

UDI Advanced Topics

DMA and Interrupts

<http://www.sco.com/forum1999/conference/developfast/f11>

Robert Lipe

UDI Development Team Lead

E-mail: robertl@sco.com



This is the sixth in a series of eight UDI presentations for SCO Forum 1999.

The sample driver presented in the UDI Tutorial illustrates the basic UDI infrastructure and Programmed I/O support, but many drivers also need to use DMA and interrupts. This session describes how these are handled in UDI.

Agenda

- **Background**
- Direct Memory Access (DMA)
- Interrupt Handling
- Q & A



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 2



Prerequisites

This presentation assumes knowledge of:

- UDI Architecture (F8)
- UDI Fundamental Types (F8)
- UDI Buffer Management (F9, F10)
- UDI Programmed I/O (F9, F10)



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 3



F8: UDI Architecture In-Depth

F9, F10: UDI Tutorial and Driver Walk-Through, parts 1 and 2

References

- **DMA and Interrupts are defined in the *UDI Physical I/O Specification***
- **Also covered in Physical I/O are:**
 - Programmed I/O (PIO)
 - Bus Bridge Metalanguage



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 4



Agenda

- Background
- **Direct Memory Access (DMA)**
- Interrupt Handling
- Q & A



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 5



DMA Overview

- **What is DMA?**
- **DMA constraints and DMA handles**
- **Mapping a buffer for DMA**
- **Scatter/gather lists**
- **Unmapping a buffer**
- **Allocating DMA-able memory**



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 6



What is DMA?

- **DMA is used by physical I/O devices to access mainstore memory directly w/o software intervention**
- **OS and driver must set up DMA mappings and tell device the addresses to use**
- **DMA is more efficient than PIO**



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 7



DMA Constraints

Overview

- **DMA constraints indicate addressing restrictions of a device's DMA engine**
 - e.g. 24-bit vs 32-bit vs 64-bit
 - Uses general constraints mechanism
 - » `udi_constraints_t` opaque handle
 - » `udi_constraints_attr_t` attribute code
- **Constraints are propagated at bind time**



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 8



DMA Constraints

Attribute Codes (udi_constraints_attr_t)

- **DMA constraints attributes include:**
 - UDI_DMA_ADDRESSABLE_BITS
 - UDI_DMA_ALIGNMENT_BITS
 - UDI_DMA_SCGTH_MAX_ELEMENTS
 - UDI_DMA_ELEMENT_LENGTH_BITS
 - many others...
- **Many specified as number of bits**



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 9



There are DMA constraints attributes to control:

- the whole transfer
- the whole scatter/gather list
- each scatter/gather segment
- each scatter/gather element
- special addressing restrictions (e.g. ISA boundaries)
- DMA access behavior (e.g. prefetching)

Since `udi_constraints_attr_t` is eight bits, quantities such as the alignment constraints are specified as a number of bits; for example, the alignment of the memory must be a multiple of $2^{\text{UDI_ALIGNMENT_BITS}}$. This also allows constraints > 32 bits to be expressed without using 64-bit data types.

DMA Constraints

Attribute Values

- **Each attribute has:**
 - Least restrictive value
 - Most restrictive value
 - Default value (usually same as least restrictive)
- **Driver can set, reset or combine attributes**



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 10



The spec defines the legal range of values for each attribute, which are considered the least and most restrictive, the default value, and whether or not zero is treated specially. For some attributes, there is no sense of least/most restrictive.

Since the constraints are stored in an opaque object, there is no way to get the value of an attribute, only to set, reset or combine.

DMA Constraints

Setting Attributes

- `udi_constraints_attr_set` sets one or more attributes to most restrictive of current and new values
- If attribute has no sense of more/less restrictive, new value is used
- `udi_constraints_attr_reset` resets one attribute to its default value



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 11



By using most restrictive of current and new values, this model allows a constraints object to be passed from driver to driver in a stack (starting with the bus bridge driver/mapper), which each driver applying its own constraints without worrying about the other drivers' constraints.

It also allows the environment to store the constraints internally in a wide range of structures (sparse or dense arrays, structures with or without extensions, ...).

DMA Constraints

Combining Constraints

- **Sometimes drivers must combine two sets of constraints from other drivers**
 - Typically done in multiplexors such as IP
- **udi_constraints_attr_combine combines two constraints objects using selected combine method:**
 - UDI_ATTR_MOST_RESTRICTIVE
 - UDI_ATTR_LEAST_RESTRICTIVE



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 12



udi_constraints_attr_combine can also be used to make a copy of an existing constraints object (by specifying a null value for one of the arguments), or to allocate a new all-default constraints object (by specifying null values for both constraints arguments).

Constraints Propagation

- **Parent driver calls**
`udi_constraints_propagate`
- **Environment copies constraints and sends child a** `udi_channel_event_ind`
- **Child driver adds its constraints using**
`udi_constraints_attr_set`



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 13



DMA Handles

- **Opaque handle used for DMA services**
 - Refers to a set of DMA mapping resources
 - Type is: `udi_dma_handle_t`
 - Constraints associated with each DMA handle
- **DMA handle typically re-used for multiple DMA mappings**



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 14



Preparing for DMA

udi_dma_prepare

- **Allocate DMA handle for buffer mapping**

```
void udi_dma_prepare (
    udi_dma_prepare_call_t *callback,
    udi_cb_t *gcb,
    udi_constraints_t constraints,
    udi_ubit8_t flags );

typedef void udi_dma_prepare_call_t (
    udi_cb_t *gcb,
    udi_dma_handle_t new_dma_handle );
```



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 15



Pre-allocating a handle allows environments to pre-allocate some DMA resources, speeding up the per-transfer `udi_dma_buf_map` call.

Mapping a Buffer for DMA

udi_dma_buf_map

- **Map a buffer into device's DMA space**

- Uses previously-allocated DMA handle

```
void udi_dma_buf_map (  
    udi_dma_buf_map_call_t *callback,  
    udi_cb_t *gcb,  
    udi_dma_handle_t dma_handle,  
    udi_buf_t *buf,  
    udi_size_t offset,  
    udi_size_t len,  
    udi_ubit8_t flags );
```

- **One mapping per handle at any one time**

- Same handle may be re-used after unmapping



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 16



Mapping a Buffer for DMA

Callback Function

- **Callback from `udi_dma_buf_map`:**

```
typedef void udi_dma_buf_map_call_t (  
    udi_cb_t *gcb,  
    udi_scgth_t *scgth,  
    udi_buf_t *new_buf,  
    udi_boolean_t complete,  
    udi_status_t status );
```



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 17



Scatter/Gather Lists

- **Lists (address, length) pairs**
 - Addresses from device viewpoint (card-relative)
 - IEEE 1212.1 compatible
- **Scatter/gather list itself visible from:**
 - Driver if UDI_SCGTH_DRIVER_MAPPED
 - Device if UDI_SCGTH_DMA_MAPPED
 - Or both...



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 18



There are three visibility cases:

The device directly supports the UDI (1212.1) format:

Use DMA-mapped; device reads scgth segments by DMA.

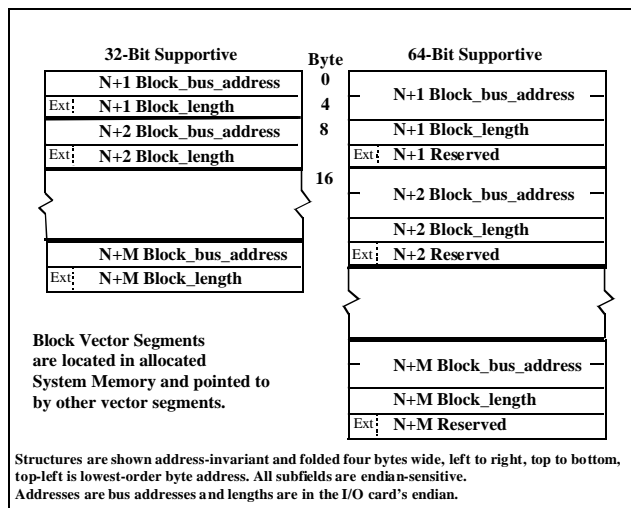
The driver can “munge” the format in place for device to read:

Use driver-mapped **and** DMA-mapped; driver reads/writes through pointer; device reads scgth segments by DMA.

The driver must allocate an entirely different structure:

Use driver-mapped; driver reads through pointer, sends to device through PIO or separately-allocated DMA.

Scatter/Gather Segments



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 19



A scatter/gather list contains one or more scatter/gather segments (aka block vector segments), which are contiguous arrays of scatter/gather elements (contiguous virtually if driver-mapped, contiguous in device address space if DMA-mapped).

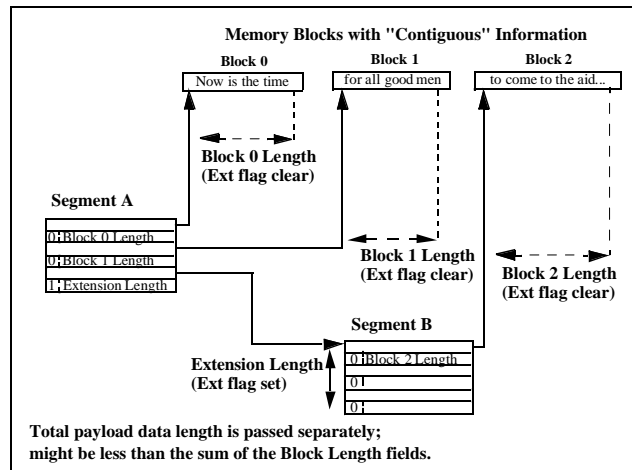
Each element contains a base address and a length, in bytes.

There are two formats: 32-bit and 64-bit. The element length (aka block_length) is 32-bit in both formats.

In both formats, the high bit of the last 32-bit word in the element is used as an Extension flag. If set, the memory addressed by the element is another scatter/gather segment, continuing the list, rather than a data element. (See the next slide for an example.)

For UDI, only the last element in a segment may have the Extension flag set.

Scatter/Gather Example



Scatter/Gather Endianness

- **Device-endian if DMA-mapped**
 - Driver sets `UDI_DMA_SCGTH_ENDIANNES`
- **Driver-endian if only driver-mapped**
- **If mapped for both, driver must do endianness conversions**
 - If `scgth_must_swap` is set
 - Helped by endianness utility functions/macros



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 21



The `UDI_DMA_SCGTH_ENDIANNES` constraints attribute can be set to `UDI_DMA_LITTLE_ENDIAN` or `UDI_DMA_BIG_ENDIAN`. It applies only to the scatter/gather segments, not to the data.

The `scgth_must_swap` flag is part of the `udi_scgth_t` structure which heads each scatter/gather list. (This struct is always driver-mapped.)

Unmapping a Buffer

- **Unmap a buffer when I/O complete**

```
void udi_dma_buf_unmap (  
    udi_dma_handle_t dma_handle );
```

- **Flushes caches if needed**
 - If needed during I/O use `udi_dma_sync`
- **DMA handle may be re-used for another mapping**



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 22



A lighter-weight alternative to `udi_dma_sync` is useful in some cases: `udi_dma_mem_barrier` inserts an ordering barrier but doesn't wait for transactions and caches to flush out.

These both affect the mapped data only. The scatter/gather list itself may be sync'ed with `udi_dma_scgth_sync`; this is only necessary when dual-mapped, after the driver finishes modifying the contents.

Allocating DMA-able Memory

- DMA-able memory for “long-term” shared control structures

```
void udi_dma_mem_alloc (
    udi_dma_mem_alloc_call_t *callback,
    udi_cb_t *gcb,
    udi_constraints_t constraints,
    udi_ubit8_t flags,
    udi_size_t len );
```

- Allocates memory and DMA handle, and maps the memory, all in one call



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 23



Values for flags:

UDI_DMA_OUT
UDI_DMA_IN
UDI_DMA_BIG_ENDIAN
UDI_DMA_LITTLE_ENDIAN
UDI_DMA_NEVERSWAP

Allocating DMA-able Memory

Callback Function

- **Callback from `udi_dma_mem_alloc`:**

```
typedef void udi_dma_mem_alloc_call_t (  
    udi_cb_t *gcb,  
    udi_dma_handle_t new_dma_handle,  
    void *mem_ptr,  
    udi_scgth_t *scgth,  
    udi_boolean_t must_swap );
```



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 24



Agenda

- Background
- Direct Memory Access (DMA)
- **Interrupt Handling**
- Q & A



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 25



Interrupt Handling Overview

- **Two phases to interrupt handling**
 - Interrupt attachment (and detachment)
 - Interrupt event handling
- **Two types of interrupt handling**
 - Interrupt preprocessing
 - Interrupt regions



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 26



Attaching Interrupt Handlers

- **Interrupt attachment channel operation:**

```
typedef struct {  
    udi_cb_t gcb;  
    udi_index_t interrupt_idx;  
    udi_ubit8_t max_event_pend;  
    udi_ubit16_t event_buf_size;  
    udi_pio_handle_t preprocessing_handle;  
} udi_intr_attach_cb_t;  
  
void udi_intr_attach_req (  
    udi_intr_attach_cb_t *cb );
```

- **Interrupt detachment channel operation:**

```
void udi_intr_detach_req (  
    udi_intr_attach_cb_t *cb );
```



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 27



Interrupt Preprocessing

- Preferred method for interrupt handling
- Driver passes PIO handle to bridge mapper during `intr_attach`
 - Contains a PIO transaction list
- When interrupts occur, bridge executes transaction list without having to call driver



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 28



Interrupt Preprocessing

(continued)

- **PIO trans list dismisses interrupt**
- **Partial or complete results sent via `udi_intr_event_ind` channel op to driver**
 - If trans list determines (shared) interrupt not generated by device, no op sent to driver
- **Drivers do *not* execute with interrupts disabled**



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 29



Interrupt Regions

- **If interrupt can't be dismissed with PIO trans list, driver can't use preprocessing**
- **Interrupt regions disable device interrupt while executing**
 - Specified by region attribute in static properties
 - Must be a secondary region
 - `udi_intr_event_ind` called with no preprocessing



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 30



Agenda

- Background
- Direct Memory Access (DMA)
- Interrupt Handling
- **Q & A**



F11: UDI Advanced Topics: DMA and Interrupts
© 1999 SCO All Rights Reserved - Slide 31



Presenter's Notes

F11: UDI Advanced Topics - DMA and Interrupts

August 19, 1999

