

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



Agenda

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



Prerequisites

This presentation assumes knowledge of:

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



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



Agenda

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



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**



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**



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**



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**



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**



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



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



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`



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**



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 );
```



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



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 );
```

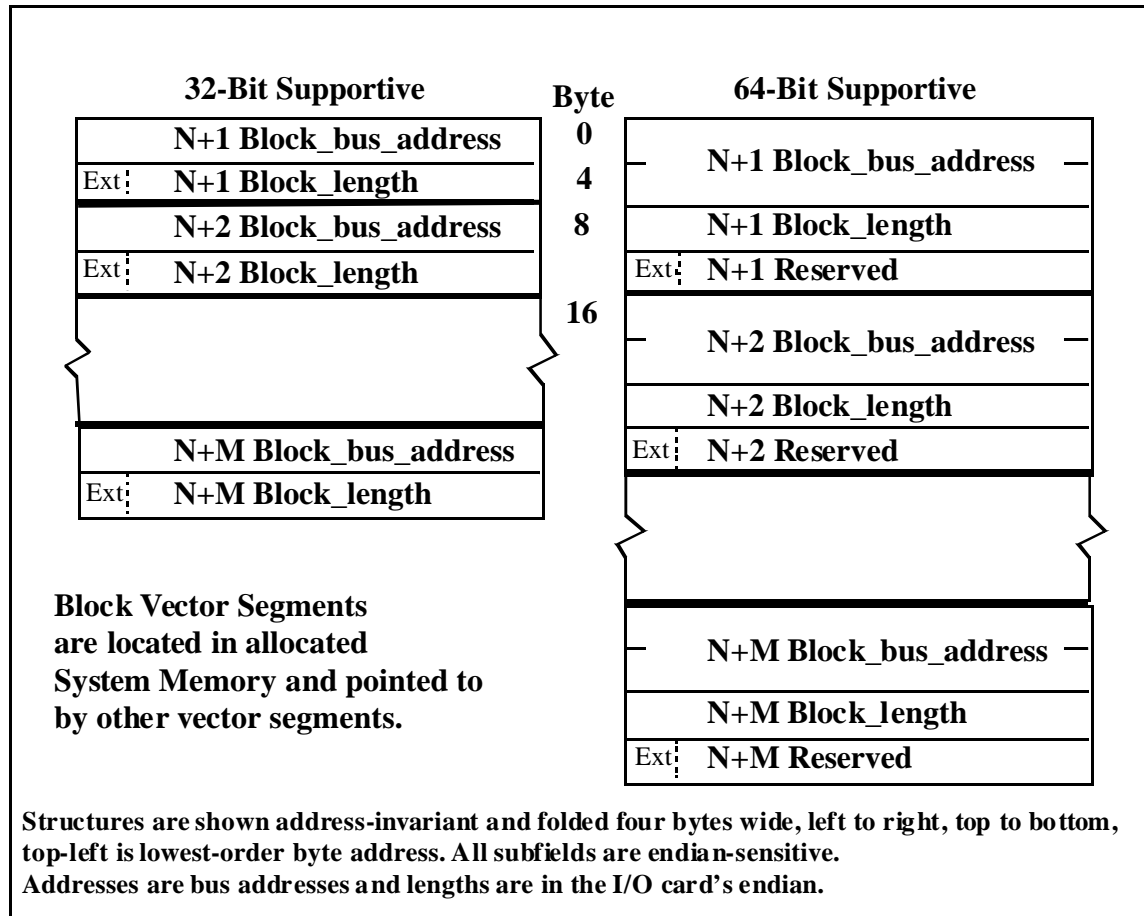


Scatter/Gather Lists

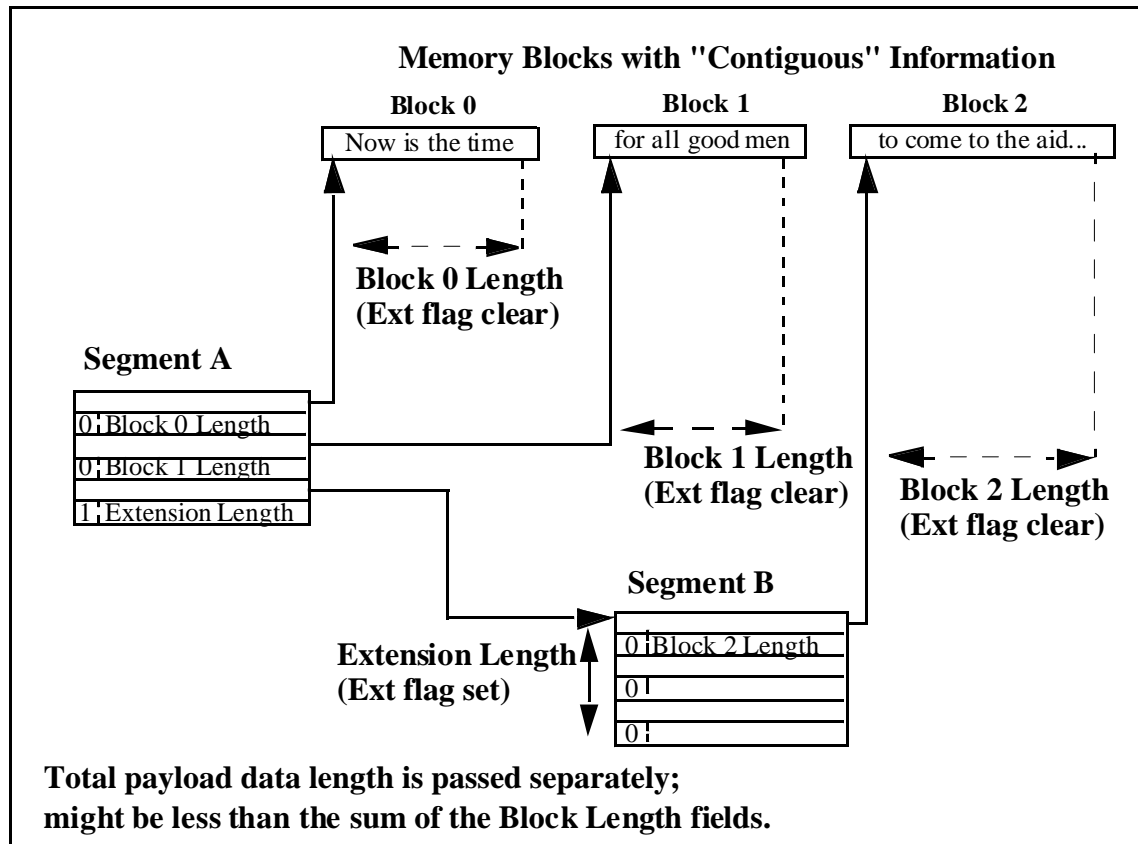
- **Lists (address, length) pairs**
 - Addresses from device viewpoint (card-relative)
 - IEEE 1212.1 compatible
- **Scatter/gather list itself visible from:**
 - Driver if UDI_SCGTH_DRIVER_MAPPED
 - Device if UDI_SCGTH_DMA_MAPPED
 - Or both...



Scatter/Gather Segments



Scatter/Gather Example



Scatter/Gather Endianness

- **Device-endian if DMA-mapped**
 - Driver sets `UDI_DMA_SCGTH_ENDIANNESS`
- **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



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**



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**



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 );
```



Agenda

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



Interrupt Handling Overview

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



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 );
```



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



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**



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



Agenda

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

