Introduction

This application note applies to the following families of devices:

- GT6426x/6xA
- MV64360
- MV9823x

For the purposes of this document, these devices are collectively referred to as "Marvell devices" or simply "devices".

Background

When the local PowerPC processor attached to a Marvell device is interfacing with PCI devices, and when using cache coherency, there are some situations that may result in a system deadlock. This application note describes these scenarios and describes several workarounds to overcome these deadlocks.

Note

Deadlocks only occur when interfacing with PCI devices that have some dependencies between their action as a master and a slave device (e.g. a device that refuses to accept a read transaction as a slave before completing a write transaction as a master). PCI simple devices, such as Ethernet NIC of FC/SCSI adapters, do not have such dependency and, therefore, are not exposed to these potential deadlocks.

Case 1: Consecutive Writes from the CPU to the PCI Device

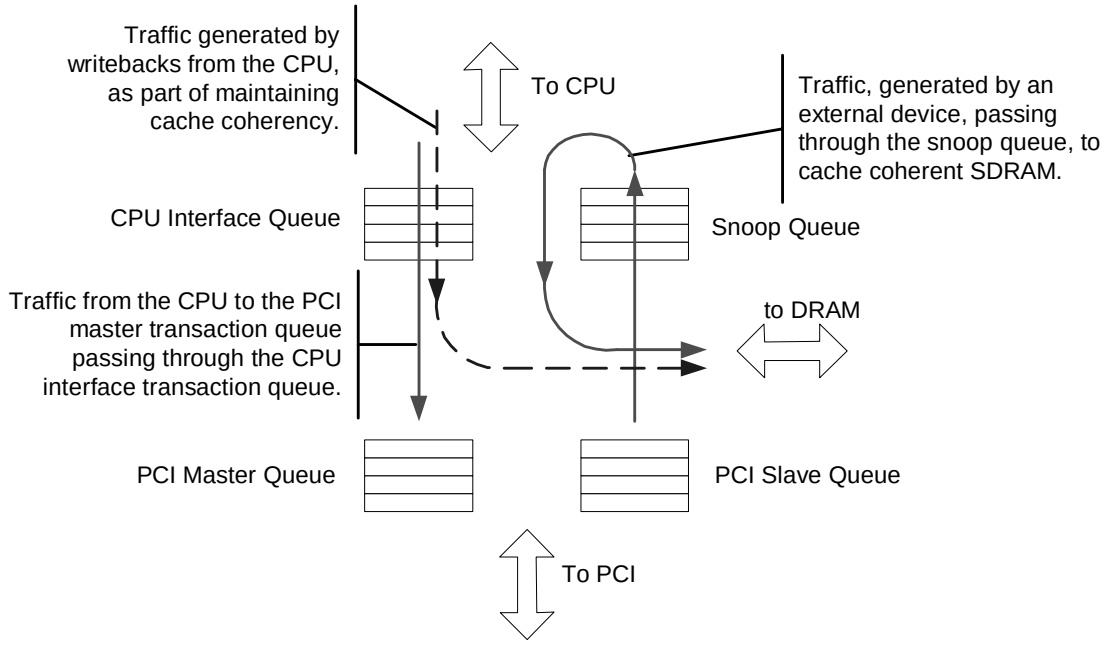
While the PCI device initiates read transactions to the Marvell device's DRAM cache coherent region, a CPU performs consecutive writes to the PCI device. If the PCI device refuses to respond to the CPU writes (consistently terminating the write transactions with RETRY) before receiving the read response, a deadlock might occur.

Note

A PCI device that behaves as described also violates PCI spec transactions ordering rules.

Figure 1 shows the Marvell devices' internal buffering scheme related to this scenario.

Figure 1: GT-6426x Internal Buffers



Note

The above figure does not detail the entire buffering and queuing implementation. It only shows the buffering that is relevant to this application note.

Some factors related to this internal buffering scheme that can influence whether a deadlock state occurs.

- The PCI master queue can absorb up to four write transactions. When it becomes full, additional CPU-to-PCI writes (green) are "stuck" in the CPU interface transaction queue.
- When the PCI slave receives a read request from a cache coherent region, the transaction first goes to the snoop queue. Only after the snoop is resolved does the transaction go to the DRAM controller for execution (red).
- The snoop process is composed of Marvell device's CPU interface generating an address only transaction on the CPU bus and, in case of a hit in the modified cache line (ARTRY* asserted), waiting for the processor cache line copy back to complete the transaction (blue).
- The cache line copy back is being served by the Marvell device's CPU interface, as is any other transaction. These transactions go through the same transaction queue.

If the processor issues consecutive writes to the PCI device that fills the PCI master queue, the next CPU-to-PCI write will be stuck in the CPU interface queue. If the PCI device then issues a read transaction to a cache coherent region, and this read transaction hits a modified cache line, the cache line copy back also gets stuck in the CPU interface queue. The cache line copy back is right after the CPU-to-PCI write transaction. It is stuck behind this CPU-to-PCI write because all writes are performed in order. If the PCI device refuses to serve the CPU writes (terminates all writes with RETRY), before the read is completed, a deadlock occurs.

Possible workarounds to solve the resulting deadlock include:

- Instead of the CPU performing consecutive writes to the PCI, allow the CPU to prepare the data in DRAM. Then, activate one of the Marvell device's IDMA channels to copy the data from DRAM to the PCI device. By read polling to the IDMA Channel Control register's active bit, the CPU can identify when the transfer is completed.

Note

If the processor has a big block of data to move to the PCI, it is a good practice to use the IDMA channels to copy the data from DRAM to the PCI device. Since the PCI bus bandwidth is much lower than the CPU bus bandwidth (and the DRAM bus), direct CPU-to-PCI transfers will limit the CPU bus throughput to be the same as the PCI bus throughput. If using the IDMA channels, the CPU bus does not suffer from the lower PCI bus bandwidth.

- Use Write Through (WT) cache policy instead of Write Back (WB). If using WT, the PCI reads do not go through the snoop queue. The deadlock is prevented.
- Do not use the Marvell device's cache coherency support. Maintain cache coherency by software.
- Implement a software flow control mechanism that will limit the number of pending CPU-to-PCI writes. As explained, the deadlock might occur when the PCI master queue becomes full. If this can be prevented by software means, the deadlock never occurs.

  One way to implement a software flow control mechanism is to generate a dummy CPU-to-PCI read after every three CPU-to-PCI writes. The CPU only continues with the next write access after it receives the read response. When the read completes, it is guaranteed that all previous writes are flushed.

Note

This flush read method only applies to the MPX CPU bus that supports out of order data tenures. In 60x bus mode, if the processor issues a cache line copy back (as a result of a Marvell device snoop transaction) immediately after the dummy read, the copy back data tenure will not happen before the dummy read data tenure completes. An alternative way is to activate an IDMA channel to generate the dummy read.

- Use the PCI master retry counter to break the deadlock. As explained, the deadlock occurs when the PCI device refuses to accept the CPU write transactions (terminates the writes with RETRY). If activating the PCI master retry counter (setting PCI Timeout and Rtry register's RetryCtr field to a value greater than '0'), the Marvell device's PCI master relinquishes the retried transaction when the retry counter expires. This results in the master flushing the transaction from its queue, setting the PCI Interrupt Cause register's MRetry bit, and asserting the CPU interrupt

  The software can identify the failed transaction by reading PCI Error Address register and re-transmitting all write data from that point.

Note

In most application, there is not a large penalty for using the retry counter since the deadlock occurrence is rare. In typical applications, most of the Marvell device's snoops do not hit dirty cache lines and no deadlock takes place.

Case 2: The CPU Performs a Read to the PCI Device

While the PCI device initiates consecutive write transactions to the Marvell device's DRAM cache coherent region, the CPU performs a read to the PCI device. If the PCI device refuses to respond to the CPU read (the PCI device continuously terminates the CPU read with RETRY) before completing the writes, a deadlock might occur.

Note

This scenario is only relevant to a 60x CPU bus that does not support out of order data tenures. This scenario does not apply when using an out of order MPX bus.

The PCI transaction ordering rules requires that a bridge-device read response flushes all posted write data that was posted prior to the read. This means that bridge-device read response, as a target, depends on the completion of its write transactions as a master. This leads to the situation in which the PCI bridge-device might be exposed to this deadlock scenario.

Some factors related to this situation that can influence whether deadlock state occurs.

- When using the Marvell device's cache coherency support, the PCI slave queue can absorb up to four cache lines of posted write data (128 bytes). When the queue is full, additional PCI writes are terminated by the device with RETRY.
- The PCI writes to a cache coherent region first goes to a snoop queue and, only after the snoop is resolved, continues to the DRAM controller for execution. A snoop queue can absorb up to four cache line transactions. This means that the PCI-to-DRAM write path is full after eight cache lines.
- The Snoop process is composed of the Marvell device's CPU interface generating an address only transaction on the CPU bus and, in case of a hit in modified cache line (ARTRY* asserted), waiting for completion of the processor cache line copy back.
- The 60x bus restricts in order data tenures. If the CPU issues a read request and then issues a cache line copy back (as a result of a Marvell device snoop), the copy back data tenure cannot be driven on the bus. This means the snoop is not resolved before the completion of the previous read transaction data tenure.

If the PCI device issues consecutive writes to the Marvell device's cache coherent DRAM and the PCI slave queue and snoop queue become full, the following PCI write is retried. Now, if one of the snoop transactions (caused by the PCI writes) results in hit in modified cache line, the CPU issues a cache line copy back. If just before this copy back, the CPU issues a read to the PCI device, and the PCI device read response depends on the completion of it's pending write transactions as a master, a deadlock occurs. The PCI device terminates the read with RETRY as long as it has pending write transactions to complete. The write transactions cannot complete since the oldest snoop must be first resolved. The snoop cannot be resolved due to the in order nature of the 60x bus. The "in order" requirement does not allow the cache line copy back to complete before the completion of the read transaction data tenure.

Possible workarounds to solve the resulting deadlock include:

- Use Write Through (WT) cache policy instead of Write Back (WB). If using WT, there is no cache line copy back. Thus, the deadlock is prevented.
- Do not use the Marvell device's cache coherency support. Maintain cache coherency by software.
- If the PCI device behavior is controllable, implement a flow control mechanism that limits the number of pending PCI writes. As explained above, the deadlock may only occur when the PCI slave queue and snoop queue are full. Prevent this condition from ever occurring and the deadlock will never happen.

  Implement a flow control mechanism by having the PCI device generate a dummy PCI read after every "N" write transactions (N is PCI device implementation dependent). The PCI device only continues with the next write access after it receives the read response. When the read response is complete, it is guaranteed that all previous writes are flushed.
- Use the PCI master retry counter to break the deadlock. As explained, the deadlock occurs when the PCI device refuses to accept the CPU read transaction (terminates with RETRY). If activating the PCI master retry counter (setting PCI Timeout and Rtry register's RetryCtr field to a value greater than '0'), the Marvell device's PCI master relinquishes the retried transaction when the retry counter expires. When this happens, the master flushes the transaction from it's queue, returns invalid data back to the CPU, sets the PCI Interrupt Cause register's MRetry bit, and asserts a CPU interrupt.

  The software identifies the failed transaction by reading the PCI Error Address register and issues the transaction again.

Note

In most application, there is not a large penalty for using the retry counter since the deadlock occurrence is rare. In typical applications, most of the Marvell device's snoops do not hit dirty cache lines and no deadlock takes place.

Case 3: Two Devices Interface Over the PCI Bus

Two Marvell devices interface with each other over the PCI bus. If each PowerPC processor attached to each of the devices simultaneously issues multiple consecutive writes to the cache coherent DRAM of the peer device, a deadlock might occur.

Some factors related to this situation that can influence whether a deadlock state occurs (refer to Figure 1):

- When using the Marvell device's cache coherency support, the PCI slave queue can absorb up to four write transactions. When it becomes full, additional CPU-to-PCI writes are "stuck" in the CPU interface transaction queue.
- When using the Marvell device's cache coherency support, the PCI slave queue can absorb up to four cache lines of posted write data (128 bytes). When it becomes full, additional PCI writes are terminated by the device with RETRY.
- The PCI writes to a cache coherent region first go to a snoop queue and, only after the snoop is resolved, go to the DRAM controller for execution. A snoop queue can absorb up to four cache line transactions. This means that the PCI-to-DRAM write path is full after eight cache lines.
- The Snoop process is composed of the Marvell device's CPU interface generating an address only transaction on the CPU bus and, in case of a hit in modified cache line (ARTRY* asserted), waiting for the completion of the processor cache line copy back.
- The cache line copy back is being served by the Marvell device's CPU interface, as is any other transaction. These transactions go through the same transaction queue.
- When the CPU interface queue is full, no new transaction are accepted on the CPU bus. The device does not respond with AACK* to new CPU transaction nor to its own snoop transaction.

If each processor issues consecutive writes to the peer device's DRAM over the PCI bus, the PCI master queues will become full and the next CPU writes are stuck in the CPU interface queue, similar to Case 1. Since the writes are targeted to cache coherent DRAM region, they all result in snoop transactions issued on the CPU bus. If snoops in both sides hit modified cache lines, the CPUs cache line copy back is stuck behind the CPUs writes to the PCI. This results in a deadlock situation. Even if snoops do not hit modified cache lines, as soon as the CPU interface queues in both devices are full, snoops are no longer being served (AACK* is not asserted) and there is a deadlock situation.

Possible workarounds:

- Instead of the CPU performing consecutive writes to the PCI, allow the CPU to prepare the data in DRAM. Then, activate one of the Marvell device's IDMA channels to copy the data from DRAM to the PCI device. The CPU can identify when the transfer is completed by read polling to the IDMA Channel Control register's active bit.

Note

If the processor has a big block of data to move to the PCI, it is a good practice to use the IDMA. Since the PCI bus bandwidth is much lower than the CPU bus (and the DRAM bus), direct CPU-to-PCI transfers limit the CPU bus throughput to match the PCI bus throughput. By using IDMA, the CPU bus does not suffer from the lower PCI bus bandwidth.

- Do not use GT-6426x cache coherency support. Maintain cache coherency by software.
- Implement a software handshake between the two CPUs that prevents simultaneous writes to cache coherent DRAM in both directions.
- Implement a software flow control mechanism that limits the number of pending CPU-to-PCI writes. As explained, the deadlock might occur when the PCI master queue becomes full. If this can be prevented by software means, the deadlock never occurs.

  One way to implement a software flow control mechanism is to generate a dummy CPU-to-PCI read after every three CPU-to-PCI writes. The CPU only continues with the next write access after it receives the read response. When the read completes, it is guaranteed that all previous writes are flushed.

Note

This flush read method only applies to the MPX CPU bus that support out of order data tenures. In 60x bus mode, when a processor issues a cache line copy back (as a result of a snoop transaction) following a dummy read, the copy back data tenure will not occur before the dummy read data tenure completes. An alternative method is to activate an IDMA channel to generate the dummy read.

- Use the PCI master retry counter to break the deadlock. As explained before, the deadlock occurs when the write paths in both Marvell devices are full and the PCI slave interfaces of the two devices cannot accept any new write transaction (terminates the writes with RETRY). If activating the PCI master retry counter (set the PCI Timeout and Rtry register's RetryCtr field to a value greater than '0'), the Marvell device's PCI master relinquishes the retried transaction when the retry counter expires. When this happens, the master flushes the transaction from it's queue, sets the PCI Interrupt Cause register's MRetry bit, and asserts a CPU interrupt.

  The software can identify the failed transaction by reading PCI Error Address register and re-transmitting all write data from that point.

Summary

There is potential deadlock when using the Marvell device's cache coherency and interfacing PCI devices that have some dependencies between their action as target and as master. The deadlock is a result of a closed loop between the local PowerPC processor and the PCI device. There are several ways to break this closed loop. The simplest method that applies to all cases is to use the Marvell device's PCI master retry counter.

Marvell Semiconductor, Inc.
2350 Zanker Road, San Jose, CA 95131
Phone: (408) 367-1400, Fax: (408) 367-1401