Interrupt Moderation Using Intel Gigabit Ethernet Controllers

Application Note (AP-450)

Revision 1.1

September 2003

Revision History

Date	Revision	Description
Sept 2003	1.1	The document was modified to be applicable to Intel Gigabit Ethernet Controllers except the 82542, 82543, and 82544. In other words, this document applies to the 82540, 82545, 82546, 82541, and 82547.
May 2003	1.0	Initial release.

Contents

1.0 Introduction

The primary topic of this application note describes the usage of the interrupt moderation features of the Intel® 82540EM Gigabit Ethernet Controller and how software may use these features to improve system and controller performance. However, the information contained in this document is also applicable to other Intel Gigabit Ethernet Controllers except the 82542, 82543, and 82544.

The reader is assumed to have a working knowledge of network device drivers.

2.0 Background

2.1 Basic Interrupt Processing

In the absence of any interrupt moderation, the controller will interrupt the CPU after every packet event (in other words, after every packet transmission and reception). Typically, this involves the following steps:

1. The controller successfully transmits or receives a packet.
2. The controller generates an interrupt in response to this event.
3. The CPU suspends its current activity in order to handle the interrupt. Typically, this involves saving state information and executing an interrupt handler for the controller.
4. Software (typically, a device driver) examines the controller to determine the cause of the interrupt. The software may also take additional action(s) based on the exact nature of the interrupt.
5. The CPU resumes its previous activity.

At low traffic rates, this behavior is acceptable since this process occurs relatively infrequently. However, as traffic rates increase, the system spends more and more time servicing these interrupts. The overhead of processing these interrupts begins to degrade overall system performance as the CPU spends the majority of its time scheduling and executing the interrupt handler. If the traffic rate continues to increase, the traffic may overrun the controller causing it to drop packets, or the system itself may become temporarily unusable.

In addition to the packet events described above, the controller may also interrupt after certain external events, such as a change in link status. Since these other events occur relatively infrequently, they will not usually degrade system performance in the same manner as the packet events. Therefore, the remainder of this document will specifically focus on handling the packet events.

2.2 Interrupt Handling with Interrupt Moderation

To combat potential problems of interrupts from packet events, the 82540EM controller employs a series of timers for moderating, or limiting, the number of interrupts it generates.

Rather than interrupting immediately upon the transmission or reception of a new packet, software can configure the controller to delay the generation of this interrupt. By postponing this interrupt, the controller can collect any additional interrupt events that arrive within this delay period. When it finally does interrupt, the controller is then able to deliver several events to software all at once.

Software can typically process these events in a single iteration of its interrupt handler. By bundling and processing interrupts in bulk, software and hardware can operate at higher rates of traffic without incurring excessive scheduling and execution overhead. Thus, software and hardware are able to operate more efficiently without compromising performance.

In effect, interrupt moderation adds an additional step to the procedure outlined above:

1. The controller successfully transmits or receives a packet.
2. The controller delays delivery of the interrupt in order to transmit or receive additional packets.

3. The controller generates an interrupt in response to the original event.
4. The CPU suspends its current activity in order to handle the interrupt.
5. Software examines the controller to determine the cause of the interrupt. Software processes the packet event(s).
6. The CPU resumes its previous activity.

Transmit tests with an 82545EM in a dual-processor Intel® Pentium® 4 Xeon™ system running Microsoft* Windows 2000* have shown an 11% reduction in CPU utilization at wire-speed using interrupt moderation. Receive tests have demonstrated a 30% reduction in CPU utilization in this same configuration, also at wire-speed.

2.3 Trade-offs Inherent with Interrupt Moderation

Although interrupt moderation increases the overall interrupt-processing efficiency, it also increases the average latency incurred by each packet. By delaying the delivery of an interrupt, the controller is also delaying the delivery of packet information. As a result, determining optimal interrupt moderation settings usually involves a trade-off between latency and efficiency.

When network utilization is low, delayed interrupts are unlikely to improve performance since the rate of packet transmission or reception is relatively infrequent. In these cases, shorter interrupt delays are desirable to minimize the latency on each packet.

When network utilization is high, the system must operate as efficiently as possible. Every extra CPU or bus cycle spent on interrupt-processing overhead is one less cycle available for processing the actual packet data. Larger interrupt delays are desirable in these situations, to minimize interrupt-processing overhead and improve efficiency. At the same time, excessive interrupt delays may lead to resource starvation and overrun conditions. Software must negotiate between these two extremes to determine the optimal configuration for the expected workload.

Ultimately, the goal of interrupt moderation is to reduce the CPU overhead inherent in handling interrupts without adding significant packet latency or causing undue packet loss.

3.0 Interrupt Moderation Features of the 82540EM Network Controller

The 82540EM contains five different mechanisms for managing the interrupt rate:

- Two absolute timers (one for transmit interrupts, and the other for receive interrupts).
- Two packet timers (one for transmit interrupts, and the other for receive interrupts).
- One master timer for throttling all interrupt sources.

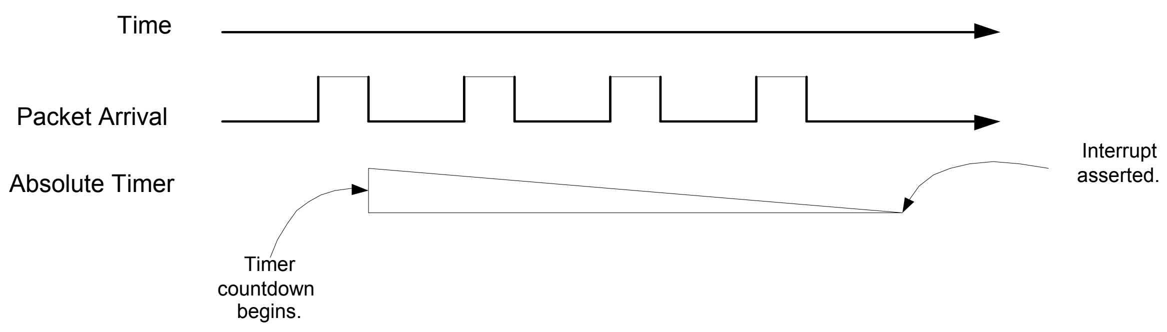
The different mechanisms each have different strengths and weaknesses; when used in conjunction, the different timers complement each other, providing a flexible and powerful scheme for managing interrupts. Each timer is described in more detail below.

3.1 Absolute Timers

The absolute timers delay the assertion of an interrupt to allow the controller to collect additional interrupt events before delivering them to software. The absolute timers are particularly useful in high traffic environments.

The receive absolute timer starts to count down upon receipt of the first packet (after software has enabled interrupts). Subsequent packets, if any, do not alter the countdown. Once the timer reaches zero, the controller generates a new interrupt. In the 82540EM, 82545EM and 82546EB controllers, software controls the receive absolute timer using the RADV register (offset 282Ch).

Figure 1. Receive Absolute Timer



NOTE: The arrival of new packets after the timer has started does not affect the countdown.

The transmit absolute timer starts to count down upon transmission of the first packet (after software has enabled interrupts). Subsequent packets, if any, do not alter the countdown. Once the timer reaches zero, the controller generates a new interrupt. Software controls the transmit absolute timer using the TADV register (offset 382Ch).

The delay values are intended to be relatively large (several packet times in duration). This allows the controller to transmit or receive multiple packets in succession prior to generating an interrupt.

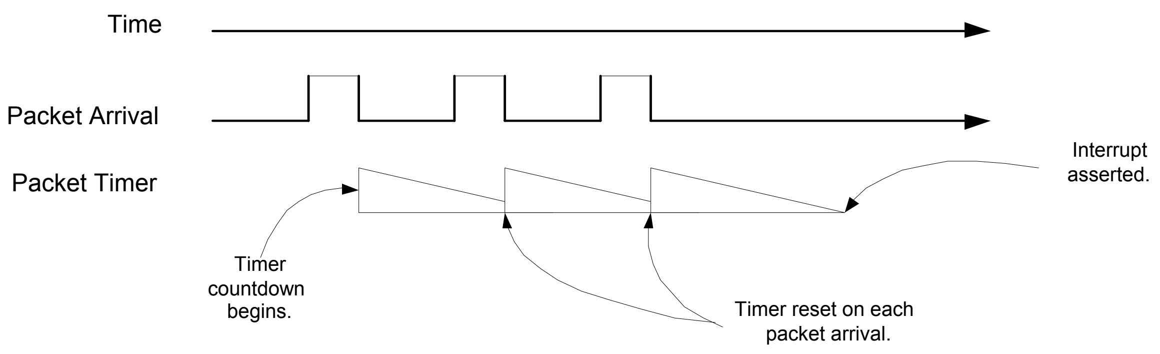
The drawback of absolute timers is that a single packet will incur the full countdown latency even when traffic rates are low. Therefore, the absolute timers will not perform well in low traffic situations.

3.2 Packet Timers

The packet timers are inactivity timers, triggering interrupts when the link has been idle for a suitably long interval. Software can use these timers to minimize packet latency in low traffic environments.

The receive packet timer starts to count down upon receipt of a new packet. If the controller receives another packet before the timer expires, it resets the timer to its original value and restarts the countdown. If the timer ever reaches zero, the controller generates a new interrupt. Software controls the receive packet timer using the RDTR register (offset 108h).

Figure 2. Receive Packet Timer



NOTE: The arrival of new packets after the timer has started causes the timer to restart.

The transmit timer begins to count down upon transmission of a new packet. If the controller transmits another packet before the timer expires, it resets the timer to its original value and restarts the countdown. If the timer ever reaches zero, the controller generates an interrupt. Software controls the receive packet timer using the TIDV register (offset 440h).

Unlike the absolute timer delays, the packet timer delays are intended to be short (possibly two or three packet times in duration). This minimizes the latency suffered by each packet.

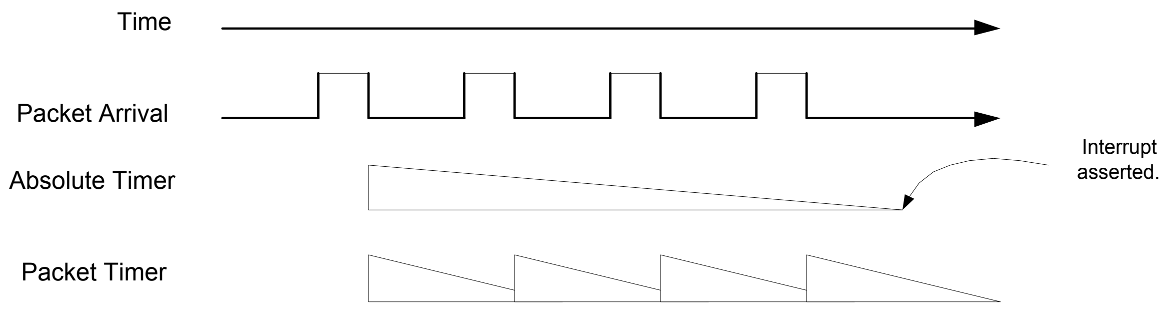
The drawback of the packet timers is that they may be chained indefinitely. Under a sustained load, the packet timer will never expire until the controller has completely exhausted all of its resources. Therefore, the packet timers will not perform well in high traffic situations.

3.3 Combining the Timers

When the absolute timer and packet timer are used together, they are complementary. The absolute timers ensure that interrupts occur even when packets arrive fast enough to prevent the packet timer from ever expiring. The packet timers ensure that even in low traffic conditions, the controller will interrupt with relatively low latency when traffic subsides. Software can use both timers simultaneously to optimize for both types of traffic loads.

Under sustained loads, the absolute timers will be the primary source of device interrupts. Figure 3 illustrates this process. In these situations, each packet will incur a latency of one-half of the absolute timer delay.

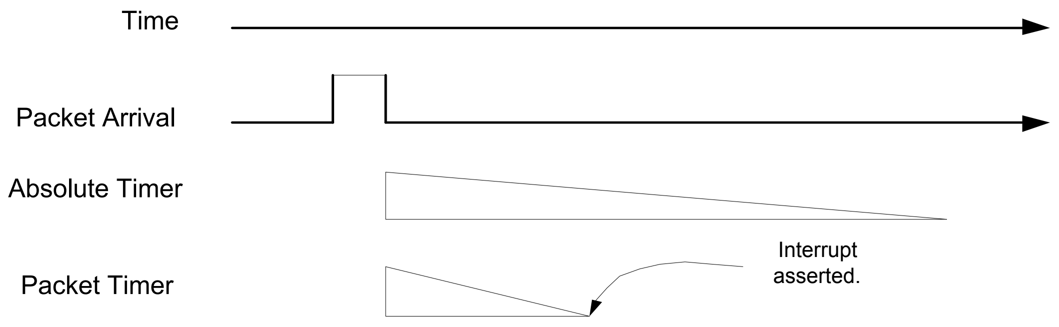
Figure 3. Absolute and Packet Timers at High Traffic Rate



NOTE: In these situations, the absolute timer is the source of most device interrupts.

Under light loads or brief bursts of traffic, the packet timers will be the primary source of interrupts. Figure 4 illustrates this process. In these situations, the packet timers determine the latency suffered by most packets. The packet timers also determine the minimum traffic rate required to trigger the absolute timer interrupts (in other words, if the traffic rate is high enough to prevent the packet timer from ever expiring, then the controller will not interrupt until the absolute timer has expired).

Figure 4. Absolute and Packet Timers at Low Traffic Rate



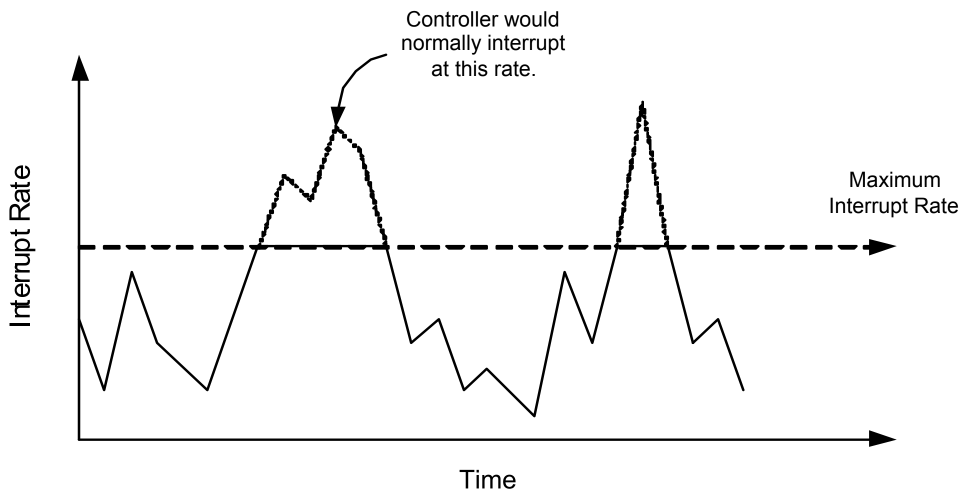
NOTE: In these situations, the packet timer is the source of most device interrupts.

3.4 Interrupt Throttling

A few extra factors make the process of determining optimal timer settings more challenging. First, while software can configure each pair of timers for the expected workload, the transmit and receive timers operate independently. Therefore, transmit interrupts can disrupt the idealized behavior of the receive interrupt processing and vice-versa. Second, Ethernet traffic is inherently unpredictable, exhibiting both large surges in traffic and other periods of relative inactivity. These fluctuations trigger corresponding storms and lulls in the interrupt rate. Lastly, the use of advanced controller features such as TCP Segmentation can introduce additional considerations by altering the usual timing of transmit events.

To limit these effects, the 82540EM also provides an interrupt throttling mechanism for placing an upper bound on the controller's interrupt rate. The throttling mechanism operates independently of any interrupt source. No network events affect the throttle mechanism. Software can use the throttle timer to limit the controller to a maximum interrupt rate, regardless of the traffic rate or other external factors. Software controls the throttle timer using the ITR register (offset C4h).

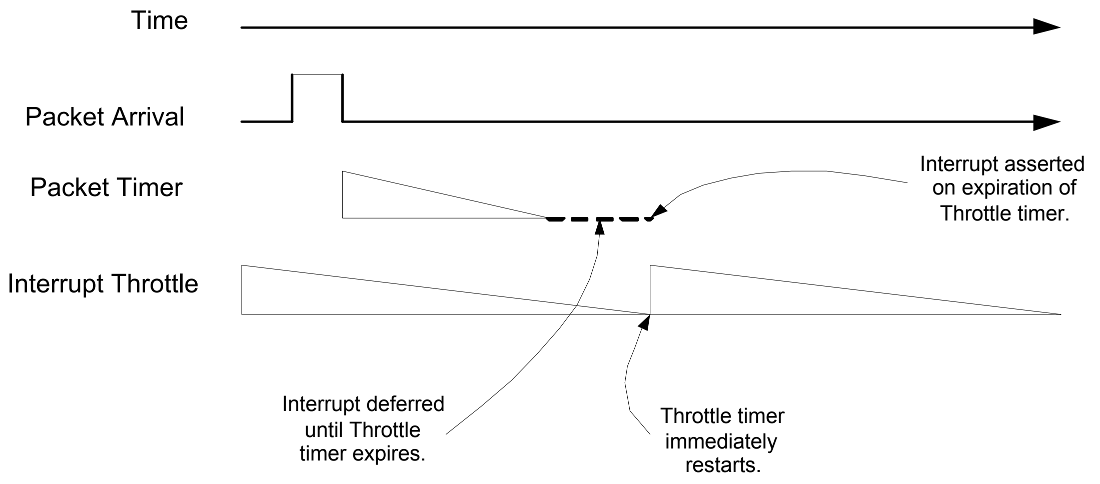
Figure 5. Effect of the Interrupt Throttle on Controller's Interrupt Rate



NOTE: The interrupt throttle places a ceiling on the number of interrupts per second that the controller may generate.

Similar to the absolute and packet timers, the throttle timer is a simple countdown timer. The controller blocks all interrupt sources until the throttle timer expires. If interrupt events are pending when the throttle timer expires, the controller then generates an interrupt. When the countdown reaches zero, the throttle timer simply resets and restarts its countdown.

Figure 6. Interrupt Throttling Used with Packet Timer


Time
Packet Arrival
Packet Timer
Interrupt Throttle
Interrupt asserted on expiration of Throttle timer.
Interrupt deferred until Throttle timer expires.
Throttle timer immediately restarts.

NOTE: The throttle timer gates all other device interrupt sources. The throttle timer runs continuously, regardless of packet events.

The strength of the interrupt throttle (its guaranteed maximum interrupt rate) is also its weakness. On average, under light loads the throttle timer adds half of its countdown delay to the latency of each packet. Under heavy loads, it is functionally equivalent to the absolute timers. Therefore, software will typically use the interrupt throttle as a means of restricting the controller's overall interrupt rate rather than as a third interrupt-generation mechanism. For optimal performance, software may configure the throttle mechanism for a slightly higher-than-desired interrupt rate to reduce the per-packet latency described above.

4.0 A Sample Configuration

Some example settings are described below. These numbers are for illustration purposes only. For optimal performance, the exact controller configuration is best determined through actual experimentation.

The discussion below assumes that software is optimizing for full-size frames of 1538 bytes.

The calculations below make use of the following facts:

- Gigabit Ethernet operates at 1.0 Gb/s or 1,000,000,000 bits per second. At this speed, the time required to transmit or receive a single bit (in other words, the bit-time) is 1.0 nanosecond.
- A full-size Ethernet frame requires 1538 bytes (12,304 bits) of bandwidth:
  - 8-byte preamble and start-of-frame delimiter
  - 14-byte Ethernet header
  - 1500-byte payload
  - 4-byte FCS
  - 12-byte inter-packet gap
- The controller can transmit or receive a full-size frame every 12.3 microseconds or approximately 81,000 packets per second.

4.1 Absolute Timers

Configuring the absolute timers is typically a matter of determining the desired interrupt rate or the desired number of packets per interrupt. To receive approximately 3000 interrupts per second, software would configure the absolute timers to interrupt every 333 microseconds.

Alternately, to receive approximately 50 packets per interrupt, the controller must interrupt approximately 1620 times per second (81,000 packets-per-second at 50 packets-per-interrupt). Software would then configure the absolute timers to interrupt every 617 microseconds.

4.2 Packet Timers

Experiments have shown that values between 20 and 40 microseconds work well for the packet timers.

Software might set the packet timers to expire after 2 full-length packet-times, or approximately 25 microseconds. The packet timers would then expire when throughput falls below about 333Mbps (two unused packet-times follow every packet arrival, so approximately one-third of the total bandwidth is in use). At greater levels of utilization, the packet timers would likely chain repeatedly until the one of the absolute timers expired.

4.3 Interrupt Throttle Timer

Configuring the throttle timer is simply a question of determining the desired maximum interrupt rate. As described earlier, software may realize better results by setting the throttle timer to interrupt slightly more often than desired to reduce unnecessary latencies.

For typical applications, software might configure the controller to interrupt no more than 5000 times per second.

Different operating systems and environments will be capable of sustaining different maximum interrupt rates. Experiments have demonstrated that Microsoft Windows-based operating systems perform best when the device interrupts between 4,000 and 12,000 times per second. Linux-based operating systems appear to perform best with an interrupt rate between 1,000 and 8,000 interrupts per second. Other operating systems will perform differently.

4.4 Additional Tuning Considerations

The example configuration described above will require modifications to suit the intended environment. The following factors may influence the tuning of the interrupt moderation parameters:

- The latency associated with scheduling an interrupt handler. Larger scheduling latencies imply larger packet latencies. Lower interrupt delays may be required in these situations to avoid overrun conditions and excessive per-packet delays.
- The cost associated with handling an interrupt. In OS with low-cost interrupts, higher interrupt rates may be acceptable. In OS with high-cost interrupts, lower interrupt rates may be required.
- The expected mixture of packet sizes. The preceding discussion assumes that software is optimizing for full-size frames. Optimizing for small packets or for a variety of packet sizes requires recalculating the expected packet rate.
- The expected network utilization. High utilization implies high traffic rates, which makes the controller more susceptible to overrun conditions if it delays interrupts too long.