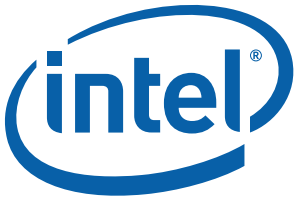

Intel® 82576 SR-IOV Driver Companion Guide

Overview of SR-IOV Driver Implementation

LAN Access Division

322192-001

Revision 1.00

June 2009

Legal

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

*Other names and brands may be claimed as the property of others.

Revisions

Date	Revision	Description
November 17, 2008	0.1	Initial Outline
April 7, 2009	0.75	Rough Draft for initial engineering review
April 14, 2009	0.90	Updates based upon feedback and updated drivers
June 10, 2009	1.00	Initial public release.

Contents

NOTE: This page intentionally left blank.

1 Introduction

This document is designed to assist those wishing to use the SR-IOV capabilities within the Intel® 82576 Gigabit Ethernet Controller. The document is intended to be a companion to the *Intel® 82576 Gigabit Ethernet Controller Datasheet*. The datasheet documents the controller. It includes descriptions of the virtualization features and registers for the control of those features. See the listing at: http://developer.intel.com/products/ethernet/index.htm?iid=nc+ethernet.[1]

The document assumes knowledge of Ethernet device drivers and discusses SR-IOV portions of the Xen Physical Function driver and the Linux* Virtual Function driver.

1.1 SR-IOV Overview

Current I/O Virtualization techniques have their advantages and disadvantages. None are based upon any sort of industry standard.

The industry recognizes the problems of the alternative architectures and is developing new devices that are natively shareable. These devices replicate the resources necessary for each VM to be directly connected to the I/O device so that the main data movement can occur without Hypervisor involvement.

Natively shared devices will typically provide unique memory space, work queues, interrupts, and command processing for each interface they expose, while utilizing common shared resources behind the host interface. These shared resources still need to be managed and will typically expose one set of management registers to a trusted partition in the Hypervisor.

[1] On the Developer picklist: select the device, then look for the datasheet.

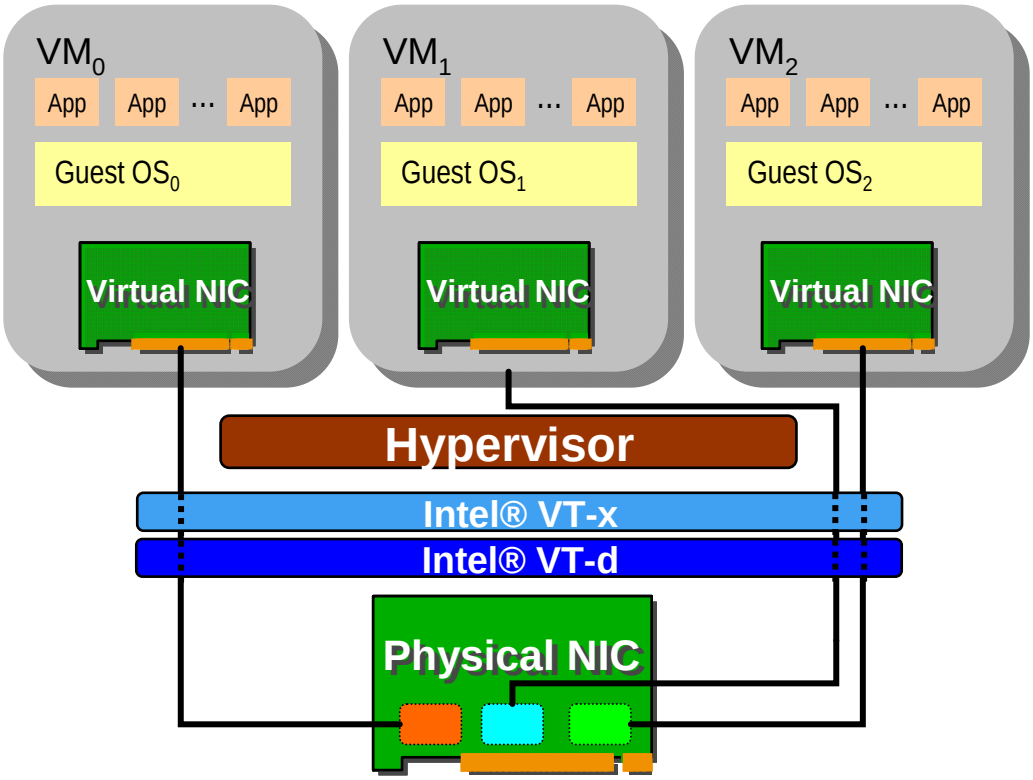

Figure 1 Natively Shared

By having separate work queues and command processing these devices are able to simultaneously receive commands from multiple sources and merge them together in an intelligent fashion before passing them to the secondary fabric (i.e. Ethernet or SAS link). The virtualization software no longer has to multiplex the I/O requests into a serial stream which reduces the Software overhead.

Natively shared devices can be implemented in numerous ways both standardized and proprietary. Since most of these devices are accessed over PCI, the PCI-SIG decided to define a standard approach to creating and managing natively shared devices through the *Single Root I/O Virtualization and Sharing* (SR-IOV) specification.

The PCI-SIG *Single Root I/O Virtualization and Sharing* (SR-IOV) specification defines a standardized mechanism to create natively shared devices.

1.1.1 SR-IOV Goal

The goal of the PCI-SIG SR-IOV specification is to standardize on a way of sharing an I/O device in a virtualized environment. This is accomplished by bypassing the Hypervisor's involvement in data movement by providing independent Memory Space, Interrupts, and DMA streams for each virtual machine. The SR-IOV architecture is designed to allow a device to support multiple virtual functions and much attention was placed on minimizing the hardware cost of each additional function.

The SR-IOV specification introduces two new function types:

- Physical Functions (PF): This is a full PCIe function that includes the SR-IOV Extended Capability which is used to configure and manage the SR-IOV functionality
- Virtual Function (VF): This is a lightweight PCIe function that contains all the resources necessary for data movement but have a carefully minimized set of configuration resources.

1.1.2 High Level Overview of PCI-SIG SR-IOV

The Direct Assignment method of virtualization provides very fast I/O, however it prevents the sharing of the I/O device. The SR-IOV specification provides a mechanism by which a Single Root Function (for example a single Ethernet Port) can appear to be multiple separate physical devices.

The SR-IOV capable device can be configured (usually by the Hypervisor) to appear in the PCI Configuration space as multiple functions, each with its own configuration space, complete with Base Address Registers(BARs).

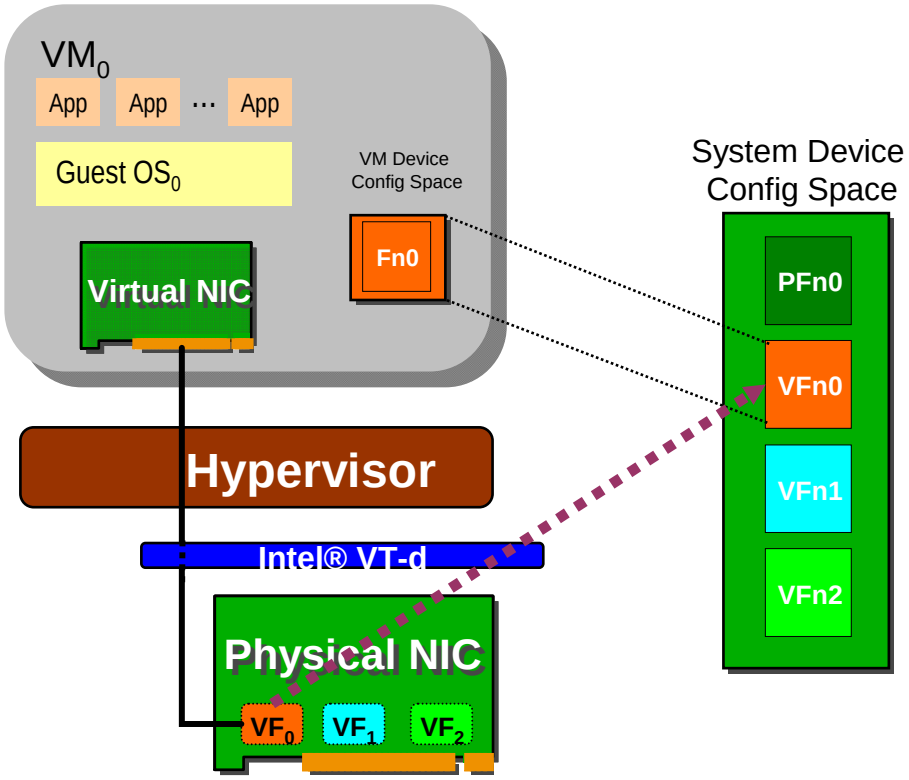

Figure 2 Mapping Virtual Function Configuration

The SR-IOV capable device provides a configurable number of independent Virtual Functions, each with its own PCI Configuration space. The Hypervisor assigns one or more

Virtual Functions to a virtual machine. Memory Translation technologies such as those in Intel® VT-d provide hardware assisted techniques to allow direct DMA transfers.

Note that the SR-IOV specification details how the PCI Configuration information is to appear, and it is different from standard PCI devices – the Hypervisor must know how to access and read this information in order to provide access to them from a VM. Refer to the PCI SIG SR-IOV specification for details.

2 High Level Architecture

This section gives a high-level overview of how the major pieces work together to provide SR-IOV functionality for Ethernet Connectivity with the 82576.

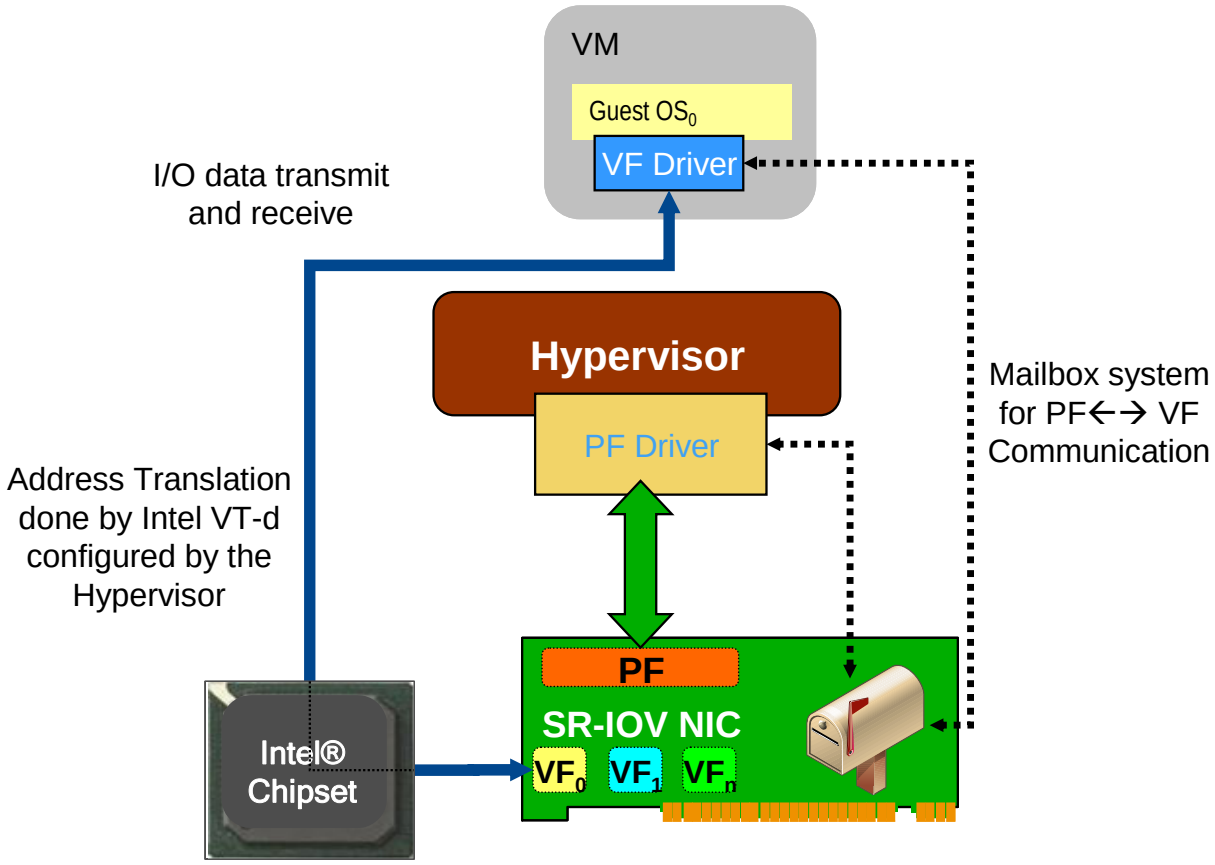

Figure 3 Intel SR-IOV Overview

2.1 Hypervisor

The Hypervisor (in this case Xen) must perform a number of activities in order to facilitate a SR-IOV capable environment.

2.1.1 PCI Configuration Access for the VM

As with all virtualized environments, in a SR-IOV system the Hypervisor must present a PCI

Configuration space to the Virtual Machine for the I/O device. In a shared I/O model, the Hypervisor usually presents a very simple Ethernet Controller to the VM. With SR-IOV, the Hypervisor presents the actual configuration space to a specific VF; granting the Virtual Function driver within the VM physical access to the VF resources.

2.1.2 PCI SIG SR-IOV PCI Configuration Area

Part of the SR-IOV specification involves the definition of how a SR-IOV capable device must advertise its SR-IOV capabilities and how the Virtual Function PCI Configuration information is to be stored and accessed. This is a new mechanism specific to SR-IOV, the Hypervisor must know how to read and parse this information. The Linux and Xen Kernels have been updated to be able utilize this new capability.

2.1.3 VT-d Setup/Configuration

VT-d enables direct assignment of an I/O device to a VM by remapping the DMA operations. The Hypervisor must configure VT-d with the I/O addresses to be remapped.

2.2 Physical Function Driver

Physical Function (PF) Driver is a specialized driver that manages global functions for the SR-IOV devices and is responsible for configuring shared resources. The PF Driver is specific to the Hypervisor and is expected to operate in a more privileged environment than a typical virtual machine driver. The PF Driver contains all the traditional driver functionality to provide access to the I/O resource for the Hypervisor; it can also be called upon to perform operations that impact the entire device. The PF Driver must be in a persistent environment, loaded before any Virtual Function Driver and unloaded (if necessary) after all Virtual Function drivers.

The Intel SR-IOV PF Driver contains all of the standard capabilities of any Intel® Ethernet Controller driver, such as VLAN filtering etc.

It has additional functionality specifically for SR-IOV utilization. Such as:

- MAC Address Generation for VF's
- Communication with VF Drivers via Mailbox system for:
    - Configuration of VLAN Filter by VF Driver
    - Configuration of Multicast Address by VF Driver
    - Configuration of Long Packet Maximum Length by VF Driver
    - VF Driver requests a reset of its resources

2.3 Virtual Function Driver

Standard device drivers (drivers that are not aware of the virtualized environment) have specific expectations regarding how it can control the device and how the device will behave. With virtualization, the standard driver typically communicates with an intermediary software layer that abstracts and emulates the underlying physical device. In

most cases, the driver is not aware that this indirection exists.

With directly assigned, natively shared devices, the expectations and device behavior changes. The Virtual Function interface does not include the full spectrum of PCIe controls and typically does not have direct control of shared components and resources, such as setting the Ethernet link rate. In this environment it is beneficial to enlighten or para-virtualize the Virtual Function Driver (VF Driver) so that it is aware of the virtualization environment.

These para-virtualized drivers will communicate directly with the hardware to perform data movement operations, they also realize the device is a shared resource and will rely on the services of the PF Driver to handle operations that can have global impact, like initialization and control of the secondary fabric.

The Intel Virtual Function driver is loaded the same way any other Intel Ethernet Device driver is loaded – based upon the device ID. The PCI Configuration information for the Intel Virtual Functions have a device ID identifying it as a virtual function, as such the appropriate driver is loaded.

2.4 PF Driver-VF Driver Communication

One necessary component for device sharing is the ability for the VF Driver to communicate with the PF Driver in order to request operations that have global effect. This communication channel needs this ability to pass messages and generate interrupts.

The SR-IOV specification does not define a mechanism for this communication path. Intel has chosen to implement this communication channel utilizing a set of hardware mailboxes and doorbells within the Intel Ethernet Controller for each Virtual Function.

The primary reason Intel chose to provide a hardware mailbox system was to ensure a communication mechanism that is Hypervisor independent and persistently available. However, a software based channel could also be used for this communication link if provided by the hypervisor subsystem. These channels will be vendor specific and require custom enabling to support. The hardware channel will always be available for backup communications unless specifically disabled by the hypervisor. Some hypervisors may wish to disable this capability for security concern reasons. At this time, not all Hypervisor vendors have committed to providing a software based messaging mechanism, therefore the reference drivers Intel is provide an API that utilizes the hardware channel.

3 Intel® 82576 Gigabit Ethernet Controller SR-IOV Support

This section will provide an overview of how the Intel® 82576 Gigabit Ethernet Controller handles such tasks as sorting packets for specific Virtual Functions, how resources are allocated for each VF and what resources the PF receives.

3.1 Queues

The Intel® 82576 has 16 Transmit and 16 Receive queues. They are generally referred to/thought of as queue pairs (1 Transmit and 1 Receive queue). This gives the Intel® 82576 16 queue pairs.

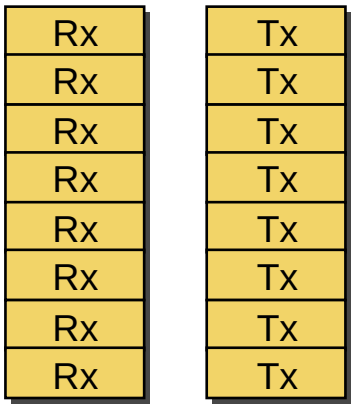

Figure 4 Queue Pair

3.2 Pools

A pool is a group of queue pairs for assigned to the same VF, used for transmit and receive.

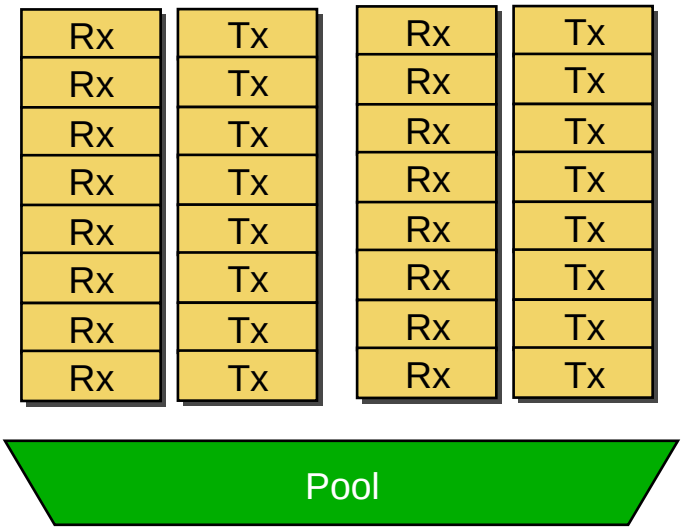

Figure 5 A 82576 Pool

The 82576 has 8 pools, with each pool having two queue pairs within it. That is 2 Transmit and 2 Receive queues assigned to each VF. For simplification purposes, for the remainder of this document, a pool will be represented similar to the representation in Figure 6.

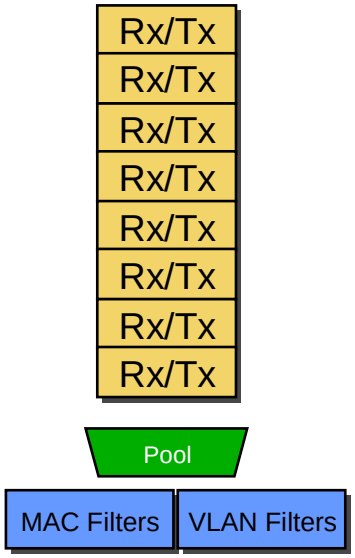

Figure 6 Pool Filters

Data is placed into the pool based upon L2 filters – MAC Address and VLAN Tag filters. Each pool can be configured for multiple MAC address and 32 VLAN Tags shared amongst all pools (both MAC addresses and VLAN tags).

3.3 Layer 2 Classifier/Sorter

The Intel® 82576 has within it what can be thought of as a small L2 switch that is

responsible for sorting packets based upon the destination MAC address or VLAN Tag.

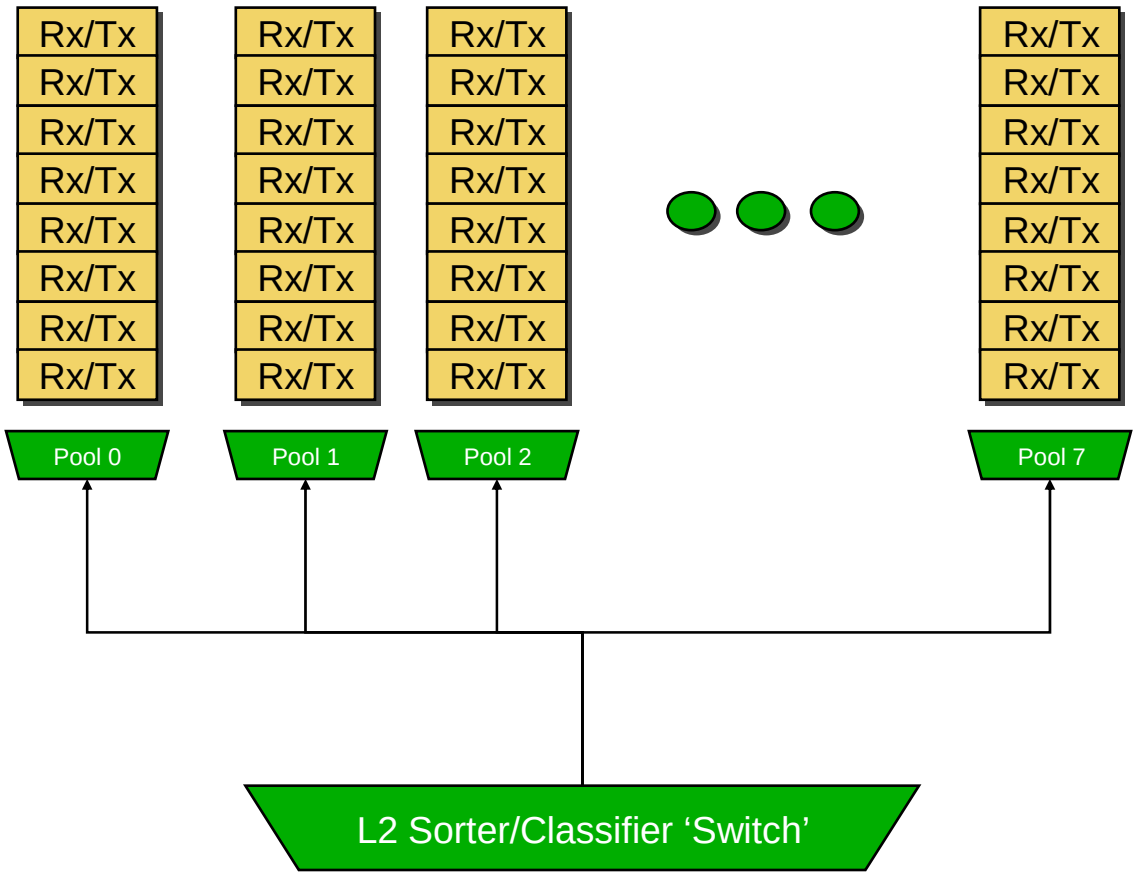

Figure 7 Layer 2 Sorter/Classifier/Switch within the Intel® 82576

When a packet comes into the Ethernet Controller it is examined to see if it matches one of the possible L2 filters, such as MAC and VLAN Tag filters. It is then forwarded to the appropriate queue based upon a filter match.

3.3.1 Additional Capabilities

3.3.1.1 Pool to Pool (VF to VF) Bridging

In a classic virtualized environment where a software based virtual switch exists in the Hypervisor, packets being transmitted from a VM are examined to see if the destination MAC address matches that of another VM running on the system.

If there is a match, the Hypervisor does not physically transmit the packet using the Ethernet Controller, but rather it sends it to the destination VM via the software switch.

One of the goals of SR-IOV is to remove the Hypervisor from the steps required to transmit and receive I/O. If the Hypervisor were to be required to examine all packets being

transmitted from a VF to see if it was destined for another VF, this goal of no Hypervisor involvement in data movement would not be achieved.

The 'switch' within the Intel® 82576 Ethernet Controller is capable of performing this bridging from one VF to another VF itself, in hardware, without the involvement of the Hypervisor.

3.3.1.2 Anti-spoofing

There are two types of anti-spoofing mechanisms supported by the Intel ® 82576.

3.3.1.2.1 MAC Address

The source MAC address of each outgoing packet can be compared to the MAC address the sending VM uses for packets reception. A packet with a non matching Source MAC address is dropped - thus preventing spoofing of the MAC address.

This feature is can be enabled on a per VF basis from the PF Driver (see Section 7.2).

3.3.1.2.2 VLAN Tag

If VLAN anti spoofing is enabled, a check is done to validate that sender VF is a member of the VLAN set in the packet.

This feature is can be enabled on a per VF basis from the PF Driver (see Section 7.3).

3.4 Example Receive Flow

This section provides a high level flow overview of how a packet may be received and sent to a VM.

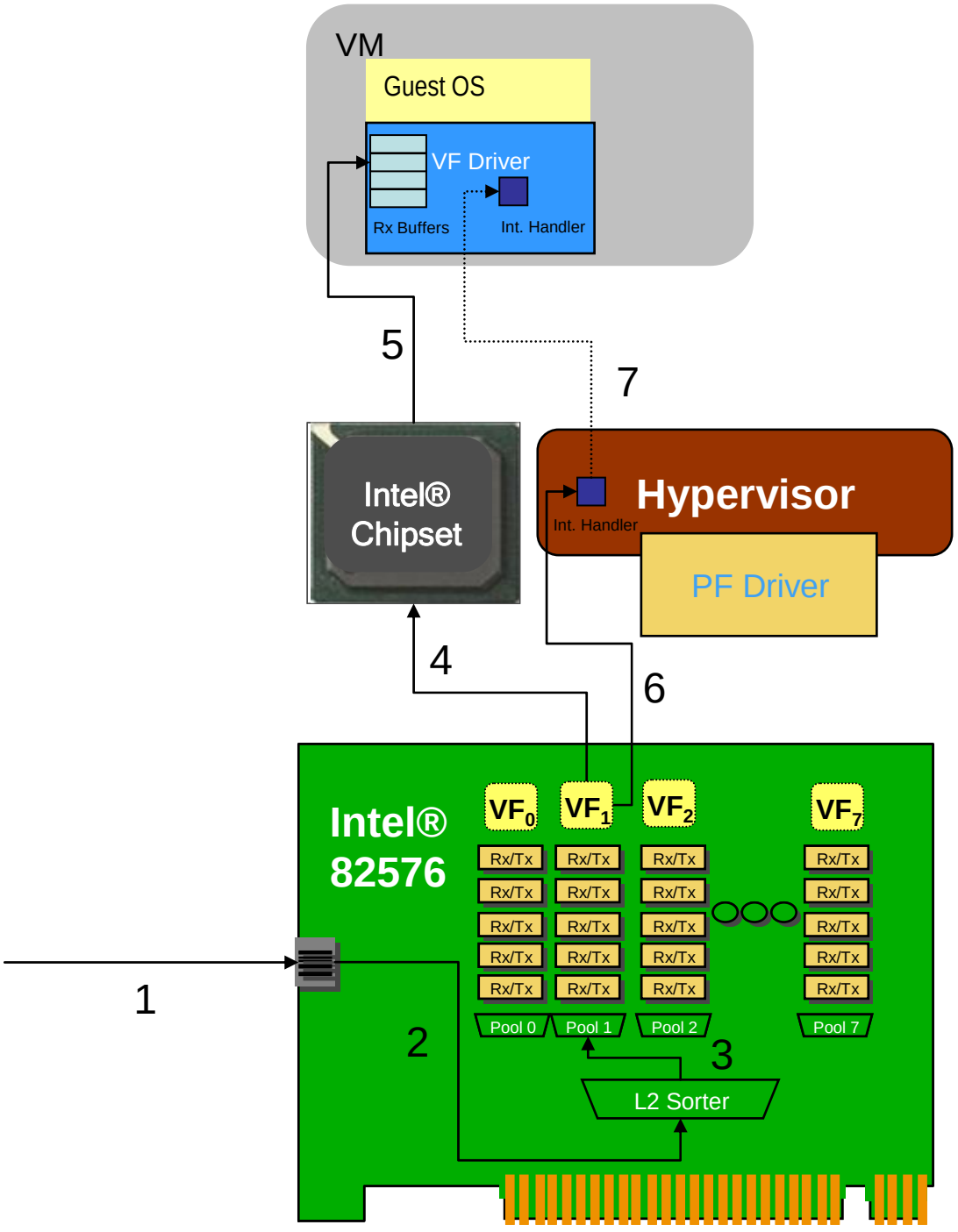

Figure 8 Packet Sent to a VM

Step 1 & 2: Packet Arrives, sent to the L2 Sorter/Switch

Step 3: Packet is sorted based upon destination MAC address, in this case matches Pool/VF 1.

Step 4: NIC initiates DMA action to move packet to VM

Step 5: DMA action hits the Intel Chipset, where VT-d (configured by the Hypervisor) performs the required Address Translation, for the DMA operation; resulting in the packet being DMA'd into the VM's VF Driver buffers.

Step 6: NIC posts MSI-X interrupt indicating a Rx transaction has been completed. This interrupt is received by the Hypervisor.

Step 7: The Hypervisor injects a virtual interrupt to the VM indicating a Rx transaction has been completed, the VM's VF Driver then processes the packet.

4 Mailbox Communication System

There are times when the VF Driver must communicate with the PF Driver in order to accomplish something not provided within the PCI resources exposed via the Virtual Function. An example of this is when the VF driver wants to define a VLAN filter – this capability is not exposed in the Virtual Function in any way, as such the VF driver cannot directly configure a VLAN filter.

The VF Driver can however make a request to the PF Driver to do this type of configuration on behalf of the VF.

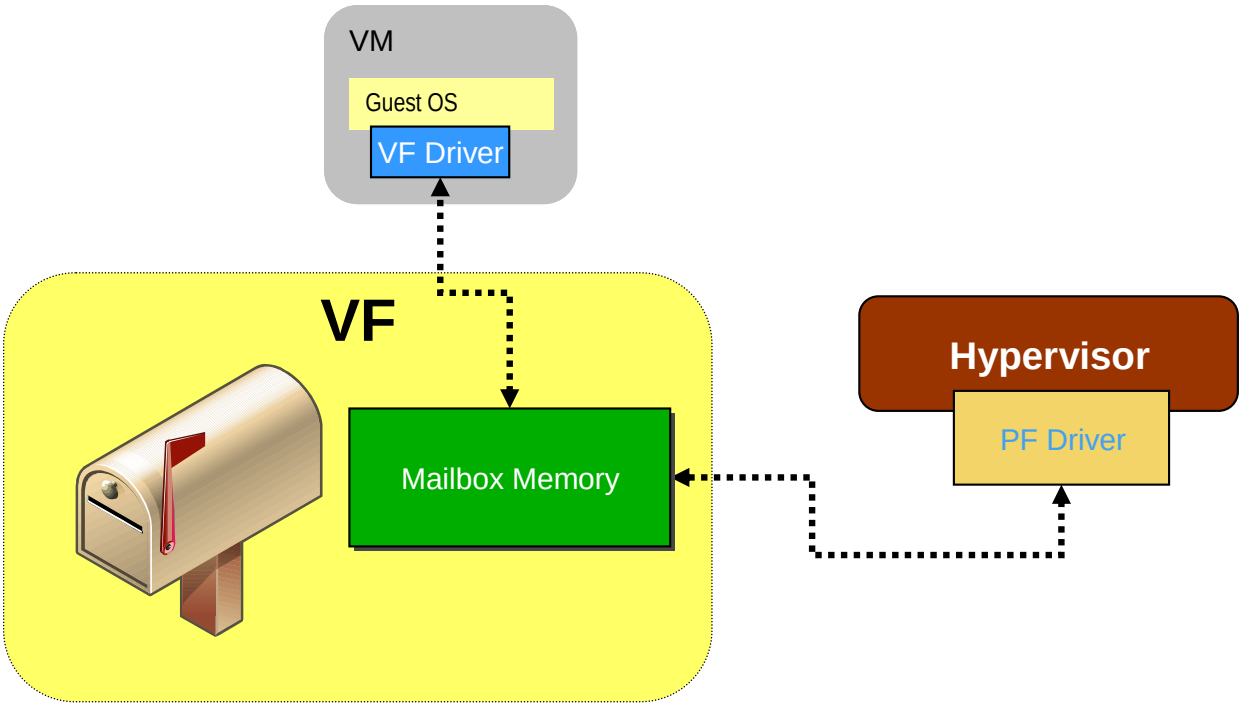

Figure 9 Mailbox Communication

4.1 Virtual Function Mailbox

When a VM is given access to a VF, part of the VF resources exposed is the Mailbox functionality. It is a fairly simple capability, composed of a buffer into which messages are written to or read from and a register (VFMailbox) providing a synchronization/semaphore between the PF and the VF when using the Mailbox.

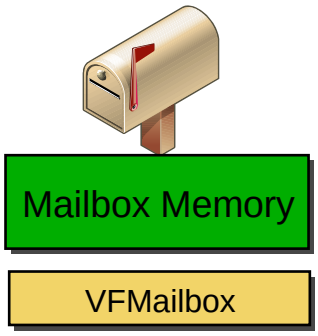

Figure 10 Mailbox from VF Perspective

The buffer is accessible by both the VF and the PF, in this way messages can be passed back and forth. Since the mailbox buffer and VFMailbox register are part of the VF resources, they are exclusive to a given VF – meaning that a VF driver for a given VF cannot write to any mailbox resources other than the one associated with the VF assigned to the VM where the VF Driver is running.

Further information regarding the mailbox buffer (Virtualization Mailbox Memory [VMBMEM]) and the VFMailbox register can be found in the Intel® 82576 Gigabit Ethernet Controller Datasheet.

4.1.1 Example Code

Within the igbvf sample driver code you can find the e1000_mbx.h and e1000_mbx.c files which create a set of APIs used by the igbvf driver for mailbox communication.

4.2 Physical Function Mailbox

The Physical Function driver has access to all of the VF mailboxes via the VMBMEM (Virtual Mailbox Memory) and PFMailbox (Physical Function Mailbox) register arrays.

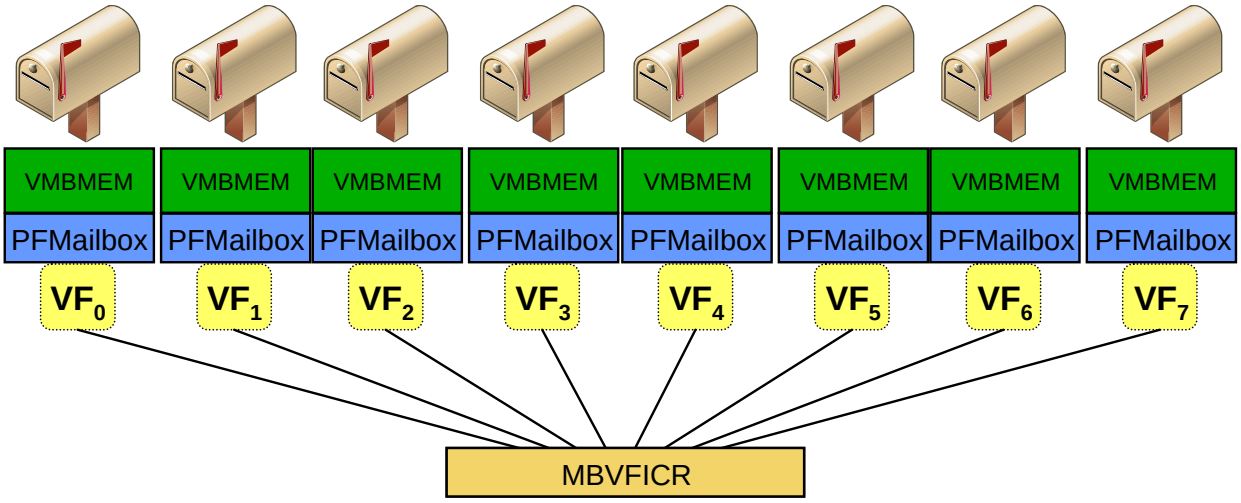

Figure 11 Mailbox from PF Perspective

When a VF driver writes a message to the VMBMEM buffer and sets the appropriate bits within its VFMailbox register, an interrupt is generated for the PF. The PF driver then goes and fetches the message and may, depending upon the message, acknowledge it.

At this time the PF Driver only sends one asynchronous message to a VF using the Mailbox; this is the E1000_PF_CONTROL_MSG message – it is simply a 'ping' message sent by the PF Driver to all VF Drivers (which respond with an ACK) to ensure VF Driver functionality.

Further information regarding the VMBMEM and the PFMailbox register can be found in the Intel® 82576 Gigabit Ethernet Controller Datasheet.

5 Virtual Function Driver

The sample Intel VF driver is a modified version of standard Intel igb Gigabit Ethernet driver. It is loaded based upon the Device ID presented to the VM as part of the PCI Configuration information from the Hypervisor. The Intel Virtual Functions have a Device ID that identifies them as Intel Virtual Functions, so that the appropriate driver can be loaded.

The Intel VF Driver can be logically split into three parts:

- The OS interface – that API's exposed to the VM Operating System
- I/O Operations – utilizing the SR-IOV capabilities for I/O without Hypervisor intervention
- Configuration Tasks – actions such as configure VLAN filtering that require communication with the PF Driver.

5.1 The Operating System Interface

As far as the Operating System in which the Intel VF Driver is running is concerned, the VF Driver is a standard driver, with all of the expected functionality. There is nothing specific to SR-IOV required.

All standard tools and API's should work just fine, including Ethtool and Netdev- though the output from these tools will differ from standard drivers. Examples of output from Ethtool running in a VM:

- ethtool – i

```
driver: igbvf
version: 0.9.5
firmware-version: N/A
bus-info: 0000:00:03.0
```

- ethtool – t

```
The test result is PASS
The test extra info:
Link test  (on/offline) 0
```

- ethtool – k

```
Offload parameters for eth3:
rx-checksumming: off
tx-checksumming: on
scatter-gather: on
tcp segmentation offload: on
udp fragmentation offload: off
generic segmentation offload: off
```

5.2 I/O Operations and Activities

The fundamental reason behind SR-IOV is to enable a driver within a VM direct access to PCI for I/O operations and to share that device amongst multiple VM's. The Intel VF Driver understands that it is running in a virtualized environment and has limited PCI resources available to it.

These are pretty standard I/O operations for an Intel Ethernet Controller. The only difference is the limited PCI resources available. The resources available include the required ability to Transmit and Receive Ethernet packets. Additional capabilities provided by the Intel® Virtual Function hardware include:

- Status Information, including:
    - Link Speed
    - Link Status
    - Duplex Mode
- Statistics include:
    - Good Packets Received Count
    - Good Packets Transmitted Count
    - Good Octets Received Count
    - Good Octets Transmitted Count
    - Multicast/ Broadcast Packets Received Count (one counter)
    - Good TX Octets loopback Count
    - Good TX packets loopback Count
    - Good RX Octets loopback Count
    - Good RX Packets loopback Count
- Function Level Reset (FLR)
- VLAN Tag Insertion
- Checksum Insertion

For a full description of capabilities provided by the VF hardware refer to the *Virtual Function Register Descriptions* section within the Intel® 82576 Gigabit Ethernet Controller Datasheet.

5.3 Actions taken via Mailbox system

The PCI resources exposed by the VF do not have provisions for all required activities the VF Driver may need to do, such as configuring a VLAN tag or Multicast address.

In such case the VF Driver utilizes the Mailbox system to pass a message to the PF Driver to that it in turn will perform the desired operation.

Currently defined actions performed utilizing this Mailbox mechanism includes:

- Resetting the VF
- Configuring the VF MAC Address
- Setting Multicast Address
- Setting VLAN Filter
- Setting Long Packet Maximum Length

The mailbox message IDs are defined within the e1000_mbx.h file for both the igb PF and igbvf VF Drivers.

5.3.1 Resetting the Virtual Function

```
Message ID: E1000_VF_RESET
```

When the VF Driver sends this message to the PF Driver, the PF Driver performs a Function Level Reset (FLR) on the VF resources.

An example of this can be found in the igbvf source file vf.c, within the e1000_reset_hw_vf() function:

```
msgbuf[0] = E1000_VF_RESET;
mbx->ops.write_posted(hw, msgbuf, 1, 0);
```

When the PF Driver receives this message, part of its acknowledgement is to send back the MAC address for the VF.

See the PF Driver Handler (Section 6.3.2.1) for further information.

5.3.2 Configuring a MAC Address

```
Message ID: E1000_VF_SET_MAC_ADDR
```

The VF Driver sends this message when it wishes to define its own MAC address (rather than use the default one defined by the PF at initialization of the VF).

An example of this can be found in the igbvf source file vf.c, within the e1000_rar_set_vf() function:

```
msgbuf[0] = E1000_VF_SET_MAC_ADDR;
memcpy(msg_addr, addr, 6);
ret_val = mbx->ops.write_posted(hw, msgbuf, 3);
```

See the PF Driver Handler (Section 6.3.2.2) for further information.

5.3.3 Setting Multicast Address

```
Message ID: E1000_VF_SET_MULTICAST
```

The VF Driver sends this message when it wishes to add a Multicast Address for filtering incoming packets on.

An example of this can be found in the igbvf source file vf.c, within the

e1000_update_mc_addr_list_vf() function:

```
u16 *hash_list = (u16 *)&msgbuf[1];
cnt = (mc_addr_count > 30) ? 30 : mc_addr_count;
msgbuf[0] = E1000_VF_SET_MULTICAST;

for (i = 0; i < cnt; i++) {
   hash_value = e1000_hash_mc_addr_vf(hw, mc_addr_list);
   hash_list[i] = hash_value & 0x0FFF;
}

mbx->ops.write_posted(hw, msgbuf, E1000_VFMAILBOX_SIZE, 0);
```

See the PF Driver Handler (Section 6.3.2.3) for further information.

5.3.4 Setting VLAN Filter

```
Message ID: E1000_VF_SET_VLAN
```

The VF Driver sends this message when it wishes to set a VLAN ID Address for filtering incoming packets on.

An example of this can be found in the igbvf source file vf.c, within the

e1000_rlpml_set_vf() function:

```
msgbuf[0] = E1000_VF_SET_VLAN;
msgbuf[1] = vid;

mbx->ops.write_posted(hw, msgbuf, 2, 0);
```

See the PF Driver Handler (Section 6.3.2.4) for further information.

5.3.5 Setting Long Packet Maximum Length

```
Message ID: E1000_VF_SET_LPE
```

The VF Driver can send the message to configure the Receive Long Packet Maximum Length (RLPML) for the VF.

An example of this can be found in the igbvf source file vf.c, within the

e1000_vfta_set_vf() function:

```
msgbuf[0] = E1000_VF_SET_LPE;
msgbuf[1] = max_size;

mbx->ops.write_posted(hw, msgbuf, 2, 0);
```

See the PF Driver Handler (Section 6.3.2.5) for further information.

6 Physical Function Driver

The Physical Function driver written for the Intel® 82576 Gigabit Ethernet Controller for the Xen Hypervisor is a modification of the standard Intel® igb driver. This driver is responsible for the physical function resources as well as handling some of the configuration of VF specific settings.

6.1 Initialization

When the driver is first probed (the igb_probe function found in igb_main.c), among the many tasks the driver does during initialization is to make a call to register itself as an SR-IOV device:

File: `igb_main.c`

Function: `__devinit igb_probe`

```
/* 82576 supports a maximum of 7 VFs in addition to the PF */
unsigned int num_vfs = (max_vfs > 7) ? 7 : max_vfs;

err = pci_enable_sriov(pdev, num_vfs);
if (!err)
    adapter->vfs_allocated_count = num_vfs;
```

This call registers the 82576 as a SR-IOV device, indicating support for 7 VFs. There are sufficient resources for 8 VF's, however if you were to allocate all resources for VF's the PF itself could not send and receive any Ethernet packets, as there would be no resources available for it to use.

There may likely be conditions where 7 VF's will be assigned to a VM while the last pool will be used for the traditional shared I/O model for additional VMs.

Recall that a lot of support for SR-IOV must be available from the Hypervisor itself. The build of Xen being used has been enhanced to support such requirements. The pci_enable_sriov() function is one of the enhancements that has been added to the kernel.

6.2 Default Configuration

There are a number of defaults configured during the initialization phase of the driver. These defaults include the number of VF's to be used, VF Offload configuration and VF MAC address assignments.

Intel investigated adding the ability to manipulate many of these settings using the sysfs infrastructure. This mechanism has to date not been accepted by the community, as such it is not available at this time.

At this time the only default that can be overridden is the number of VF's to use. This can be overridden using a command-line switch when the PF driver is loaded. The command-line parameter is as follows:

```
max_vfs=n
```

Where n can be up to 7.

Details regarding may of the defaults for the PF Driver are as follows:

6.2.1 Enabling VF to VF Bridging

The enablement of the VF to VF bridging is done within the igb_vmdq_set_loopback_pf() function within the e1000_82575.c file. This function manipulates the Loopback_en bit of the DTXSWC register to enable the bridging.

Refer to the Intel® 82576 Gigabit Ethernet Controller Datasheet for details regarding the DTXSWC register.

6.2.2 Default Pool

The default pool is the place where packets that are not sent to a specific VF (based upon VLAN or MAC address) are sent for processing. Essentially, this is the pool resource reserved for the PF.

Default Value: It is by default configured to be Pool 7, with 0 through 6 assigned to individual VFs.

Files: `igb_main.c`

Functions: `igb_configure_vt_default_pool`

```
vtctl = rd32(E1000_VT_CTL);
vtctl &= ~(E1000_VT_CTL_DEFAULT_POOL_MASK |
     E1000_VT_CTL_DISABLE_DEF_POOL);
vtctl |= pf_id << E1000_VT_CTL_DEFAULT_POOL_SHIFT;
wr32(E1000_VT_CTL, vtctl);
```

The driver manipulates the VT_CTL (0x0581C) register to configure the default pool, specifically the DEF_PL bits (9:7).

6.2.3 Replication Enable

This is the Broadcast and Multicast replication capability. It replicates incoming Broadcast and Multicast to all pools (so the Hypervisor does not have to).

File: `e1000_8275.c`

Function: igb_vmdq_set_replication_pf

```
u32 vt_ctl = rd32(E1000_VT_CTL);

if (enable)
   vt_ctl |= E1000_VT_CTL_VM_REPL_EN;
else
   vt_ctl &= ~E1000_VT_CTL_VM_REPL_EN;

wr32(E1000_VT_CTL, vt_ctl);
```

The driver manipulates the VT_CTL (0x0581C) register to configure the replication capability, specifically the Rpl_En bit (30).

6.2.4 Local Loopback

This is provided the capability for an individual VF to be able to send traffic and have it loop-backed to itself.

Default Value: By default Local Loopback is enabled for all VFs.

File: e1000_8275.c

Function: igb_vmdq_set_loopback

```
u32 dtxswc = rd32(E1000_DTXSWC);

if (enable)
   dtxswc |= E1000_DTXSWC_VMDQ_LOOPBACK_EN;
else
   dtxswc &= ~E1000_DTXSWC_VMDQ_LOOPBACK_EN;

wr32(E1000_DTXSWC, dtxswc);
```

The PF Driver manipulates the LLE field (bits 23:16) of the DTXSWC (0x3500) register to enable/disable local loopback for each VF.

6.2.5 Broadcast Accept Mode

Allows a VF to accept Broadcast Packets.

Default Value: By default Broadcast Accept Mode is enabled for all VFs.

File: igb_main.c

Function: _igb_set_vmolr

```
reg_data = rd32(E1000_VMOLR(vfn));
reg_data |= E1000_VMOLR_BAM | /* Accept broadcast */
      E1000_VMOLR_ROPE |      /* Accept packets matched in UTA */
      E1000_VMOLR_ROMPE |    /* Accept packets matched in MTA */
      E1000_VMOLR_AUPE |      /* Accept untagged packets */
      E1000_VMOLR_STRVLAN;   /* Strip vlan tags */
wr32(E1000_VMOLR(vfn), reg_data);
```

The driver manipulates the BAM field (bit 27) of the VM Offload Register (VMOLR 0x05AD0 + 4*n [n=0...7]) register to enable/disable Broadcast Accept Mode for each VF.

6.2.6 Accept Packets Matching UTA Table

Allows a VF to accept packets that match the Unicast address entries in the Unicast Table Array (UTA 0xA000).

Default value: is to accept.

File: igb_main.c

Function: _igb_set_vmolr

```
reg_data = rd32(E1000_VMOLR(vfn));
reg_data |= E1000_VMOLR_BAM | /* Accept broadcast */
      E1000_VMOLR_ROPE |      /* Accept packets matched in UTA */
```

```
      E1000_VMOLR_ROMPE |    /* Accept packets matched in MTA */
      E1000_VMOLR_AUPE |      /* Accept untagged packets */
      E1000_VMOLR_STRVLAN;   /* Strip vlan tags */
wr32(E1000_VMOLR(vfn), reg_data);
```

The driver manipulates the RPPE field (bit 26) of the VM Offload Register (VMOLR 0x05AD0 + 4*n [n=0...7]) register to enable/disable accepting packets from the UTA Table for each VF.

6.2.7 Accept Packets Matching MTA Table

Allows a VF to accept packets that match the Multicast address entries in the Multicast Table Array (UTA 0xA000).

Default value: is to accept.

File: `igb_main.c`

Function: `_igb_set_vmolr`

```
reg_data = rd32(E1000_VMOLR(vfn));
reg_data |= E1000_VMOLR_BAM | /* Accept broadcast */
      E1000_VMOLR_ROPE |      /* Accept packets matched in UTA */
      E1000_VMOLR_ROMPE |    /* Accept packets matched in MTA */
      E1000_VMOLR_AUPE |      /* Accept untagged packets */
      E1000_VMOLR_STRVLAN;   /* Strip vlan tags */
wr32(E1000_VMOLR(vfn), reg_data);
```

The driver manipulates the ROMPE field (bit 25) of the VM Offload Register (VMOLR 0x05AD0 + 4*n [n=0...7]) register to enable/disable accepting packets from the MTA Table for each VF.

6.2.8 Accept Untagged Packets Enable

Allows a VF to accept packets without a matching VLAN tag as long as MAC address matches.

Default Value: By default Accept Untagged Packets Mode is enabled VFs.

File: `igb_main.c`

Function: `_igb_set_vmolr`

```
reg_data = rd32(E1000_VMOLR(vfn));
reg_data |= E1000_VMOLR_BAM | /* Accept broadcast */
      E1000_VMOLR_ROPE |      /* Accept packets matched in UTA */
      E1000_VMOLR_ROMPE |    /* Accept packets matched in MTA */
      E1000_VMOLR_AUPE |      /* Accept untagged packets */
      E1000_VMOLR_STRVLAN;   /* Strip vlan tags */
wr32(E1000_VMOLR(vfn), reg_data);
```

The driver manipulates the AUPE field (bit 24) of the VM Offload Register (VMOLR 0x05AD0 + 4*n [n=0...7]) register to enable/disable Accept Untagged Packets Mode for each VF.

6.2.9 Strip VLAN Tag for Incoming Packets

Allows a VF to have the VLAN Tag stripped from incoming packets after it has passed the L2 filtering.

Default value: is enabled.

Files: `igb_main.c`

Functions: `_igb_set_vmolr`

```
reg_data = rd32(E1000_VMOLR(vfn));
reg_data |= E1000_VMOLR_BAM | /* Accept broadcast */
      E1000_VMOLR_ROPE |      /* Accept packets matched in UTA */
      E1000_VMOLR_ROMPE |    /* Accept packets matched in MTA */
      E1000_VMOLR_AUPE |      /* Accept untagged packets */
      E1000_VMOLR_STRVLAN;   /* Strip vlan tags */
wr32(E1000_VMOLR(vfn), reg_data);
```

The driver manipulates the STRVLAN field (bit 30) of the VM Offload Register (VMOLR 0x05AD0 + 4*n [n=0...7]) register to enable/disable the stripping of VLAN tags from incoming packets.

Note that VLAN insertions is configured as part of the transmit descriptor from within the VF Driver.

6.2.10 VF MAC Address Assignment

The PF Driver dynamically assigns each VF a MAC address using the random_ether_addr() function provided by the kernel. The following code snippet is an example:

File: `igb_main.c`

Function: `__devinit igb_probe`

```
for (i = 0; i < adapter->vfs_allocated_count; i++) {
   random_ether_addr(mac_addr);
   igb_set_vf_mac(adapter, i, mac_addr);
}
```

The igb_set_vf_mac() function configures a pool to have a specific MAC address filter. The VF gets its MAC address when it performs function level reset and sends the E1000_VF_RESET message to the PF via the mailbox. The PF in response sends the VF MAC address as part of the reset ack.

6.3 Mailbox Messages

Because the PCIe resources exposed to a VF are limited, there are some actions that the VF Driver must request the PF Driver perform on its behalf. This is accomplished using the mailbox system discussed in Section 3 to pass messages back and forth.

6.3.1 PF Driver to VF Driver Messages

Currently the PF Driver only sends a single asynchronous message to the VF Driver – all

other messages are in response to a message from the VF Driver.

6.3.1.1 E1000_PF_CONTROL_MSG

This message is sent from the PF Driver to each VF Driver at a regular interval as a sort of PING/Watchdog check to ensure that the VF Driver is functioning. Upon receipt of this message, the VF Driver simply acknowledges receipt.

This work is done within the igb_ping_all_vfs() function located in igb_main.c.

6.3.2 VF Driver to PF Driver Messages

There are a number of messages that the VF Driver passes to the PF Driver using the mailbox system. These are discussed from the VF Driver side in Section 5.3. This section will detail the actions taken when receiving a specific message from the VF Driver by the PF Driver.

All message IDs are defined in the e1000_mbx.h file as part of both the PF and VF Drivers. When a VF Driver sends a message via the Mailbox, the PF driver is notified via an interrupt, which it then services and calls the appropriate message handler.

6.3.2.1 E1000_VF_RESET

```
Message Handler Function: igb_vf_reset_msg() : igb_main.c
```

When this message is processed by the PF Driver, all settings regarding the VF are cleared except for the MAC address. Any offloads, VLAN Tags or multicast addresses are cleared.

The VF is put into an initialized state, ready for reception and transmission. The MAC address assigned to the VF is then returned to the VF Driver as part of an acknowledgement.

6.3.2.2 E1000_VF_SET_MAC_ADDR

```
Message Handler Function: igb_set_vf_mac_addr() : igb_main.c
```

The VF Driver sends this message if it wants to configure its own MAC address – overriding the one generated by the PF Driver (see Section 6.2.10). The VF receives its default MAC Address in response to the E1000_VF_RESET message.

If successful the PF Driver returns an ACK message(E1000_VT_MSGTYPE_ACK) with the default VF MAC address, otherwise a NACK message (E1000_VT_MSGTYPE_NACK).

6.3.2.3 E1000_VF_SET_MULTICAST

```
Message Handler Function: igb_set_vf_multicasts() : igb_main.c
```

The VF Driver sends a hash table of multicast addresses as part of this message. The PF Driver takes this hash table and adds it to the MTA Hash Table. Refer to the Intel® 82576 Gigabit Ethernet Controller Datasheet for information regarding the MTA Hash Table.

If successful the PF Driver returns an ACK message(E1000_VT_MSGTYPE_ACK), otherwise a NACK message (E1000_VT_MSGTYPE_NACK).

6.3.2.4 E1000_VF_SET_VLAN

```
Message Handler Function: igb_set_vf_vlan() : igb_main.c
```

When the PF Driver receives this message it attempts to update the global VLAN Tag and associate the VF with the specific Tag.

If successful the PF Driver returns an ACK message(E1000_VT_MSGTYPE_ACK), otherwise a NACK message (E1000_VT_MSGTYPE_NACK).

6.3.2.5 E1000_VF_SET_LPE

```
Message Handler Function: igb_set_vf_rlpml() : igb_main.c
```

This message is sent from the VF along with a size value indicating the Long Packet Size. The PF Drivers updates the VM Offload Register (VMOLR 0x05Ad0) with this size, taking care to add the additional size required for a VLAN Tag if VLANs are enabled for the VF.

If successful the PF Driver returns an ACK message(E1000_VT_MSGTYPE_ACK), otherwise a NACK message (E1000_VT_MSGTYPE_NACK).

7 Thoughts for Customization

The PF and VF Drivers written by Intel provide a robust set of capabilities, including MAC Address filtering, VLAN Tag Filtering, configuring MAC addresses for the VF etc. The Intel® 82576 Gigabit Ethernet Controller has a large number of capabilities that are not exploited by the reference driver provided by Intel.

This section highlights some features of the Intel® 82576 Gigabit Ethernet Controller customers may want to investigate providing support for.

7.1 Multicast Promiscuous Enable

The controller has provisions to allow each VF to be put into Multicast Promiscuous mode. The Intel reference driver leaves this un-configured.

This capability can be enabled/disabled by the manipulation of the MPE field (bit 28) of the VM Offload Register (VMOLR 0x05AD0 + 4*n [n=0...7]) register.

For a full description of the VM Offload Register refer to the Intel® 82576 Gigabit Ethernet Controller Datasheet.

7.2 MAC Anti Spoofing

This is the Anti Spoofing filtering based upon MAC address.

This capability can be added by setting the MACAS field (bits 0:7) of the DTXSWC (0x3500) register to enable/disable anti spoofing based upon MAC address for each VF.

If a spoof condition is detected, an interrupt can be configured. If such an interrupt occurs, the ICR register will have the IRC.WVB set indicating that there was an error with a VM.

The Wrong VM Behavior Register (WVBR 0x3553) will then indicate which VM or possibly VM's had an error. Finally the Last VM Misbehavior Cause register (LVMMC 0x3548) will have the MAC Spoof field (bit 0) set indicating that a spoof condition was found.

7.3 VLAN Tag Anti Spoofing

This is the Anti Spoofing filtering based upon VLAN Tag address.

This capability can be added by setting the VLANAS field (bits 15:8) of the DTXSWC (0x3500) register to enable/disable anti spoofing based upon VLAN Tag for each VF.

If a spoof condition is detected, an interrupt can be configured. If such an interrupt occurs, the ICR register will have the IRC.WVB set indicating that there was an error with a VM.

The Wrong VM Behavior Register (WVBR 0x3553) will then indicate which VM or possibly VM's had an error. Finally the Last VM Misbehavior Cause register (LVMMC 0x3548) will have the VLAN Spoof field (bit 1) set indicating that a spoof condition was found.

7.4 External Switch Loopback Support

One of the long term solutions for the packet switching is a mode where an external switch would do the loopback of VF to VF traffic and the NIC is responsible for the replication of multicast and broadcast packets only. In order to support this mode, the internal loopback mode should be disabled and received packets Source Address should be compared to the exact MAC addresses to check if the packet originated from a local source, so that the packet is not forwarded to the originator.

This mode is enabled by manipulating the Filter Local Packets (FLP) field (bit 27) of the Next Generation VMDq Control Register (0x0581C).

7.5 PF Reporting VF Statistics

As discussed in Section 5.2 each VF has several statistics:

- Statistics include:
    - Good Packets Received Count
    - Good Packets Transmitted Count
    - Good Octets Received Count
    - Good Octets Transmitted Count
    - Multicast/ Broadcast Packets Received Count (one counter)
    - Good TX Octets loopback Count
    - Good TX packets loopback Count
    - Good RX Octets loopback Count
    - Good RX Packets loopback Count

It may be desirable for some Hypervisors to know the statistics of each individual VF. The PF Driver has access to the statistics for each VF and could provide this information to a Hypervisor if so desired.

7.6 Additional Capabilities

The Intel® 82576 Gigabit Ethernet Controller has a number of additional capabilities that lend themselves to possible implementation/customization. These include:

- Storm Control
    - The PF Driver can set limits on broadcast or multicast traffic
- Transmit Bandwidth Allocation to VFs
    - Define minimum transmit bandwidth for individual VMs

8 Additional Materials

There are documents and materials available to someone investigating SR-IOV. For a complete list, see the *Intel® Ethernet SR-IOV Toolkit Overview* at http://developer.intel.com/products/ethernet/index.htm?iid=nc+ethernet.[2]

[2] On the Developer picklist: select the device, then look for the datasheet.