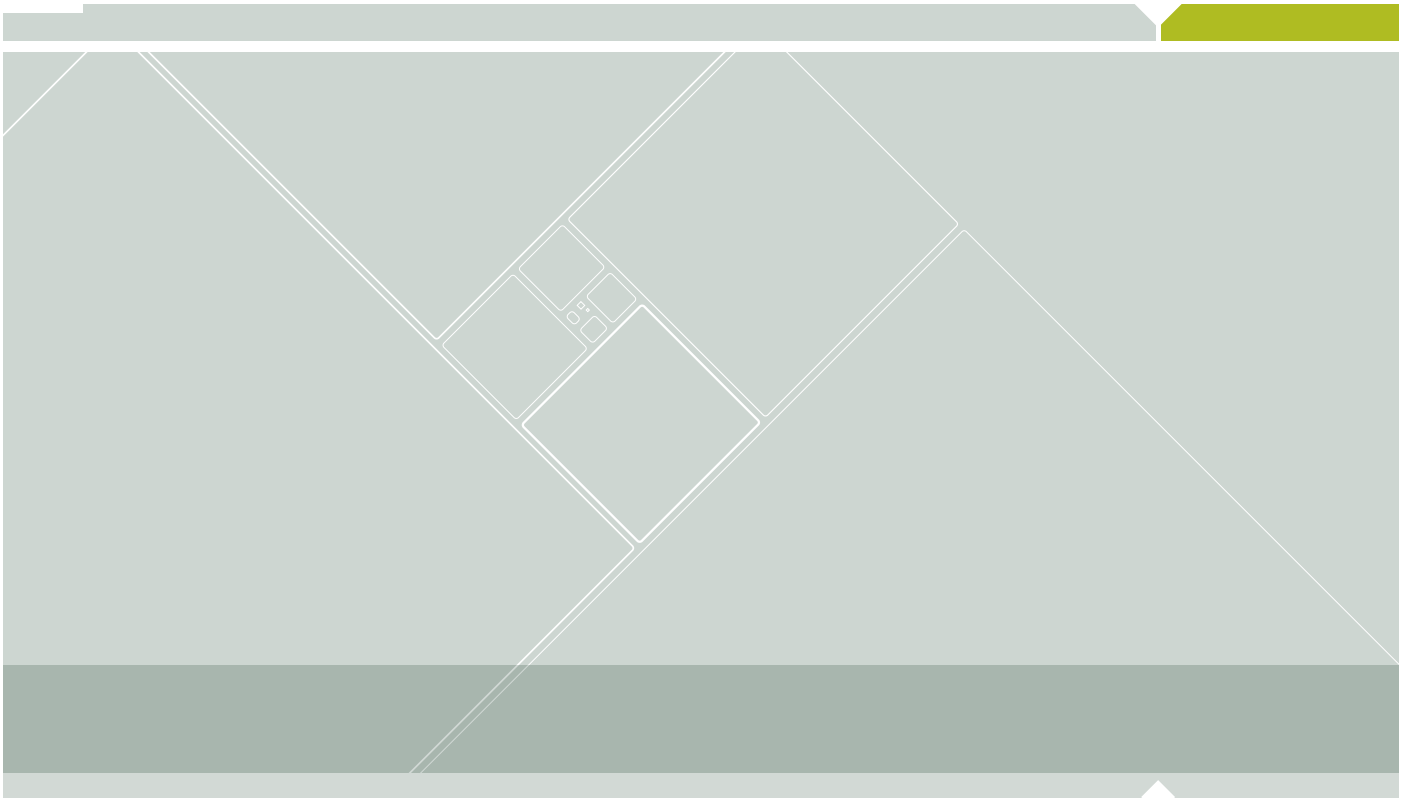


White Paper

Tuning QorIQ Processor Performance with Prism Software Analysis

Processor enablement

criticalblue
Accelerating Embedded Software

freescale™
semiconductor

Introduction

Prism is a software analysis tool which provides support to optimize code for multicore platforms. This example illustrates how Prism was used to assist porting Freescale Semiconductor's LTE Layer 2 software from a single-core processor, based on Power Architecture® technology, to Freescale's QorIQ P4080 eight-core processor. It focuses on parallel programming related issues which would have been difficult and time consuming to resolve without the use of Prism. These issues can be classified as optimizations and defect resolutions. Using Prism, improved code quality and reduced development time ultimately saved both development and maintenance cost.

Contents

1 Freescale's LTE Layer2 Software 4

2 Understanding Your Multicore Application on QorIQ Processors 4

3 Implementing LTE Up/Downlink Parallel Processing 4

3.1 Optimizing LTE Up/Downlink Parallel Processing 5

4 LTE Execution Error Analysis 7

4.1 LTE HARQ Process Corruption 7

4.2 IP Network Packet Reception and Downlink MAC Scheduling Erroneous Interaction 8

5 Conclusion 9

1 Freescale's LTE Layer2 Software

For several years, Freescale has been developing LTE code for processors built on Power Architecture technology. The majority of this development has concentrated on Layer 2, which is medium access controller (MAC), radio link controller (RLC) and packet data convergence protocol (PDCP). Moreover, development was initially focused on single-core devices. Recently the LTE Layer 2 code has been ported to the multicore QorIQ communications platforms. During this transition a number of issues related to parallel programming were encountered. Prism was successfully used to assist in the resolution of these issues.

2 Understanding Your Multicore Application on QorIQ Processors

A QorIQ platform can provide up to eight cores (P4080), making it ideal for high-bandwidth telecom applications such as LTE. In order to utilize the available processing power it is vital to ensure a threaded application runs efficiently. This is especially important when moving an application from a single-core to multicore device due to the potential magnification of any minor issue.

Understanding the dynamic behavior of an application can be very difficult. Even if an application is deemed bug free, how can it be examined to ensure it is running optimally?

Prism allows for the capture and analysis of dynamic software activity on QorIQ devices, enabling quick identification of any performance bottlenecks which can occur during execution. Furthermore, Prism is able to pinpoint potential issues, such as data races, in a threaded implementation including the ability to trace issues back to the root cause in the source code.

3 Implementing LTE Up/Downlink Parallel Processing

A LTE base station must be capable of processing multiple users on both the uplink and downlink within a 1 ms time window. On traditional single-core devices, this is achieved by using a low priority thread to receive IP network traffic and a high priority thread to execute the downlink and uplink sequentially. Consequentially, this restricts each part of the processing chain to a slice of the 1 ms processing budget.

The following code fragments highlight the sequential implementation:

Figure 1: LTE Sequential Processing Flow

```
static void run_dowlink(void)
{
  SBL2_DL_SetQos(logChan, QOS_AMBR_CREDIT);
  hrn_process_downlink();
}

static void run_uplink(void)
{
  hrn_process_uplink();
}

int main(int argc, char *argv[])
{
  create_downlink_packet_reception_thread();

  run_dowlink(); // Initially fill uplink buffer

  while (1)
  {
    err = sigwait(&proc_set, &sig);
    if (err || (sig != SIGALRM))
      ERROR("HRN: Illegal Signal.\n");

    run_dowlink();
    run_uplink();
  }
}
```

The up/downlink functions are called in a loop, which iterates based on a 1 ms timer, to process data accumulated in various buffers. The downlink processes the incoming data stream (from the IP network) and places the results into a buffer. Ordinarily, this data would be sent to Layer 1 for processing, however, the test application routes the data to the L2 uplink. The uplink function processes the data during the next loop iteration. This flow is supported by using two destinations for the data generated by the downlink.

A QorIQ multicore platform provides an opportunity to run IP packet reception, uplink processing and downlink processing simultaneously. This can be achieved by using multiple threads on a Linux® operating system (OS) to spread the workload across multiple cores. In this example, the initial partitioning between up and downlinks has been made with the entire uplink processing taking place in its own thread while the downlink process continues to run in the original main thread of the application. The following code fragments show how this was implemented.

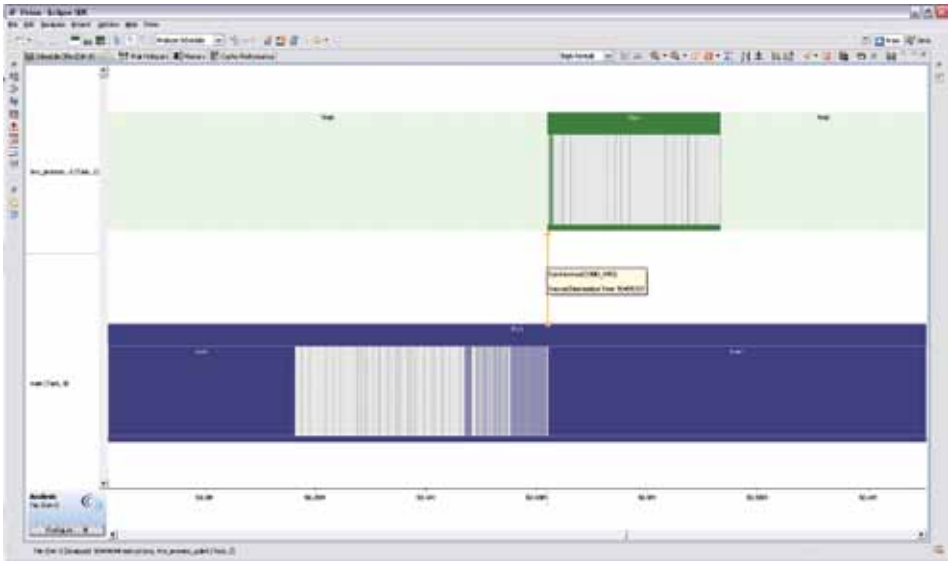
Figure 2: Initial Threaded Implementation

```
static void run_dowlink(void)
{
  SBL2_DL_SetQos(logChan, QOS_AMBR_CREDIT);
  hrn_process_downlink();
}

static void run_uplink(void)
{
  pthread_mutex_lock(&uplink_mutex);
  process_uplink = TRUE;
  pthread_cond_signal(&uplink_condition);
  pthread_mutex_unlock(&uplink_mutex);
}

int main(int argc, char *argv[])
{
  create_downlink_packet_reception_thread();

  create_uplink_processing_thread();

  run_dowlink(); // Initially fill uplink buffer

  while (1)
  {
    err = sigwait(&proc_set, &sig);
    if (err || (sig != SIGALRM))
      ERROR("HRN: Illegal Signal.\n");

    run_dowlink();    // These should
    run_uplink();     // run in parallel
  }
}
```

3.1 Optimizing LTE Up/Downlink Parallel Processing

When the performance was measured on a P4080 processor, it was determined that the code was not running at expected rates. To investigate further and determine where to optimize, the application testbench was run again with Prism trace capture enabled. Once loaded into Prism, it quickly became obvious that the uplink and downlink were not running in parallel. This can be seen clearly in the following screen shot of the Schedule View in Prism:

Figure 3: Schedule View of Original Trace Showing Serialization of Uplink and Downlink Threads



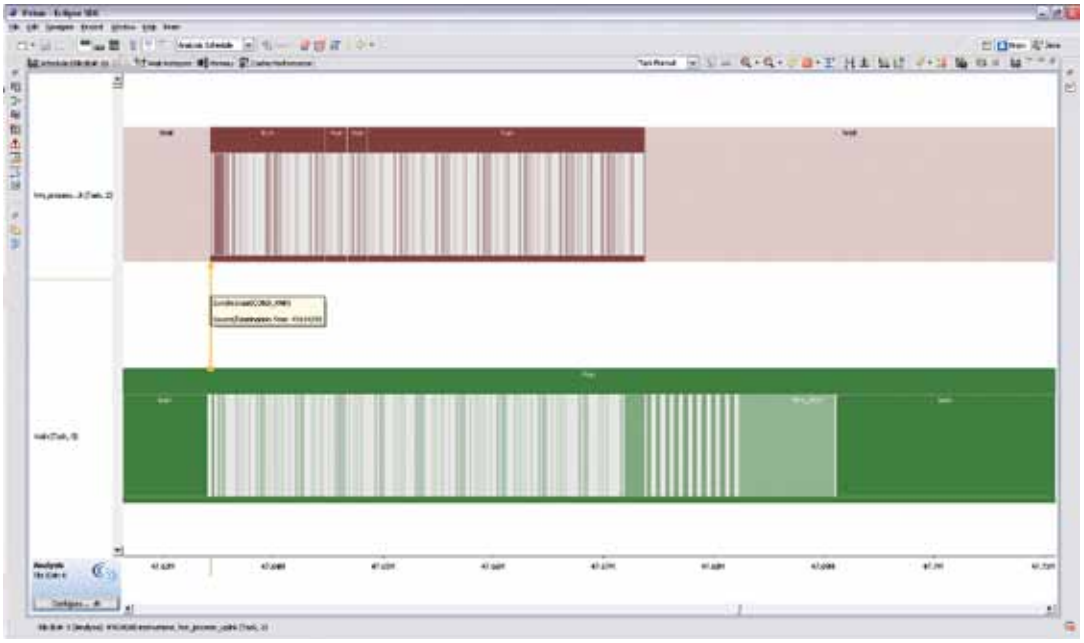
In this view, Prism shows the activity of the two application threads (uplink and main which runs the downlink function) and they are clearly serialized due to synchronization through a conditional variable. After utilizing Prism to pinpoint the relevant lines of source code, it became apparent that it is the *pthread_cond_signal (&uplink_condition)* call in the *run_uplink (void)* function which signals the uplink thread to begin processing. This call is made after the downlink processing has completed due to the processing flow of the original sequential code.

The solution is to call the *run_uplink (void)* function before starting downlink processing, enabling the uplink thread to begin executing while the main thread continues running the downlink routine in parallel. Making the code change outlined in Figure 4 and re-tracing allows us to see the impact in the Prism Schedule view (see Figure 5).

Figure 4: Optimized Processing Flow

```
static void run_dowlink(void)
{
  SBL2_DL_SetQos(logChan, QOS_AMBR_CREDIT);
  hrn_process_downlink();
}

static void run_uplink(void)
{
  pthread_mutex_lock(&uplink_mutex);
  process_uplink = TRUE;
  pthread_cond_signal(&uplink_condition);
  pthread_mutex_unlock(&uplink_mutex);
}

int main(int argc, char *argv[])
{
  create_downlink_packet_reception_thread();

  create_uplink_processing_thread();

  run_dowlink(); // Initially fill uplink buffer

  while (1)
  {
    err = sigwait(&proc_set, &sig);
    if (err || (sig != SIGALRM))
      ERROR("HRN: Illegal Signal.\n");

    run_uplink();
    run_dowlink();    // These now run in parallel
  }
}
```

Figure 5: Uplink and Downlink Running in Parallel



Now there is substantial parallel execution and a corresponding performance increase.

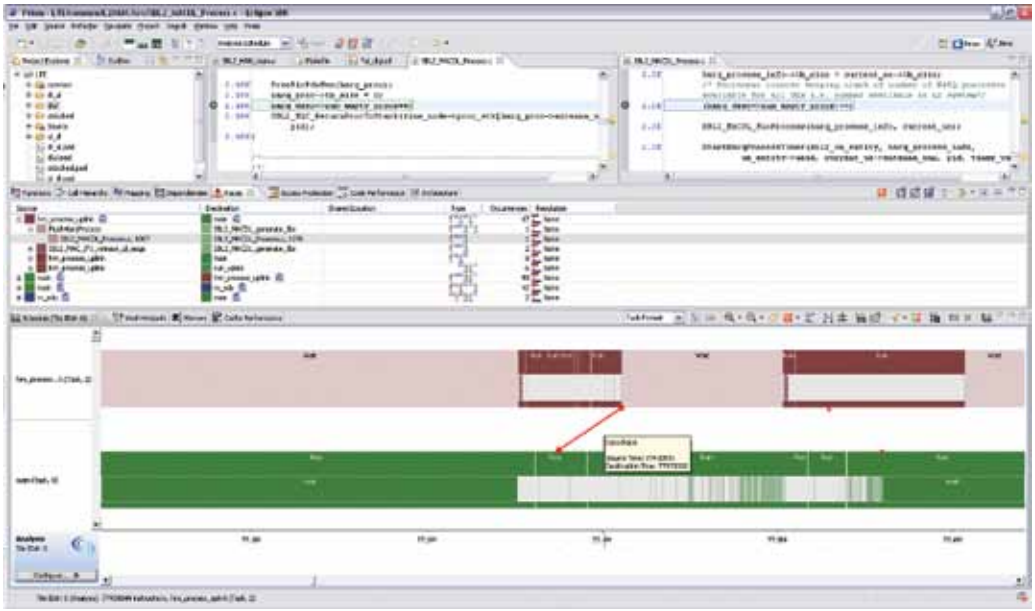
4 LTE Execution Error Analysis

Although the parallel implementation successfully executed all static tests, random failures were experienced when using variable network traffic loads. Prism was used to successfully detect issues which would have otherwise been difficult to identify.

4.1 LTE HARQ Process Corruption

When a transport block is sent by the base station on the downlink it is stored by the MAC layer until feedback is received from the User Equipment (UE). This allows the transport block to be retransmitted with minimal delay upon error detection. This process is known as hybrid automatic repeat request (HARQ). Downlink HARQ feedback is received from the UE on the uplink. Therefore, the obvious solution is to have the uplink process the HARQ feedback upon reception. This works successfully if the uplink and downlink are processed sequentially. However, using the Prism Data Race View (see Figure 6) it was quickly established that HARQ process corruption was possible due to the uplink and downlink threads accessing the HARQ-related structures simultaneously. Realization of the issue is shown in the schedule view where downlink access to the HARQ structures punctuates the uplink access. Finally, the exact source code lines are identified in the source code window. The solution is for the uplink to gather the downlink HARQ feedback but leave processing to the downlink thread. This removes the dependency between the threads and consequently the resulting processing issue.

Figure 6: HARQ Process Race Condition



4.2 IP Network Packet Reception and Downlink MAC Scheduling Erroneous Interaction

When packets are received from the IP network they are processed by the PDCP layer before being passed to the RLC as logical channel service data units (SDUs). The RLC generates a RLC protocol data unit (PDU) for a given logical channel using its stored SDUs when requested from the downlink MAC scheduler. As a result, there are two sources which access the pool of RLC SDUs. During LTE software porting this potential conflict was addressed by using software locks to ensure synchronized access to the RLC buffer pool. However, a side effect of this interaction was overlooked.

In order to ensure an efficient scheduler implementation, the Freescale RLC informs the MAC scheduler when the status of a particular RLC logical channel changes, i.e., if a channel changes from empty to non-empty or from non-empty to empty. This ensures the MAC scheduler only processes channels containing data rather than all open logical channels. This feature operated effectively on single-core devices. Unfortunately, in a multicore environment, it produced an error which manifested as an occasional crash when processing variable network traffic, making the issue difficult to replicate and debug.

Executing the LTE software and capturing a trace allowed the issue to be tracked down without construction of an environment to replicate the problem. Prism's Data Races feature (see Figure 7) outlined an issue with access to the list of "active" logical channels. Moreover, it pin-pointed the lines of code which caused the data race. Furthermore, Prism's "Dependencies" features (see Figure 8) highlighted other related dependencies. This ensured that the scope of the issue was clear, which permitted a holistic view when designing the resolution. The solution is to store any logical channel status changes which occur during scheduling in a mirror list and then apply the changes on completion of scheduling.

Figure 7: Packet Reception and Scheduling Race Condition

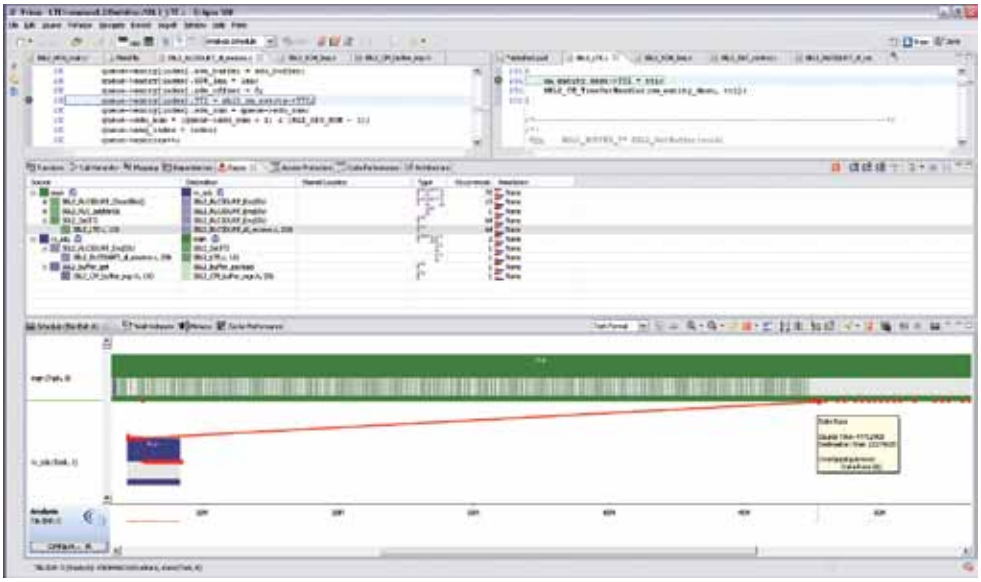
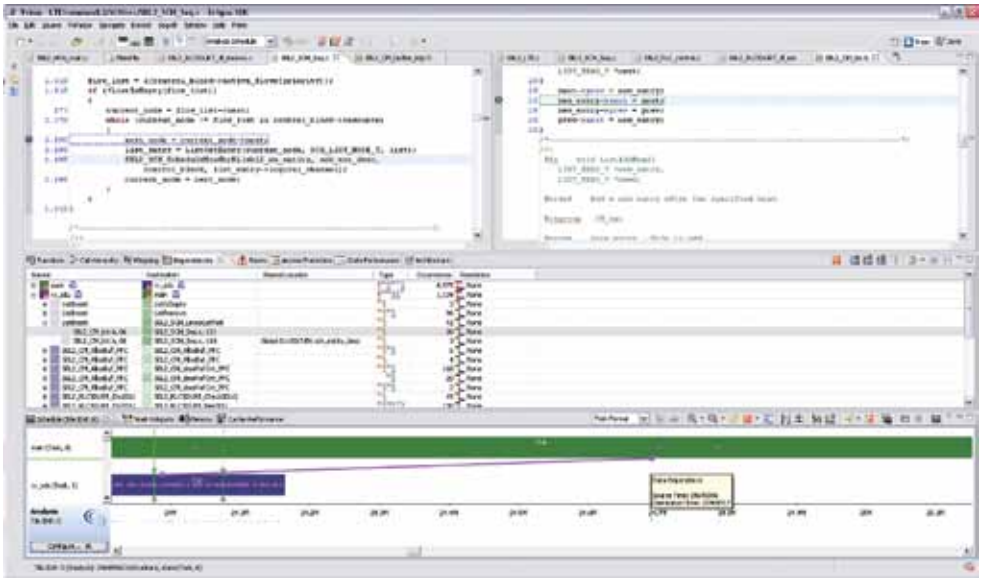


Figure 8: Packet Reception and Scheduling Data Dependency



5 Conclusion

Prism can be used to quickly identify and resolve performance issues and to locate code defects in an LTE implementation running on a P4080 processor. In addition to verification and performance optimizations, Prism supports “What If” analysis features to analyze how sequential code can be parallelized to run on multiple cores. Combined, these features allow the developer to quickly and efficiently target software onto multiple cores.

The QorIQ platform, with its extensive ecosystem of tools and software, provides a low risk path to high-performance multicore for telecom software developers.

How to Reach Us:

Home Page:
www.freescale.com

Power Architecture Information:
www.freescale.com/powerarchitecture

Web Support:
www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

criticalblue
Accelerating Embedded Software

Home Page: **www.criticalblue.com**

Prism Evaluation Request:
www.criticalblue.com/prism/eval

CriticalBlue UK
181 The Pleasance
Edinburgh
EH8 9RU
United Kingdom
Tel: +44 131 655 1500
Fax: +44 131 655 1501
enquiries@criticalblue.com

CriticalBlue USA
San Jose, California
2033 Gateway Place
6th Floor
San Jose, CA 95110
USA
Tel: +1 408 573 3609
Fax: +1 408 437 1201
enquiries_us@criticalblue.com

CriticalBlue JAPAN
Tokyo
Tel: +81(0) 70 6473 5856
enquiries_jp@criticalblue.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright license granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale, the Freescale logo are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. QorIQ is trademark of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.
© 2010 Freescale Semiconductor, Inc.

Document Number: LTEWHTPPRCRTBLA4
REV 0

freescale
semiconductor