MSI and MSI-X Implementation

Govinda Tatti
Staff Engineer
Sun Microsystems

Contents

- Introduction
  - ✓ MSI and MSI-X interrupts
- MSI and MSI-X Hardware Implementation
  - ✓ SPARC Systems
  - ✓ x64 Systems
- MSI and MSI-X Software Implementation
  - ✓ Solaris Device Driver Software Programming Model
  - ✓ Example

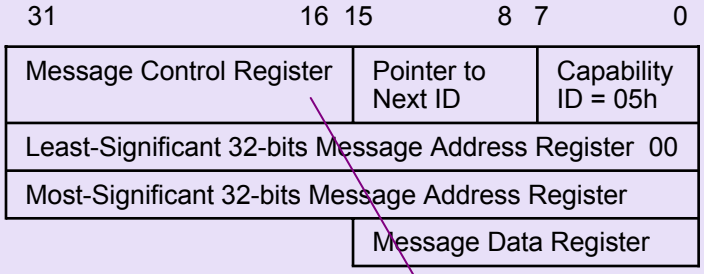
Introduction

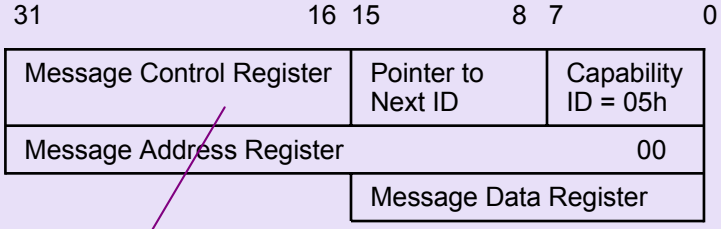
- In this presentation, we will be discussing the following:
  - ✓ Root Complex MSI and MSI-X implementation, some generic high-level hardware details for
    - SPARC systems
    - x64 systems
  - ✓ Software implementation, focusing mainly on Solaris Device Driver Software Programming Interfaces to use MSI and MSI-X interrupts
    - Register and Unregister MSI and MSI-X interrupts
    - Some debugging techniques

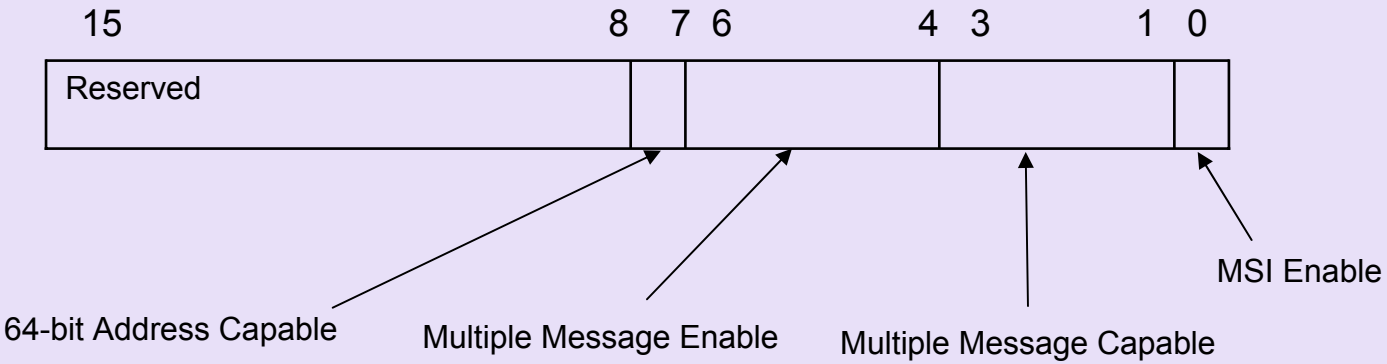
Message Signaled Interrupts

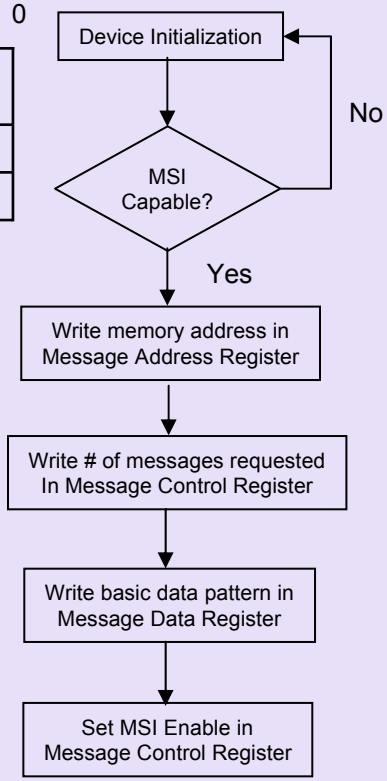
- Defined by all PCI Technology specifications
  - ✓ Implement as per ECN for PCI Express and PCI-X
- In-band messages implemented as a posted memory write to an address with a data value, both specified by software. The address and data values used are system specific
  - ✓ Required for PCI Express devices, optional for PCI devices
  - ✓ Edge-triggered mechanism
  - ✓ Maximum of 32 MSIs per function
  - ✓ Capability Structures in Configuration Space
  - ✓ A function may support INTx, MSI or MSI-X but only one mechanism can be used at any given time
- MSI has a number of distinct advantages over INTx
  - ✓ Larger number of interrupt vectors
  - ✓ Sharing of interrupt vectors is eliminated, simplifying interrupt servicing
  - ✓ Multi-function devices may have multiple interrupts per function

MSI Capability Register





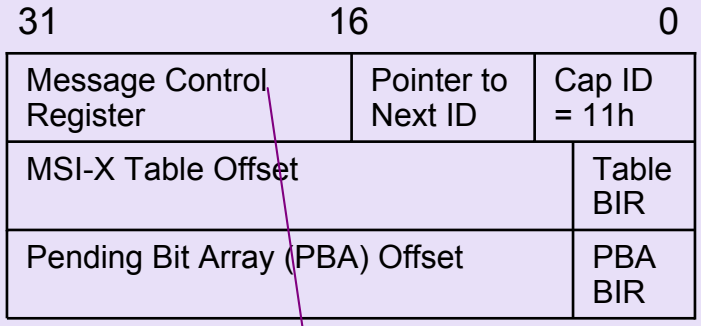


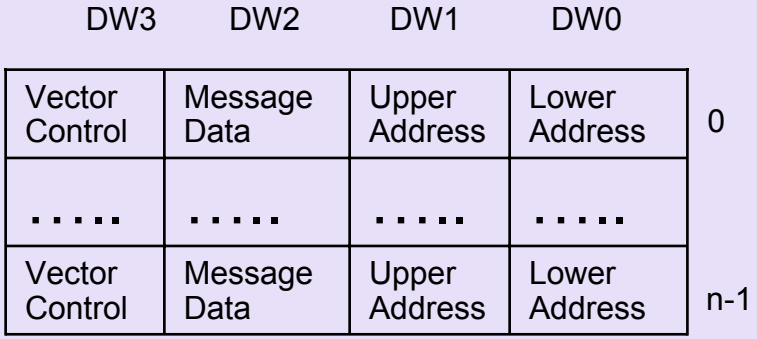


MSI-X (Extensions to MSI)

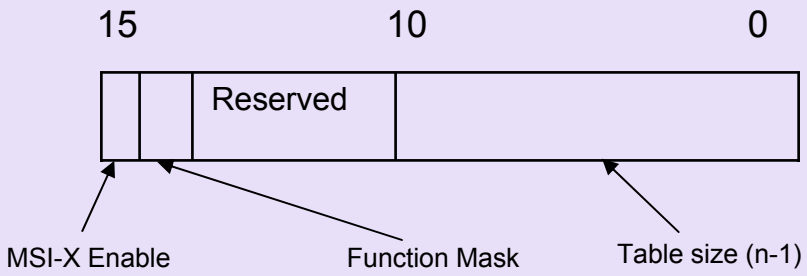
- MSI-X has the same features as MSI, the key differences are:
  - ✓ MSI-X support is optional
  - ✓ Maximum of 2048 MSI-Xs per function
  - ✓ MMIO region required for MSI-X tables and Pending Bit Arrays
  - ✓ Table entries contain unique address and data for each interrupt vector
  - ✓ Per function vector masking and per vector masking (optional for MSI)

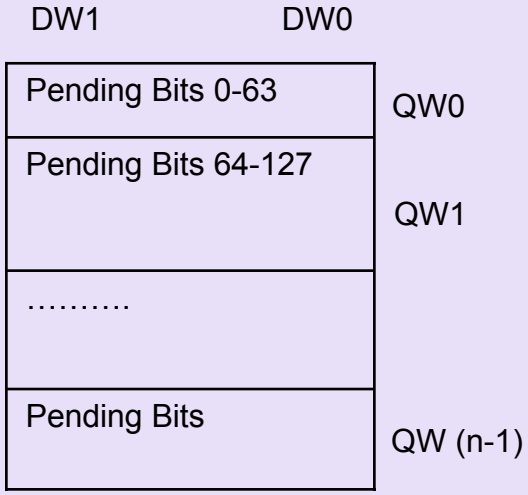
MSI-X Capability Register





MSI-X Table





MSI-X PBA Table

Example MSI and MSI-X Implementation on SPARC Systems

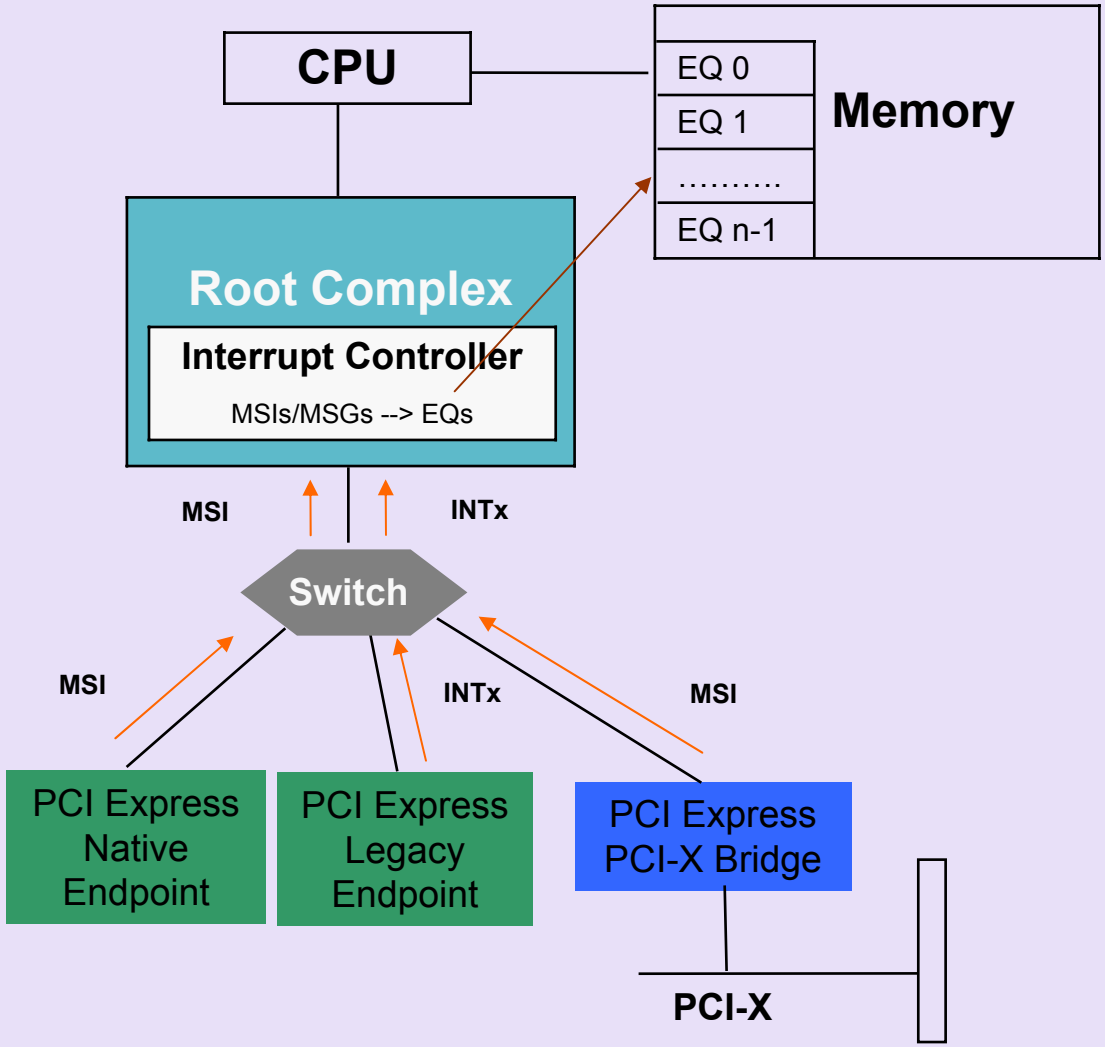
Introduction

- Sun's currently shipping PCI-Express Root Complex (RC) for SPARC systems supports MSI and MSI-X interrupts
  - ✓ Processed through Event Queues along with other PCI Express messages
  - ✓ MSI/MSI-X can use Solaris interrupt priorities from 1 through 12
  - ✓ INTx messages are handled similar to legacy PCI hardware interrupts

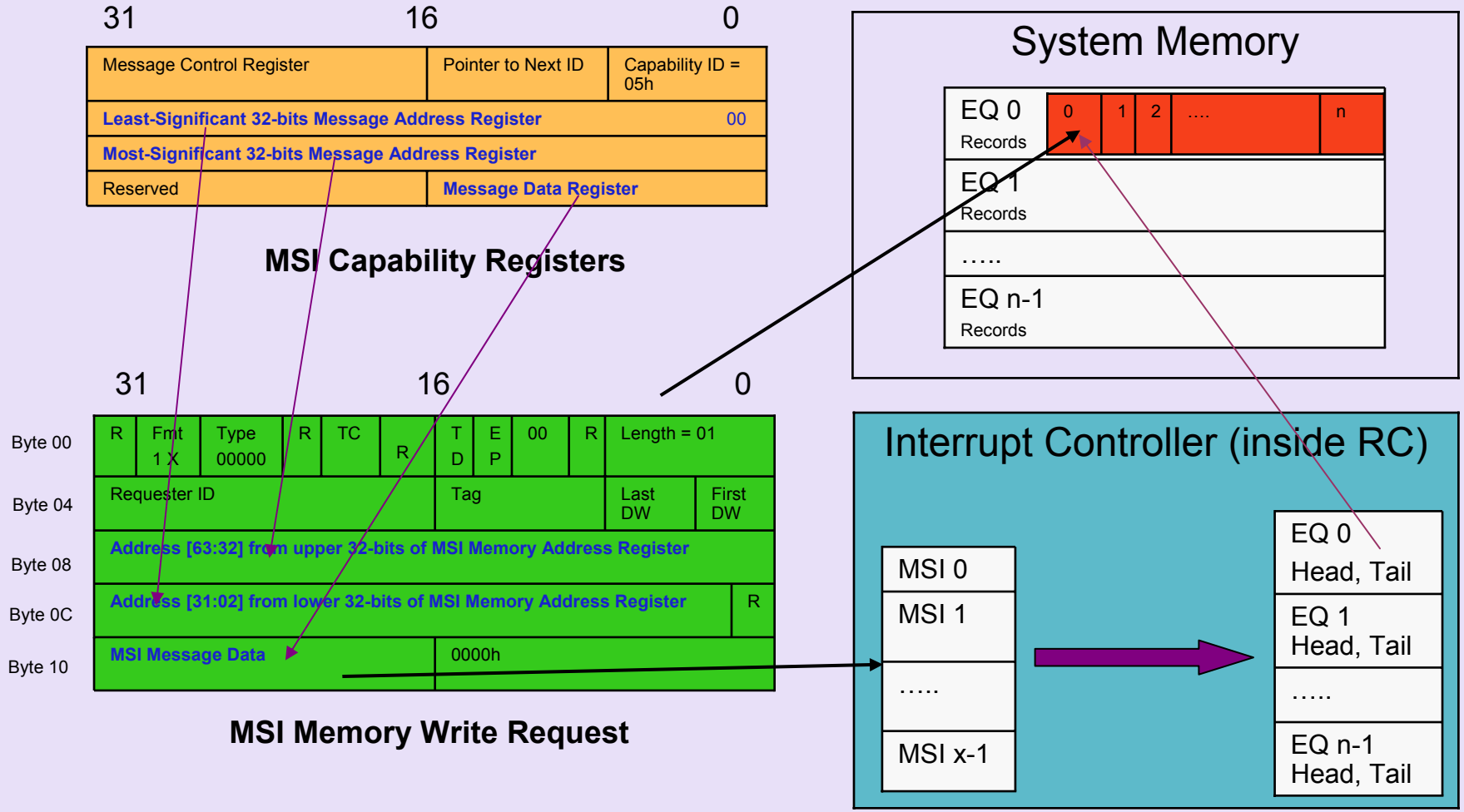
MSI and MSI-X processed through Event Queues

- Sun only supports MSI and MSI-X in their PCI Express Root Complexes for SPARC Systems
  - ✓ All PCI, PCI-X and PCI Express devices are supported
  - ✓ Both MSI32 and MSI64 are supported and are eligible for address remapping
  - ✓ MSI64 can also use direct physical addresses
- Event Queues
  - ✓ Event Queues (EQ) are ring buffers in system memory; hardware owns tail pointers and software owns head pointers. An entry in the EQ is called an Event Queue Record and each EQ has a fixed number of entries
  - ✓ An EQ can be mapped to a specific processor and generates only one outstanding host bus interrupt for one or more writes to the EQ
  - ✓ Each EQ write to memory is caused by the reception of a MSI or PCIe message from PCIe I/O bus
  - ✓ Multiple devices may share an EQ and each one of these devices may require a different interrupt priority level (IPL)
  - ✓ Solaris OS goes through each EQ record and invokes the appropriate device driver's ISR

MSI and MSI-X processed through Event Queues (cont.)



MSI and MSI-X processed through Event Queues (cont.)



Example MSI and MSI-X Implementation on x64 Systems

Introduction

- MSI and MSI-X implementation topics
  - ✓ Mapping to APIC vectors
  - ✓ Priority based implementation
    - This is Solaris based implementation choice, not bound by software in general and tries to map to IA-32's 16 priorities
  - ✓ PCI and IA architecture based implementation
- IOAPIC based only
  - ✓ One of 256 – 32 i.e. 232 vectors available
  - ✓ Looks at either IA MP Specification tables or ACPI tables or uses APIC to locate the correct interrupt pin of the right IOAPIC for a given device

IOAPIC Based Implementation

- MSI and MSI-X vectors mapped to APIC
  - ✓ Based on IA: Software Developer's Manual Chapter 8
  - ✓ Multiple MSIs and MSI-X supported with limitations
  - ✓ PCI Firmware Spec defines Local APIC memory mapped address used to program the MSI address
  - ✓ Limited to PCI Express devices only

    (Solaris implementation choice)

Solaris Priority Based Implementation

- 256 vectors divided up among 16 IPLs
  - ✓ Device priority is based on a function's class/subclass
  - ✓ Most priority levels get 16 vectors
  - ✓ Commonly used class/subclass like network or storage gets 32 vectors i.e. IPL 5 and 6. Other priorities like 1-3 and 7-9 are grouped together to share 16 vectors
  - ✓ Interrupts from a device with higher IPL get serviced before lower priority device interrupts
  - ✓ Instead of maintaining device to IPL mapping in OS, vector values are hardcoded with IPL
  - ✓ Pitfalls: limited number of vectors per IPL
  - ✓ Workaround: Allow a driver to specify a different IPL if system is running out of interrupts via driver.conf(4)

Concluding Remarks

- Useful future enhancements
  - ✓ Hardware support > 256 vectors
  - ✓ MSI address is shared and locked to one CPU; thus multiple MSIs of the same device cannot be bound to different CPUs
- MSI and MSI-X support limited to functions connected to PCI Express Root Complex topology on Solaris x64
  - ✓ Can be extended to PCI topology via tunables

Example MSI and MSI-X Implementation in Software

Introduction

- DDI is a set of Device Driver Interfaces between device drivers and the Solaris kernel
- MSI and MSI-X interrupts require
  - ✓ Ability to choose any interrupt types
  - ✓ Configuration and negotiation
  - ✓ Masking, pending, aliasing operations

Solaris DDI Interrupt Interfaces

- Solaris DDI Interrupt Interfaces used for MSI and MSI-X interrupts are classified as
  - ✓ System and Device interrupt capability interfaces
  - ✓ Interrupt Allocation, Free, Get and Set capability interfaces
  - ✓ Interrupt Add, Remove and Dup interfaces. Dup is used for aliasing a device's unused MSI-X vectors
  - ✓ Interrupt Enable and Disable interfaces including block operations
  - ✓ Interrupt Mask and Pending interfaces
  - ✓ Priority Management interfaces
- Primary argument for DDI Interrupt Interfaces
  - ✓ System/Device capability and Allocation interfaces takes a pointer to a device information structure (**dip**)
  - ✓ Other interfaces take a pointer to a DDI Interrupt handle structure (**ddi_intr_handle_t**)

Solaris DDI Interrupt i/fs (cont.)

Device/System Interrupt Capability Interfaces

```
int     ddi_intr_get_supported_types(dev_info_t *dip, int *intr_types)

int     ddi_intr_get_nintrs(dev_info_t *dip, int intr_type, int *nintrs)

int     ddi_intr_get_navail(dev_info_t *dip, int intr_type, int *navail)
```

Alloc/Free/Cap Interfaces

```
int     ddi_intr_alloc(dev_info_t *dip, ddi_intr_handle_t *h_array,
        int intr_type, int inum, int count, int *actual, int behavior)

int     ddi_intr_free(ddi_intr_handle_t handle)

int     ddi_intr_get_cap(ddi_intr_handle_t handle, int *cap_flags)

int     ddi_intr_set_cap(ddi_intr_handle_t handle,  int cap_flags)
```

Add/Remove/Dup Interfaces

```
int     ddi_intr_add_handler(ddi_intr_handle_t handle,
        ddi_intr_handler_t intr_handler, void *arg1, void *arg2)

int     ddi_intr_remove_handler(ddi_intr_handle_t handle)

int     ddi_intr_dup_handler(ddi_intr_handle_t from_handle, int to_inum, ddi_intr_handle_t *to_handle)
```

Solaris DDI Interrupt i/fs (cont.)

Enable/Block/Disable Interfaces

```
int     ddi_intr_enable(ddi_intr_handle_t handle)

int     ddi_intr_block_enable(ddi_intr_handle_t *h_array,  int count)

int     ddi_intr_disable(ddi_intr_handle_t handle)

int     ddi_intr_block_disable(ddi_intr_handle_t *h_array, int count)
```

Masking/Pending Interfaces

```
int     ddi_intr_set_mask(ddi_intr_handle_t handle)

int     ddi_intr_clr_mask(ddi_intr_handle_t handle)

int     ddi_intr_get_pending(ddi_intr_handle_t handle, int *pending)
```

Priority Management Interfaces

```
int     ddi_intr_get_pri(ddi_intr_handle_t handle, int *intr_pri)

int     ddi_intr_set_pri(ddi_intr_handle_t handle, int intr_pri)

int     ddi_intr_get_hilevel_pri()
```

Example: MSI and MSI-X Registration

```
/*
 * bge_attach: (part of bge driver attach)
 * Reference:  www.opensolaris.org/os/community/device_drivers
 * http://cvs.opensolaris.org/source/xref/on/usr/src/uts/common/io/bge/
 */
if (ddi_intr_get_supported_types(bge_p->bge_dip, &bge_p->bge_intr_type) != DDI_SUCCESS) {
        cmn_err(CE_WARN, "ddi_intr_get_supported_types failed");
        return (DDI_FAILURE);
}

if (bge_p->bge_intr_type & DDI_INTR_TYPE_MSIX) {
        /* First try MSI-X, but fall back to MSI or FIXED if MSI-X fails */
        if (bge_add_intrs(bge, DDI_INTR_TYPE_MSIX) == DDI_SUCCESS) {
                cmn_err(CE_NOTE, "Using MSI-X interrupt type");
                bge_p->bge_intr_type = DDI_INTR_TYPE_MSIX;
                return (DDI_SUCCESS);
        }
        cmn_err(CE_NOTE, "MSI-X registration failed, try MSI interrupts");
}

if (bge_p->bge_intr_type & DDI_INTR_TYPE_MSI) {
        /* Next try MSI, but fall back to FIXED if MSI fails */
        if (bge_add_intrs(bge, DDI_INTR_TYPE_MSI) == DDI_SUCCESS) {
                cmn_err(CE_NOTE, "Using MSI interrupt type");
                bge_p->bge_intr_type = DDI_INTR_TYPE_MSI;
                return (DDI_SUCCESS);
        }
        cmn_err(CE_NOTE, "MSI registration failed, trying FIXED interrupts");
}

if (bge_p->bge_intr_type & DDI_INTR_TYPE_FIXED) {
        if (bge_add_intrs(bge, DDI_INTR_TYPE_FIXED) == DDI_SUCCESS) {
                cmn_err(CE_NOTE, "Using FIXED interrupt type");
                bge_p->bge_intr_type = DDI_INTR_TYPE_FIXED;
                return (DDI_SUCCESS);
        }
        cmn_err(CE_WARN, “Interrupt registrations failed");
        return (DDI_FAILURE);
}
```

Example: MSI and MSI-X Registration (cont.)

```
/*
 * bge_add_intrs(bge_t *bge, int intr_type)
 *
 * Register MSI-X/MSI/FIXED interrupts
 */

/* Get number of interrupts */
If ((ddi_intr_get_nintrs(bge_p->bge_dip, intr_type, &count) != DDI_SUCCESS) || (count == 0)) {
            cmn_err(CE_WARN, "ddi_intr_get_nintrs() failed, count %d", count);
            return (DDI_FAILURE);
}

/* Get number of available interrupts */
If ((ddi_intr_get_navail(bge_p->bge_dip, intr_type, &avail) != DDI_SUCCESS) || (avail == 0)) {
            cmn_err(CE_WARN, "ddi_intr_get_navail() failed, avail %d", avail);
            return (DDI_FAILURE);
}

if (avail < count) {
            cmn_err(CE_NOTE, “avail %d is less than count %d”, avail, count);
            count = avail;
}

/* Allocate an array of interrupt handles */
intr_size = count * sizeof (ddi_intr_handle_t);
bge_p->bge_htable = kmem_alloc(intr_size, KM_SLEEP);
flag = (intr_type == DDI_INTR_TYPE_FIXED) ?
   DDI_INTR_ALLOC_NORMAL:DDI_INTR_ALLOC_STRICT;

/* call ddi_intr_alloc() */
If ((ddi_intr_alloc(bge_p->bge_dip, bge_p->bge_htable, intr_type, 0, count,
    &bge_p->bge_intr_cnt, flag) != DDI_SUCCESS) || (bge_p->bge_intr_cnt == 0)) {
            cmn_err(CE_WARN, "ddi_intr_alloc() failed");
            return (DDI_FAILURE);
}
```

Example: MSI and MSI-X Registration (cont.)

```
/* Get priority for first interrupt, assume remaining are all the same */
if ((ddi_intr_get_pri(bge_p->bge_htable[0], &bge_p->bge_intr_pri) != DDI_SUCCESS) {
        cmn_err(CE_WARN, "ddi_intr_get_pri() failed");
        return (DDI_FAILURE);
}

/* Test for high level mutex */
if (bge_p->bge_intr_pri >= ddi_intr_get_hilevel_pri()) {
        cmn_err(CE_WARN, "bge_add_intrs: "High level interrupt not supported");
        return (DDI_FAILURE);
}

/* Initialize mutex used in interrupt handler */
mutex_init(&bge_p->bge_mutex, NULL, MUTEX_DRIVER, DDI_INTR_PRI(bge_p->bge_intr_pri)):

/* Call ddi_intr_add_handler() */
for (i = 0; i < bge_p->bge_intr_cnt; i++) {
        if (ddi_intr_add_handler(bge_p->bge_htable[i], bge_intr, (caddr_t)bge, (caddr_t)i)) != DDI_SUCCESS) {
                cmn_err(CE_WARN, "ddi_intr_add_handler() failed");
                return (DDI_FAILURE);
        }
}

if ((ddi_intr_get_cap(bge_p->bge_htable[0], &bge_p->bge_intr_cap)  != DDI_SUCCESS) {
        cmn_err(CE_WARN, "ddi_intr_get_cap() failed");
        return (DDI_FAILURE);
}

/* Enable interrupts */
If (bge_p->bge_intr_cap & DDI_INTR_FLAG_BLOCK) {
        (void) ddi_intr_block_enable(bge_p->bge_htable, bge_p->bge_intr_cnt);
} else {
        for (i = 0; i < bge_p->bge_intr_cnt; i++)
                (void) ddi_intr_enable(bge_p->bge_htable[i]);
}
```

Example: MSI and MSI-X De-Registration

```
/*
 * bge_rem_intrs(bge_t *bge)
 *
 * Unregister MSI-X/MSI/FIXED interrupts
 */

/* Disable all interrupts */
if (bge_p->bge_intr_cap & DDI_INTR_FLAG_BLOCK) {
        (void) ddi_intr_block_disable(bge_p->bge_htable, bge_p->bge_intr_cnt);
} else {
        for (i = 0; i < bge_p->bge_intr_cnt; i++)
                (void) ddi_intr_disable(bge_p->bge_htable[i]);
}

/* Call ddi_intr_remove_handler() */
for (i = 0; i < bge_p->bge_intr_cnt; i++) {
        (void) ddi_intr_remove_handler(bge_p->bge_htable[i]);
        (void) ddi_intr_free(bge_p->bge_htable[i]);
}

kmem_free(bge_p->bge_htable, bge_p->bge_intr_cnt * sizeof (ddi_intr_handle_t));
```

Debugging Tool

Sun Ultra 45:
echo ::interrupts | mdb -k

Device	Shared	Type	MSG #	State	INO	Mondo	Pil	CPU
uata#0	no	Fixed	---	enbl	0x4	0x784	4	0
tavor#0	no	MSI-X	9	enbl	0x21	0x7a1	6	0
bge#1	no	MSI	8	enbl	0x20	0x7a0	6	0
bge#0	no	MSI	7	enbl	0x1f	0x79f	6	0
ohci#5	no	MSI	6	enbl	0x1e	0x79e	9	0
ohci#4	no	MSI	5	enbl	0x1d	0x79d	9	0
ohci#3	no	MSI	4	enbl	0x1c	0x79c	9	0
ohci#2	no	MSI	3	enbl	0x1b	0x79b	9	0
ohci#1	no	MSI	2	enbl	0x1a	0x79a	9	0
ohci#0	no	MSI	1	enbl	0x19	0x799	9	0
ehci#1	no	Fixed	---	enbl	0x17	0x797	9	0
ehci#0	no	Fixed	---	enbl	0x1	0x781	9	0
pfb#0	no	Fixed	---	enbl	0x10	0x790	9	0
mpt#0	no	MSI	0	enbl	0x18	0x798	4	0
mi2cv#1	no	Fixed	---	enbl	0x3d	0x7bd	4	0
mi2cv#0	no	Fixed	---	enbl	0x3c	0x7bc	4	0
px#0	no	PCIe	27	enbl	0x3b	0x7bb	1	0
px#0	no	PCIe	51	enbl	0x3a	0x7ba	14	0
px#0	no	PCIe	49	enbl	0x39	0x7b9	14	0
px#0	no	PCIe	48	enbl	0x38	0x7b8	9	0
su#0	no	Fixed	---	enbl	0x8	0x7c8	12	0
power#0	no	Fixed	---	enbl	0x3	0x7c3	14	0
pcf8584#0	no	Fixed	---	enbl	0x1	0x7c1	4	0
px#1	no	PCIe	27	enbl	0x3b	0x7fb	1	0
px#1	no	PCIe	51	enbl	0x3a	0x7fa	14	0
px#1	no	PCIe	49	enbl	0x39	0x7f9	14	0
px#1	no	PCIe	48	enbl	0x38	0x7f8	9	0

x64:
echo ::interrupts | mdb -k

IRQ	Vector	IPL	Bus	Type	CPU	Share	APIC/INT#	ISR(s)
4	0xb0	12	ISA	Fixed	1	1	0x0/0x4	asyintr
9	0x80	9	PCI	Fixed	1	1	0x0/0x9	acpi_wrapper_isr
14	0x41	5	ISA	Fixed	3	1	0x0/0xe	ata_intr
19	0x83	9	PCI	Fixed	0	1	0x0/0x13	hci1394_isr
20	0x84	9	PCI	Fixed	1	2	0x0/0x14	nge_chip_intr, audio810_intr
21	0x21	1	PCI	Fixed	3	1	0x0/0x15	ohci_intr
22	0x20	1	PCI	Fixed	2	1	0x0/0x16	ehci_intr
23	0x40	5	PCI	Fixed	3	1	0x0/0x17	ata_intr
47	0x60	6	PCI	Fixed	1	1	0x1/0x17	nge_chip_intr
48	0x81	7		MSI	0	1	-	pepb_pwr_msi_intr
49	0x82	7		MSI	0	1	-	pepb_err_msi_intr
160	0xa0	0		IPI	ALL	0	-	poke_cpu
192	0xc0	13		IPI	ALL	1	-	xc_serv
208	0xd0	14		IPI	ALL	1	-	kcpc_hw_overflow_intr
209	0xd1	14		IPI	ALL	1	-	cbe_fire
210	0xd3	14		IPI	ALL	1	-	cbe_fire
224	0xe0	15		IPI	ALL	1	-	xc_serv
225	0xe1	15		IPI	ALL	1	-	apic_error_intr

References

- Specifications
  - ✓ PCI Local Bus Specification Revision 3.0
  - ✓ PCI Firmware Specification Revision 3.0
- Acknowledgement
  - ✓ David Kahn
  - ✓ Wesley Shao
  - ✓ Edward Gillett
  - ✓ Anish Gupta
  - ✓ Johnny Cheung

Thank you for attending the
PCI-SIG Developers Conference 2006.

For more information please go to
www.pcisig.com

MSI and MSI-X Implementation

Govinda Tatti
Staff Engineer
Sun Microsystems