DESIGN DOCUMENT

DSP/BIOS™ LINK

RING IO

LNK 129 DES

Version 0.91

This page has been intentionally left blank.

IMPORTANT NOTICE

This page has been intentionally left blank.

TABLE OF CONTENTS

TABLE OF FIGURES

1 Introduction

1.1 Purpose & Scope

This document describes the design of the Ring IO component for DSP/BIOS™ LINK.

The document is targeted at the development team of DSP/BIOS™ LINK.

1.2 Terms & Abbreviations

DSPLINK	DSP/BIOS™ LINK
RingIO	Ring Input/Output
SHM	Shared Memory
SMA	Shared Memory Allocator
⚐	This bullet indicates important information. Please read such text carefully.
❑	This bullet indicates additional information.

1.3 References

1.	LNK 012 DES	DSP/BIOS™ LINK Link Driver Version 1.20, dated DEC 30, 2004
2.	LNK 128 DES	DSP/BIOS™ LINK IPS & Notify Version 0.10, dated DEC 15, 2005

1.4 Overview

DSP/BIOS™ LINK is runtime software, analysis tools, and an associated porting kit that simplifies the development of embedded applications in which a general-purpose microprocessor (GPP) controls and communicates with a TI DSP. DSP/BIOS™ LINK provides control and communication paths between GPP OS threads and DSP/BIOS™ tasks, along with analysis instrumentation and tools.

The RingIO component provides data streaming between GPP and DSP using ring buffer data as the transport.

This document provides a detailed description of the Ring Input/Output (RingIO) design.

2 Assumptions

The RingIO design makes the following assumption:

1. The hardware provides a shared region of memory. This region is accessible directly or indirectly to both the GPP and the DSP.
2. In the case of indirect access to the memory regions - the shared region of memory can be synchronized across processors through DMAs or some other mechanism.

For the purpose of this document, actual physically sharable memory architecture shall be used.

3 Constraints

None.

4 High Level Design

In a multiprocessor system having shared access to a memory region, an efficient mode of data transfer can be implemented, which uses a ring buffer created within the shared memory. The reader and writer of the ring buffer can be on different processors.

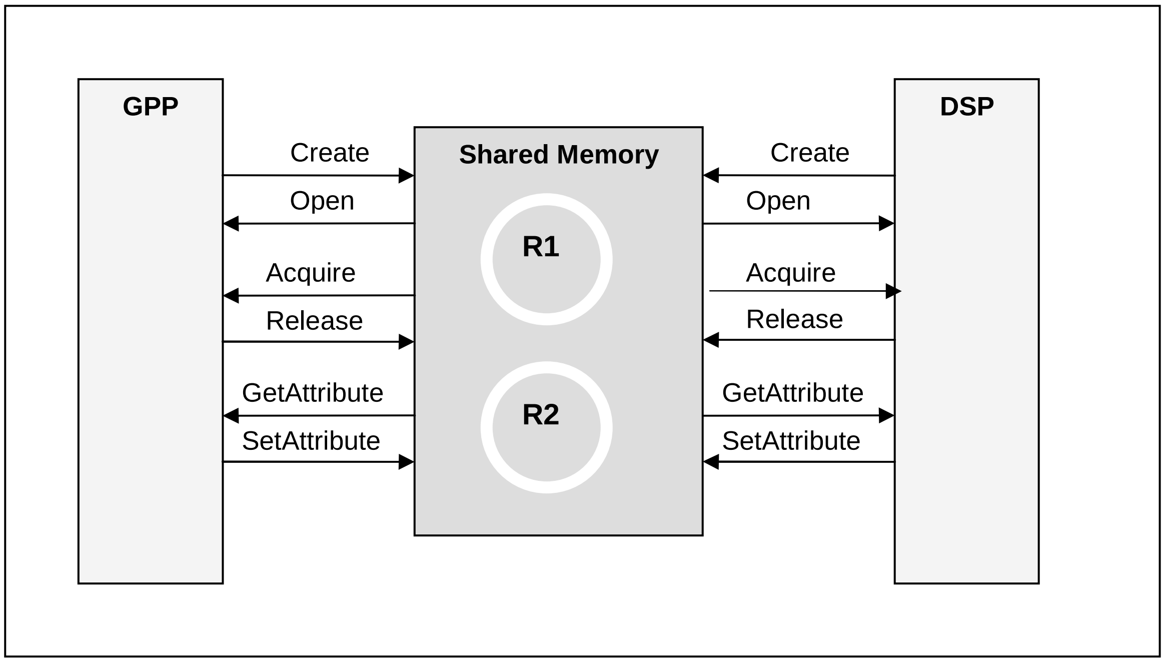

Figure 1. Basic architecture of system supporting RingIO transfer

The RingIO component on each processor shall provide the ability to create RingIO buffers within the memory provided by the application. The memory provided may be within the shared memory region (SHM) between two processors. If the RingIO buffer is created within shared memory, it shall be accessible to reader and writer present on the two processors between which the memory is shared. The application can obtain a handle to the RingIO through a call to open it, by passing the ID received from the call to create the RingIO.

The RingIO component shall provide the ability for the writer to acquire empty regions of memory within the data buffer. The contents of the acquired region are committed to memory when the data buffer is released by the writer.

The RingIO component shall provide the ability for the reader to acquire regions of memory within the data buffer with valid data within them. On releasing the acquired region, the contents of this region are marked as invalid.

The conceptual view of the RingIO is demonstrated in the Figure below. A single writer, single reader shall be supported for each RingIO buffer.

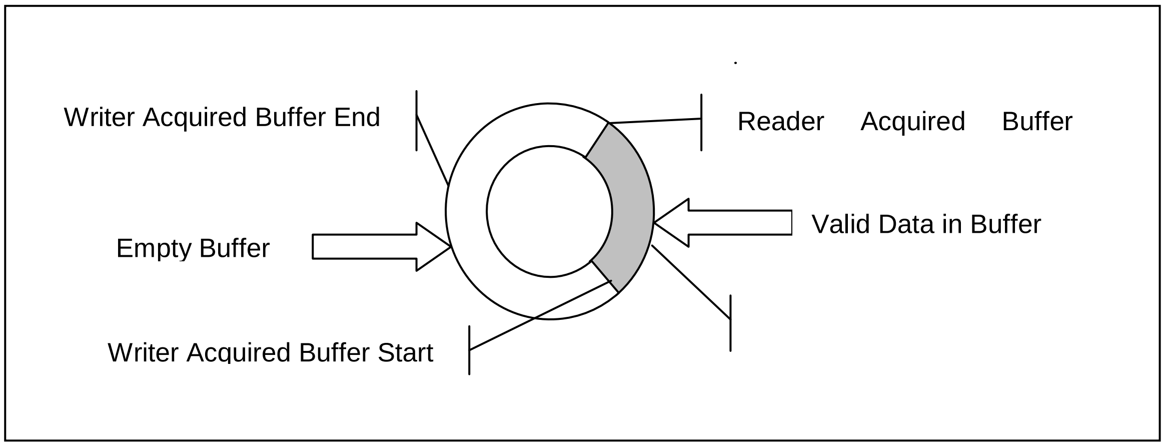

Figure 2. Conceptual view of the RingIO data buffer

The RingIO component shall also support APIs for enabling synchronous transfer of attributes with data. End of Stream (EOS), Time Stamps, Stream offset etc. are examples of such attributes and these shall be associated with offsets in the ring buffer.

4.1 Features

4.1.1 Generic features

- A client using RingIO is a single unit of execution. It may be a process or thread on the GPP or the DSP.
- The RingIO instance can be created between a client on the ARM and a client on the DSP or between two DSP clients.
- Either the reader or writer can create or delete the RingIO instance.
- The RingIO instance should be created in a shared memory region which can be accessed directly by both the reader and the writer.
- Both the reader and the writer need to open the RingIO instance and get a handle. Any data access on the RingIO instance should be made using these handles.
- Each RingIO can have a single writer client and a single reader client. A RingIO handle may not be shared between multiple clients on the GPP or DSP. For example, the following scenario is not permitted: One thread acquires from the RingIO, passes the buffer pointer to another thread, which then releases the buffer. This scenario is a multi-reader/writer scenario, which is not supported.
- Each RingIO instance is associated with a unique RingIO name. This RingIO name is specified while creating, opening and deleting the RingIO.
- The RingIO client can be closed only if there is no currently acquired data or attributes. If there is any unreleased data or attributes, they must be released or cancelled before the RingIO client can be closed.
- The RingIO can be deleted only when both reader and writer clients have successfully closed their RingIO clients.
- Each RingIO instance has an associated footer area, if configured. The foot-buffer can be configured to be of zero size if not required. If configured, the

foot-buffer is physically contiguous with the data buffer, and hence the memory size specified must be equal to (data buffer size + foot buffer size).

- The RingIO data and attribute buffer sizes must comply with any constraints imposed by the pool that they are specified to be allocated from. For example, for the Shared Memory Pool, the buffer sizes must be aligned to DSP cache line.

4.1.2 DSP cache-related information

- On the DSP-side, cache-related flags are provided to the writer and reader clients while opening the RingIO. These flags enable the user to get the maximum performance from the system and customize it for their own use. Separate cache flags are available for:
  - Control structures
  - Data buffer
  - Attribute buffer
- These flags indicate whether cache coherence is to be performed for the RingIO control structures, data buffer or attribute buffer. The flags need not be specified when opening the RingIO for the following application scenarios:
  - DSP-DSP RingIO
  - If RingIO control structures are specified to be placed into an internal memory pool, cache flag need not be specified for control structures.
  - If the RingIO data buffer is specified to be placed into an internal memory pool, cache flag need not be specified for data buffer.
  - If the RingIO attribute buffer is specified to be placed into an internal memory pool, cache flag need not be specified for attribute buffer.

4.1.3 Acquiring and releasing data

- The writer/reader client can acquire data buffers of any arbitrary size. RingIO does not maintain the acquired data as separate buffers, but as the complete acquired size.
  - Each buffer received from the acquire call is guaranteed to be a contiguous data buffer.
  - However, buffers received from multiple consecutive acquire calls may not be contiguous.
  - No assumption should be made that consecutively acquired buffers are contiguous in memory.
  - The writer/reader client can acquire multiple buffers and release the size completely, or in smaller chunks of varying sizes.
- The data is released into the RingIO by specifying the size to be released. Buffer pointers are not provided to the release call.
- As long as the size to be released does not exceed the total acquired size, the data can be released in any granularity. The sequence of release calls does not need to match the acquire calls.
- Cancel: Any acquired data that is not required can be cancelled back to the RingIO through the RingIO_cancel () API.

- The cancel call removes all acquired but un-released data from the RingIO for the calling client.
- In case of writer, any attributes that were set within this acquired but un-released region are also removed.
- In case of reader, any attributes that were removed within the acquired region are replaced back into the RingIO.

4.1.4 Writer

- The writer writes data into the RingIO data buffer by first acquiring a contiguous data buffer, writing data into the acquired buffer, and then releasing the filled up data to the RingIO.
- The behavior of acquire varies depending on the NEED_EXACT_SIZE specified while opening the writer client. The NEED_EXACT_SIZE flag indicates whether the writer always needs buffers only of a specific size, and buffers of lesser size are not acceptable.
  - NEED_EXACT_SIZE is TRUE
    - If the requested empty size is not available within the RingIO as a contiguous data buffer, error is returned.
    - If the requested empty size is not available till the end of the RingIO buffer, but is available from the top of the buffer, a wraparound occurs, and a contiguous buffer is returned from the top of data buffer.
  - NEED_EXACT_SIZE is FALSE: If the requested buffer size is not available, RingIO returns the amount of empty contiguous data buffer that is available till the end of the data buffer, with a status code indicating this.
- Five different types of notification mechanisms are supported. Details of the notification types are present in later sections.
- The writer can flush the data that it has written into the RingIO in two different modes. In the case of hard-flush, all data and associated attributes present in the RingIO will be removed. In the case of soft-flush, all data and associated attributes after the first readable attribute will be flushed, and the attribute is also removed.

4.1.5 Reader

- The reader reads data from the RingIO data buffer by first acquiring a contiguous data buffer, reading data from the acquired buffer, and then releasing the empty buffer to the RingIO.
- The behavior of acquire varies depending on the NEED_EXACT_SIZE specified while opening the reader client. The NEED_EXACT_SIZE flag indicates whether the reader always needs buffers only of a specific size., and buffers of lesser size are not acceptable.
  - NEED_EXACT_SIZE is TRUE
    - If the requested valid size is not available within the RingIO as a contiguous data buffer, error is returned.

        - If the requested empty size is not available till the end of the RingIO buffer, but is available from the top of the buffer, the behavior varies depending on whether a foot-buffer has been configured.
            - If non-zero size foot-buffer is configured, the required amount of valid data is copied from the top of the data buffer into the foot-buffer (assuming foot-buffer size is sufficient). A contiguous data buffer is then returned to the user as requested. Further acquires will happen from the specific offset from the top of the buffer. If foot-buffer size is not sufficient to return a contiguous data buffer of specified size, error is returned.
            - If foot-buffer is not configured, error is returned in this case.
    - NEED_EXACT_SIZE is FALSE: If the requested buffer size is not available, RingIO returns the amount of valid contiguous data buffer that is available till the end of the data buffer, with a status code indicating this. Foot-buffer is not used in this scenario.
- Five different types of notification mechanisms are supported. Details of the notification types are present in later sections.
- The reader can flush the data that is available from the RingIO in two different modes. In the case of hard-flush, all data and associated attributes present in the RingIO will be removed. In the case of soft flush, all data and associated attributes before the first readable attribute will be flushed

4.1.6 Attributes

Generic information

- Attributes are used to communicate in-band information from the writer to the reader.
- Typical attributes could be the EOS marker at the end of the stream that’s being written, or an attribute to indicate changes in the stream’s status.
- Attributes can be of two types
    - Fixed attributes: Fixed attributes have an attribute type and an optional parameter
    - Variable attributes: Variable attributes can be provided a data buffer as payload data in addition to the attribute type and the optional parameter. The attributes are copy-based. The information in writer-provided buffer is copied into the attribute buffer. The size of provided buffer in variable attributes must be a multiple of 4 bytes.

Setting attributes

- Attributes can be set by the writer only on a data buffer that has been acquired. This means that, if the writer has acquired a buffer of size x, attributes can be set at any of offset position between 0 and x (inclusive).
- The only exception to the above rule is if the writer wishes to set an attribute when no data has been acquired. In this case, the writer can set attributes at offset 0. Attempts to set attributes at any other offset are ignored, and the attributes get set at offset 0.

- When the writer writes attributes for the data buffer it has acquired, it should set attributes in the increasing order of buffer offsets. Setting attributes in any arbitrary order can lead to undefined behavior.
- The writer commits attributes to the attribute buffer when the associated write buffer is released. Any attributes set when writer has no acquired data are released immediately.

Getting attributes

- The attributes written by the writer should be "read" before the reader can read any more data after the offset at which the attribute is set.
- If the reader could not read data due to presence of attributes at the current read location, an error code mentioning the presence of an attribute is returned.
- When a variable attribute is being read, a valid buffer must be provided to the getAttribute function. The attribute information is copied into this application buffer.
- Attributes are removed from the attribute buffer when the reader releases the data buffer that contains the associated attributes or the writer flushes valid data which will clear associated attributes.
- Fixed and Variable attributes can be set and received using different APIs. In case a fixed attribute get function is called when a variable attribute is present, an error code is returned informing of the presence of the variable attribute.

4.1.7 Notification

The notification mechanism as well as other configuration parameters for the notification can be set by the reader or writer through an API call to set the notifier. Parameters that can be configured include the notification type, watermark, callback function, and fixed parameter to the callback function.

In case of GPP-DSP RingIO, the notifier can be set only if the other client is valid, i.e. it has opened the RingIO. This is because some information about the other client is required for setting the notifier for GPP-DSP RingIO.

Five different types of notification are supported:

1. **RINGIO_NOTIFICATION_NONE**: No notification is required.
2. **RINGIO_NOTIFICATION_ALWAYS**:
   - The notification is enabled when an attempt to acquire data by the client has failed.
   - Once enabled, the notification remains enabled till:
     - For writer client, empty data size falls below the watermark.
     - For reader client, valid data size falls below the watermark.

     At this point, the notification is disabled again. Only a RingIO_release call will disable the notification. RingIO_cancel and RingIO_flush will not disable the notification.
   - Notifications are sent each time when the other client releases data, as long as the data size is above the watermark:

    - Empty data size for writer – This condition can be met in the functions RingIO_release and RingIO_flush.
    - Valid data size for reader
  - Cancel call does not enable or disable notification.
  - Flush call does not enable or disable notification. Flush called by the reader client may cause empty data size to fall above the watermark and cause a notification to be sent to the writer client.

  As a design decision, it has been decided that cancel and flush call will not affect notification. This is done irrespective of previous acquire call state. It is not possible to maintain the state log for previous acquire call failures.

3. **RINGIO_NOTIFICATION_ONCE**:
   - The notification is enabled when an attempt to acquire data by the client has failed.
   - The notification is sent when the other client releases data, when the below condition is true:
     - For writer client, empty data size is above the watermark – This condition can be met in the functions RingIO_release and RingIO_flush.
     - For reader client, valid data size is above the watermark.

     As soon as the notification is sent, it is disabled.
   - The notification is re-enabled, only when the first condition is met again (acquire attempt fails).
   - Cancel call does not enable or disable notification.
   - The notification is disabled only once the notification is sent by the other client.
   - Flush call does not enable or disable notification. Flush called by the reader client may cause empty data size to fall above the watermark and cause a notification to be sent to the writer client.
4. **RINGIO_NOTIFICATION_HDWRFIFO_ALWAYS**:
   - Notifications are sent each time when the other client releases data, as long as the data size is above the watermark:
     - Empty data size for writer - This condition can be met in the function RingIO_release and RingIO_flush.
     - Valid data size for reader
   - This notification is always enabled. Unlike RINGIO_NOTIFICATION_ALWAYS, this notification does not require buffer to get full/empty or acquire to fail to get enabled.
   - Cancel call does not enable or disable notification.
   - Flush call does not enable or disable notification. Flush called by the reader client may cause empty data size to fall above the watermark and cause a notification to be sent to the writer client.

5. **RINGIO_NOTIFICATION_HDWRFIFO_ONCE**: This notification type will send a notification only once when a low watermark condition is satisfied and then it is disabled.
   - Unlike RINGIO_NOTIFICATION_ONCE, this notification does not require buffer to get full/empty or acquire to fail to get enabled.
   - The notification is sent when the other client releases data, when the below condition is true:
     - For writer client, empty data size is above the watermark - This condition can be met in the function RingIO_release and RingIO_flush.
     - For reader client, valid data size is above the watermark.

     As soon as the notification is sent, it is disabled.
   - The notification is re-enabled when the data size crosses the watermark:
     - For writer client, empty data size falls below the watermark.
     - For reader client, valid data size falls below the watermark.
   - Cancel call will affect the notification state. If the notification has been enabled earlier either because of a failed acquire call or a low watermark condition is satisfied, this notification will be disabled if the low watermark condition is no longer true.
     - The notification will be disabled when the data size crosses the watermark:
       1. For writer client, empty data size falls above the watermark.
       2. For reader client, valid data size falls above the watermark.
   - Flush call will affect the notification state.
     - For writer client, the notification will be disabled when the data size falls above the watermark i.e. empty data size is greater than the watermark.
     - For reader client, the notification will be enabled when the data size falls below the watermark i.e. valid data size is lesser than the watermark.

4.1.8 Notification related details

4.1.8.1.1 Writer Client calls Acquire

The below table discusses the effect of the acquire call on the writer client's notification depending upon the notification type.

The acquire call called by the writer does not effect the reader client's notification status.

API Writer RingIO_acquire ()	Watermark crossed i.e. size falls below watermark and current acquire has passed			Current acquire call has failed		
Client Notification Type	Enable Notify	Disable Notify	Send Notify	Enable Notify	Disable Notify	Send Notify

RINGIO_NOTIFICATION _ONCE: size = emptySize	NO	NO	NO	YES	NO	NO
RINGIO_NOTIFICATION _ALWAYS: size = emptySize	NO	YES	NO	YES	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ONCE: size = emptySize	YES	NO	NO	YES	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ALWAYS: size = emptySize	NO	NO	NO	YES	NO	NO

4.1.8.1.2 Reader Client calls Acquire

The below table discusses the effect of the acquire call on the reader client's notification state depending upon the notification type.

The acquire call called by the reader does not effect the writer client's notification status.

API Reader RingIO_acquire ()	Watermark crossed i.e. size falls below watermark and current acquire has passed			Current acquire call has failed		
Client Notification Type	Enable Notify	Disable Notify	Send Notify	Enable Notify	Disable Notify	Send Notify
RINGIO_NOTIFICATION _ONCE: size = validSize	NO	NO	NO	YES	NO	NO
RINGIO_NOTIFICATION _ALWAYS: size = validSize	NO	YES	NO	YES	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ONCE: size = validSize	YES	NO	NO	YES	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ALWAYS: size = validSize	NO	NO	NO	YES	NO	NO

4.1.8.1.3 Writer Client calls Release

The below table discusses the effect of the release call on the reader and writer client notification state depending upon the notification type.

If notification is sent for RINGIO_NOTIFICATION_ONCE/ RINGIO_NOTIFICATION_ HDWRFIFO_ALWAYS the notification is disabled for the reader client.

API Writer RingIO_release ()	**Notification state for the Reader client after writerRelease call.**					
	Watermark crossed i.e. size goes above the watermark and current release has passed.			Watermark not crossed i.e. size remains below the watermark and current release has passed.		
Client Notification Type	Enable	Disable	Send	Enable	Disable	Send Notify

	Notify	Notify for Reader Client	Notify	Notify	Notify for Reader Client	
RINGIO_NOTIFICATION _ONCE: size = validSize	NO	YES	YES	NO	NO	NO
RINGIO_NOTIFICATION _ALWAYS: size = validSize	NO	NO	YES	NO	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ONCE: size = validSize	NO	YES	YES	NO	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ALWAYS: size = validSize	NO	NO	YES	NO	NO	NO

4.1.8.1.4 Reader Client calls Release

The below table discusses the effect of the release call on the reader and writer client notification state depending upon the notification type.

If notification is sent for RINGIO_NOTIFICATION_ONCE/ RINGIO_NOTIFICATION_ HDWRFIFO_ALWAYS the notification is disabled for the writer client.

API Reader RingIO_release ()	**Notification state for the Writer client after readerRelease call.**					
	Watermark crossed i.e. size is goes above the watermark and current release has passed.			Watermark not crossed i.e. size remains below the watermark and current release has passed.		
Client Notification Type	Enable Notify	Disable Notify	Send Notify	Enable Notify	Disable Notify	Send Notify
RINGIO_NOTIFICATION _ONCE: size = emptySize	NO	YES	YES	NO	NO	NO
RINGIO_NOTIFICATION _ALWAYS: size = emptySize	NO	NO	YES	NO	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ONCE: size = emptySize	NO	YES	YES	NO	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ALWAYS: size = emptySize	NO	NO	YES	NO	NO	NO

4.1.8.1.5 Writer Client calls Cancel

The below table discusses the effect of the cancel call on the writer client's notification state depending upon the notification type.

The acquire call called by the writer does not effect the reader client's notification status.

API Writer	Watermark crossed i.e. size goes above the watermark	Watermark crossed i.e. size goes above the watermark and

RingIO_cancel ()	and notification was enabled.			notification was disabled.		
Client Notification Type	Enable Notify	Disable Notify	Send Notify	Enable Notify	Disable Notify	Send Notify
RINGIO_NOTIFICATION _ONCE: size = emptySize	NO	NO	NO	NO	NO	NO
RINGIO_NOTIFICATION _ALWAYS: size = emptySize	NO	NO	NO	NO	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ONCE: size = emptySize	NO	YES	NO	NO	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ALWAYS: size = emptySize	NO	NO	NO	NO	NO	NO

4.1.8.1.6 Reader Client calls Cancel

The below table discusses the effect of the cancel call on the reader client's notification state depending upon the notification type.

The acquire call called by the reader does not effect the writer client's notification status.

API Reader RingIO_cancel ()	Watermark crossed i.e. size goes above the watermark and notification was enabled.			Watermark crossed i.e. size goes above the watermark and Notification was disabled.		
Client Notification Type	Enable Notify	Disable Notify	Send Notify	Enable Notify	Disable Notify	Send Notify
RINGIO_NOTIFICATION _ONCE: size = validSize	NO	NO	NO	NO	NO	NO
RINGIO_NOTIFICATION _ALWAYS: size = validSize	NO	NO	NO	NO	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ONCE: size = validSize	NO	YES	NO	NO	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ALWAYS: size = validSize	NO	NO	NO	NO	NO	NO

4.1.8.1.7 Writer Client calls Flush

The below table discusses the effect of the flush call on both the reader and writer client's notification state depending upon the notification type.

API Writer RingIO_flush ()	**Notification state for the RingIO client after writerFlush call**					
	Reader client			Writer client		
Client Notification Type	Enable	Disable	Send	Enable	Disable	Send

	Notify	Notify	Notify	Notify	Notify	Notify
RINGIO_NOTIFICATION _ONCE	NO	NO	NO	NO	NO	NO
RINGIO_NOTIFICATION _ALWAYS	NO	NO	NO	NO	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ONCE	YES (if valid size falls below the watermark), NO (if valid size falls below the water mark)	NO	NO	NO	YES (if empty Size falls above the watermark) NO (if empty Size falls below the watermark)	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ALWAYS	NO	NO	NO	NO	NO	NO

4.1.8.1.8 Reader Client calls Flush

The below table discusses the effect of the flush call on both the reader and writer client's notification state depending upon the notification type.

API Reader RingIO_flush ()	**Notification state for the RingIO client after readerFlush call**					
	Reader client			Writer client		
Client Notification Type	Enable Notify	Disable Notify	Send Notify	Enable Notify	Disable Notify	Send Notify
RINGIO_NOTIFICATION _ONCE	NO	NO	YES (if empty size falls above the watermark) NO (if empty size falls above the watermark)	NO	NO	NO
RINGIO_NOTIFICATION _ALWAYS	NO	NO	YES (if empty size falls above the watermark) NO (if empty size falls above the watermark)	NO	NO	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ONCE	YES (if valid Size falls below the watermark), NO (if valid size falls above the water mark)	NO	YES (if empty size falls above the watermark) NO (if empty size falls above the watermark)	NO	YES (if empty Size falls above the watermark) NO (if empty Size falls below the watermark)	NO
RINGIO_NOTIFICATION _ HDWRFIFO_ALWAYS	NO	NO	YES (if empty size falls above the watermark)	NO	NO	NO

			NO (if empty size falls above the watermark)			

4.2 Control flow

The typical control flow of an application using the RingIO includes the following activities

1. **Create a RingIO**

   A RingIO in shared memory is identified by a RingIO name. Hence, to create a RingIO, the application has to specify a RingIO name. This name should be used to open the RingIO buffer for reading or writing. At any point in time, only a fixed number of RingIO can be created. This fixed number can be modified through configuration.

   The memory to be used for creation of the RingIO buffers is specified by the application writer through a POOL interface. The application specifies the POOL ID to be used for allocating the buffers. The POOL may be a shared memory pool for a RingIO shared between two processors, or within local memory if the RingIO is for intra-processor transfer. POOL IDs for four different buffers must be provided, the data buffer memory, attribute buffer memory, the control structure memory, and the memory for the lock to be used for access to the RingIO structures.

2. **Get a handle to the RingIO**

   To write and read data from the RingIO, a handle needs to be acquired. This can be done using the RingIO_open call. The reader and writer will get different handles using which they can read and write data to the RingIO. A set of open flags can also be provided which will inform the RingIO whether cache is being used for the shared buffers and whether write requests have to be satisfied to the exact size (If requested size is not available, partial buffer is not returned).

3. **Write into an acquired buffer**

   To write data into the RingIO, an empty data buffer has to be first acquired. After this data can be written into the acquired buffer and the filled up buffer can be released back to the RingIO. Any sized data buffer can acquired and released. Data buffer acquired using a single acquire call can also be released in parts. Similarly data buffers acquired in multiple calls can be released using a single release call. When the writer releases a buffer, any attributes set in the acquired buffer also gets released.

4. **Read from an acquired buffer**

   To read data from the RingIO, a filled up data buffer has to be first acquired. After this the data buffer can be used up and the empty buffer can be released back to the RingIO. Any sized data buffer can be acquired and released. Data buffer acquired using a single acquire call can also be released in parts. Similarly data buffers acquired in multiple calls can be released using a single release call.

5. **Set attributes**

   The writer can set attributes into the RingIO to communicate any synchronous information to the reader. Attributes can be set at any offset on the acquired buffer. One exception to this rule is when no buffer has been acquired. In this case, the attribute is set at the next write location.

6. **Get attributes**

   The reader can read the attributes set by the writer using the get attribute function. The attributes can be read only at the current read offset.

7. **Flush**

   Both the reader and writer can call the flush function in two different modes soft-flush and hard-flush. In hard-flush, all valid data and associated attributes are removed from the RingIO. In soft-flush, the functionality of flush is different depends on the caller. For the writer, flush will remove all valid data after the first readable attribute. This call removes the attribute as well. For the reader, flush will remove all valid data before the first readable attribute. The attribute is left in the stream itself. The flush call will also return the first readable attribute type and the optional parameter to the caller.

8. **Close RingIO**

   Closing the writer or the reader allows another writer or reader to open the RingIO again. Closing of the RingIO can be done only if no data buffers have been acquired.

9. **Delete the RingIO**

   This removes the RingIO instance from shared memory. Both the reader and writer will not be able to use the RingIO after it is deleted.

4.3 RingIO component

The RingIO component is implemented as a user-level library within *DSPLINK*, and a library on the DSP-side.

The RingIO component utilizes the shared memory between the GPP and DSP for implementing the data transfer protocol. The RingIO component uses the functionality provided by the IPS component for notification of events across the processor boundary. On the GPP-side, since RingIO is on the user-side, it uses the NOTIFY component to register for the IPS event.

4.3.1 GPP-side

4.3.1.1 Component interaction

The component interaction diagram indicates the interaction of RingIO with the various components within *DSPLINK*.

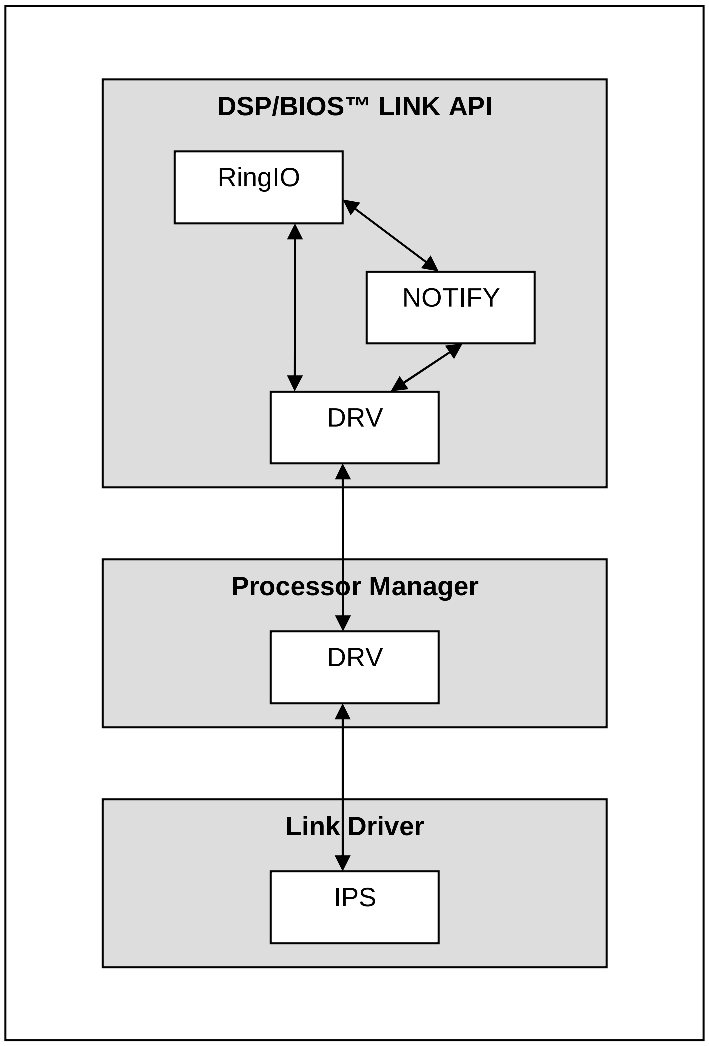

Figure 3. GPP-side Component Interaction

4.3.2 DSP-side

4.3.2.1 Overview

The DSP-side of the RingIO component shall be implemented as a library over the base inter-processor communication functionality provided by *DSPLINK*.

Scalability for RingIO shall be provided through compile-time flags, which shall be set by the common configuration tool.

4.3.2.2 Component interaction

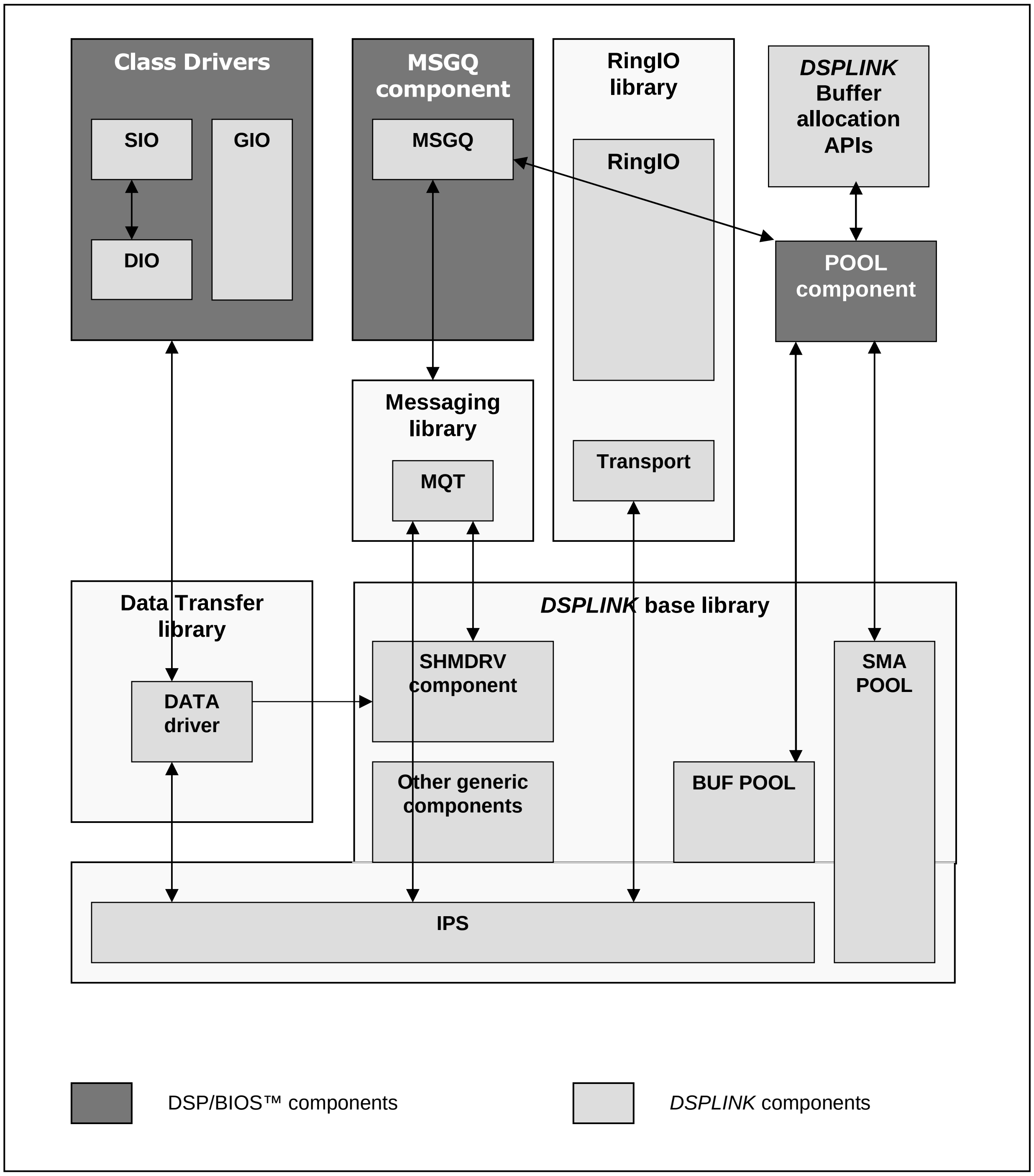

Figure 4. DSP-side Component Interaction

5 RingIO

The RingIO component on a processor is responsible for implementing the ring buffer data transfer protocol for intra-DSP and DSP<->GPP transfer.

This section provides a detailed design for the RingIO component on the GPP and DSP-sides. The RingIO component has a similar design on both the GPP and DSP sides.

5.1 Constants & Enumerations

5.1.1 RINGIO_NAME_MAX_LEN

This constant defines the maximum length of the RingIO names.

Definition

```
#define RINGIO_NAME_MAX_LEN  32
```

Comments

None

Constraints

None

See Also

None

5.1.2 RINGIO_INVALID_ATTR

This constant defines the invalid attribute type. This value should not be used for user defined attribute types.

Definition

```
#define RINGIO_INVALID_ATTR    (Uint16) 0xFFFF
```

Comments

None

Constraints

None

See Also

None

5.1.3 RINGIO_DATABUF_CACHEUSE

This constant defines the flag used for opening the RingIO. It indicates that cache coherence needs to be maintained for the data buffer.

Definition

```
#define RINGIO_DATABUF_CACHEUSE 0x1
```

Comments

None

Constraints

None

See Also

None

5.1.4 RINGIO_ATTRBUF_CACHEUSE

This constant defines the flag used for opening the RingIO. It indicates that cache coherence needs to be maintained for the attribute buffer.

Definition

```
#define RINGIO_ATTRBUF_CACHEUSE 0x2
```

Comments

None

Constraints

None

See Also

None

5.1.5 RINGIO_CONTROL_CACHEUSE

This constant defines the flag used for opening the RingIO. It indicates that cache coherence needs to be maintained for the control structure buffer.

Definition

```
#define RINGIO_CONTROL_CACHEUSE 0x4
```

Comments

None

Constraints

None

See Also

`None`

5.1.6 RINGIO_NEED_EXACT_SIZE

This constant defines the flag used for opening the RingIO. It indicates that exact size buffer should be provided while acquiring a buffer for this RingIO.

Definition

```
#define RINGIO_NEED_EXACT_SIZE 0x8
```

Comments

None

Constraints

None

See Also

None

5.2 Typedefs & Data Structures

5.2.1 RingIO_NotifyType

This structure enumerates the different RingIO notification types. Please refer to section Notification related details for more detailed information.

Definition

```
typedef enum {
    RINGIO_NOTIFICATION_NONE = 0,
    RINGIO_NOTIFICATION_ALWAYS,
    RINGIO_NOTIFICATION_ONCE,
    RINGIO_NOTIFICATION_HDWRFIFO_ALWAYS,
    RINGIO_NOTIFICATION_HDWRFIFO_ONCE
} RingIO_NotifyType ;
```

Fields

`RINGIO_NOTIFICATION_NONE`	No notification required.
`RINGIO_NOTIFICATION_ALWAYS`	Notify whenever the other side sends data/frees up space. This notification is enabled only when an attempt to acquire data fails. The notification is sent to the writer if empty buffer size is more than watermark, and sent to the reader if valid buffer size is more than watermark.
`RINGIO_NOTIFICATION_ONCE`	Notify when the other side sends data/frees up space. Once the notification is done, the notification is disabled until it is enabled again by a failed attempt to acquire data. The notification is sent to the writer if empty buffer size is more than watermark, and sent to the reader if valid buffer size is more than watermark.
`RINGIO_NOTIFICATION_HDWRFIFO_ALWAYS`	Notify whenever the other side sends data/frees up space. This notification is never disabled. This notification type will always send a notification when low watermark condition has been satisfied. Unlike RINGIO_NOTIFICATION_ALWAYS, this notification does not require buffer to get full/empty or acquire to fail to get enabled.

`RINGIO_NOTIFICATION_HDWRFIFO_ONCE`	Notify when the other side sends data/frees up space. Once the notification is done, the notification is disabled until it is enabled again. The notification is enabled once the watermark is crossed and does not require buffer to get full/empty. This notification type will send a notification only once when a low watermark condition is satisfied and then it is disabled. The notification gets enabled when the amount of buffer full/empty crosses above the watermark. Unlike RINGIO_NOTIFICATION_ONCE, this notification does not require buffer to get full/empty or acquire to fail to get enabled.

Comments

None.

Constraints

None.

See Also

`RingIO_setNotifier ()`

5.2.2 RingIO_OpenMode

This structure enumerates the different modes in which the RingIO can be opened.

Definition

```
typedef enum{
    RINGIO_MODE_READ = 0,
    RINGIO_MODE_WRITE
} RingIO_OpenMode ;
```

Fields

`RINGIO_MODE_READ`	Reader mode.
`RINGIO_MODE_WRITE`	Writer mode.

Comments

None.

Constraints

None.

See Also

`RingIO_open ()`

5.2.3 RingIO_TransportType

This structure enumerates the different types of transports for the RingIO.

Definition

```
typedef enum{
    RINGIO_TRANSPORT_DSP_DSP = 1,
    RINGIO_TRANSPORT_GPP_DSP
} RingIO_TransportType ;
```

Fields

`RINGIO_TRANSPORT_DSP_DSP`	Intra-DSP transport.
`RINGIO_TRANSPORT_GPP_DSP`	GPP<->DSP transport.

Comments

This enumeration is provided as part of RingIO_create () attributes.

Constraints

None

See Also

`RingIO_create ()`

5.2.4 RingIO_Attrs

This structure defines the attributes to be provided while creating the RingIO.

Definition

```
typedef struct RingIO_Attrs_tag {
    RingIO_TransportType   transportType ;
    Uint16                 ctrlPoolId  ;
    Uint16                 dataPoolId  ;
    Uint16                 attrPoolId  ;
    Uint16                 lockPoolId  ;
    Uint32                 dataBufSize ;
    Uint32                 footBufSize ;
    Uint32                 attrBufSize ;
} RingIO_Attrs ;
```

Fields

`transportType`	Transport type - This specifies whether the data transport is between DSP<-> DSP or DSP<->ARM.
`ctrlPoolId`	Pool to be used to allocate memory for control structure.
`dataPoolId`	Pool to be used to allocate memory for data buffer.
`attrPoolId`	Pool to be used to allocate memory for attribute buffer.
`lockPoolId`	Pool to be used to allocate memory for lock structure.
`dataBufSize`	Data Buffer Size in bytes.
`footBufSize`	Size of the footer area.
`attrBufSize`	Attribute Buffer Size in bytes.

Comments

This structure is provided to the RingIO_create () call.

Constraints

None.

See Also

`RingIO_create ()`

5.2.5 RingIO_Client

This structure defines the RingIO Shared memory client structure.

The RingIO Reader or Writer state information is stored in this structure.

Definition

```
typedef struct RingIO_Client_tag {
    Uint32                  procId ;
    RingIO_OpenMode         openMode ;
    RingIO_BufPtr           pDataStart ;
    RingIO_BufPtr           pAttrStart ;
    Uint32                  acqStart ;
    Uint32                  acqSize ;
    Uint32                  acqAttrStart ;
    Uint32                  acqAttrSize ;
    Uint32                  notifyType ;
    RingIO_NotifyFunc       notifyFunc ;
    RingIO_NotifyParam      notifyParam ;
    Uint32                  notifyWaterMark ;
    Uint32                  flags ;
    RingIO_ControlStruct *  virtControlHandle ;
    Void *                  virtLockHandle;
    Uint32                  isValid ;
    Uint32                  refCount ;
    Uint16                  notifyFlag ;
    ADD_PADDING             (padding, RINGIO_CLIENT_PADDING)
} RingIO_Client ;
```

Fields

`procId`	Processor Id where the client is executing
`openMode`	Indicates whether the client is a reader or writer
`pDataStart`	Virtual start address of the data buffer
`pAttrStart`	Virtual start address of the attr buffer
`acqStart`	Start offset of data buffer that has been acquired by the application.
`acqSize`	Size of data that has been acquired
`acqAttrStart`	Start offset of the acquired attribute buffer
`acqAttrSize`	Size of attribute data that has been acquired
`notifyType`	Notification type
`notifyFunc`	Notification function for this client
`notifyParam`	Parameter to the Notification function
`notifyWaterMark`	Watermark that should be satisfied before notification is done

`flags`	Client-specific flags indicating cache requirements and exact size requirement.
`virtControlHandle`	Handle to the Control structure. Apps do not have direct access to the control structure. The Control structure can only be accessed through the client handle
`virtLockHandle`	Virtual (GPP) address of the lock that should be used to protect the Control structure from multiple accesses.
`isValid`	Indicates whether the Client is initialized
`refCount`	Reference count of whether RingIO has been opened.
`notifyFlag`	Denotes whether notification needs to be done or not
`padding`	Padding for cache-line alignment, if required.

Comments

This structure contains client-specific information for each RingIO.

Constraints

None.

See Also

`RingIO_ControlStruct`

5.2.6 RingIO_ControlStruct

This structure defines the RingIO Control Structure. This structure is stored in shared memory and is accessible by all clients. The control structure supports a single reader and a single writer for the ring buffer.

Definition

```
struct RingIO_ControlStruct_tag {
    Uint32               entryId;
    RingIO_TransportType  transportType;
    RingIO_BufPtr        phyBufStart;
    Uint32               phyBufEnd;
    Uint32               curBufEnd;
    Uint32               dataBufEnd;
    Uint32               dataBufSize;
    Uint32               footBufSize;
    Uint32               validSize;
    Uint32               emptySize;
    RingIO_BufPtr        phyAttrStart;
    Uint32               phyAttrBufEnd;
    Uint32               curAttrBufEnd;
    Uint32               validAttrSize;
    Uint32               emptyAttrSize;
    Int32                prevAttrOffset;
    Void *               phyLockHandle;
    ADD_PADDING          (padding, RINGIO_CONTROLSTRUCT_PADDING)
    RingIO_Client        writer ;
    RingIO_Client        reader ;
} RingIO_ControlStruct, *RingIO_ControlHandle;
```

Fields

`entryId`	ID of the RingIO within the entry array.
`transportType`	Transport type - This specifies whether the data transport is between DSP<-> DSP or DSP<->ARM.
`phyBufStart`	Physical start address of the data buffer.
`phyBufEnd`	Total size of the Data buffer (offset from phyBufStart)
`curBufEnd`	Current buffer size. This may be <= dataBufEnd (offset from phyBufStart)
`dataBufEnd`	End offset of the main data buffer. This excludes the footer region.
`dataBufSize`	Buffer size of the main data buffer. This excludes the footer region.
`footBufSize`	Size of the footer region as specified by the user.
`validSize`	Amount of valid data available in the data buffer. Valid Data is the total data that is readable by the reader using an acquire call. This does not include the size of the data buffer already acquired by the reader

`emptySize`	Amount of empty space in the data buffer. This does not include the empty space already acquired by the writer
`phyAttrStart`	Physical start address of the attribute buffer
`phyAttrBufEnd`	Total Size of the attribute buffer (offset)
`curAttrBufEnd`	Current Attribute buffer size. This may be <= the phyAttrBufEnd (offset)
`validAttrSize`	Amount of valid attribute bytes available in the attribute buffer. The valid attribute bytes does not include the attribute bytes already acquired by the reader
`emptyAttrSize`	Amount of empty space in the attr buffer. This does not include the empty attr space already acquired by the writer
`prevAttrOffset`	Offset of the most recent attribute
`phyLockHandle`	Physical (DSP) address of the lock that should be used to protect the Control structure from multiple accesses.
`padding`	Padding used for cache line alignment
`writer`	Writer state information
`reader`	Reader state information

Comments

This structure contains the complete shared information for each RingIO.

Constraints

None.

See Also

`RingIO_Client`

5.2.7 RingIO_Entry

This structure defines the Entry structure for the RingIO data transport.

Definition

```
typedef struct RingIO_Entry_tag {
    Pvoid        phyControl ;
    Pvoid        virtControl ;
    Char8        name [RINGIO_NAME_MAX_LEN] ;
    Uint16       ownerProcId ;
    Uint16       ctrlPoolId ;
    Uint16       dataPoolId ;
    Uint16       attrPoolId ;
    Uint16       lockPoolId ;
    ADD_PADDING  (padding, RINGIO_RINGIOENTRY_PADDING)
} RingIO_Entry ;
```

Fields

`phyControl`	Physical (DSP) address of the Control structure for the RingIO.
`virtControl`	Virtual (GPP) address of the Control structure for the RingIO.
`name`	System wide unique identifier for the RingIO
`ownerProcId`	Creator's processor ID of this ringio.
`ctrlPoolId`	Pool to be used to allocate memory for control structure.
`dataPoolId`	Pool to be used to allocate memory for data buffer.
`attrPoolId`	Pool to be used to allocate memory for attribute buffer.
`lockPoolId`	Pool to be used to allocate memory for lock structure.
`padding`	Padding for cache line alignment, if required.

Comments

The RingIO entry structure is used to maintain system-wide information about each RingIO created.

Constraints

None.

See Also

`RingIORegion`

5.2.8 RingIO_Ctrl

This structure defines the control structure required by the RINGIO component. It contains information about all RINGIO objects shared between the GPP and a specific DSP.

Definition

```
typedef struct RingIO_Ctrl_tag {
    Uint32         isInitialized ;
    Uint32         dspId ;
    Uint32         maxEntries ;
    Uint32         ipsId ;
    Uint32         ipsEventNo ;
    RingIO_Entry * dspAddrEntry ;
    ADD_PADDING    (padding, RINGIO_CTRL_PADDING)
    MPCS_ShObj     lockObj ;
} RingIO_Ctrl ;
```

Fields

`isInitialized`	Flag to indicate if this region was initialized
`dspId`	ID of the DSP with which the RingIO Region is shared
`maxEntries`	Maximum number of RingIO instances supported by the RingIO.
`ipsId`	ID of the IPS to be used.
`ipsEventNo`	IPS Event number associated with the RingIO.
`dspAddrEntry`	Pointer to array in DSP address space of RINGIO objects that can be created.
`padding`	Padding for cache line alignment, if required.
`lockObj`	Lock used to protect the shared RingIO_Ctrl from multiple simultaneous accesses.

Comments

The RingIO region contains all global information about RingIOs present in the system.

Constraints

None.

See Also

`RingIOEntry`

5.2.9 RingIO_MemInfo

This structure defines the memory information structure for the RingIO component.

Definition

```
typedef struct RingIO_MemInfo_tag {
    ProcessorId procId ;
    Uint32      physAddr ;
    Uint32      kernAddr ;
    Uint32      userAddr ;
    Uint32      size ;
} RingIO_MemInfo ;
```

Fields

`procId`	Processor ID of the processor with which the RingIO is shared.
`physAddr`	Physical address of the memory region for RingIO
`kernAddr`	Kernel address of the memory region for RingIO
`userAddr`	User address of the memory region for RingIO
`size`	Size of the memory region for RingIO

Comments

The RingIOMemInfo structure contains information about the RingIO memory region, mapped into user space.

Constraints

None.

See Also

`RingIORegion`

5.3 API Definition

5.3.1 RingIO_getAcquiredOffset

This macro returns the current acquire offset for the client.

Syntax

```
#define RingIO_getAcquiredOffset(client)                \
                    (((RingIO_Client *) client)->acqStart)
```

Arguments

`IN`	`RingIO_Handle`	`handle`
	Handle of the RingIO instance	

Return Value

`Current acquire offset`	The operation has been successfully completed.

Comments

This macro is provided as a helper utility to return information about the current state of the RingIO client.

Constraints

None.

See Also

`None.`

5.3.2 RingIO_getAcquiredSize

This macro returns the size of buffer currently acquired by the client.

Syntax

```
#define RingIO_getAcquiredSize(client)                  \
                    (((RingIO_Client *) client)->acqSize)
```

Arguments

`IN`	`RingIO_Handle`	`handle`
	Handle of the RingIO instance	

Return Value

`Current acquired size`	The operation has been successfully completed.

Comments

This macro is provided as a helper utility to return information about the current state of the RingIO client.

Constraints

None.

See Also

`None.`

5.3.3 RingIO_getWatermark

This macro returns the current watermark level specified by the client.

Syntax

```
#define RingIO_getWatermark(client)                     \
                    (((RingIO_Client *) client)->notifyWaterMark)
```

Arguments

`IN`	`RingIO_Handle`	`handle`
	Handle of the RingIO instance	

Return Value

`Current watermark level`	The operation has been successfully completed.

Comments

This macro is provided as a helper utility to return information about the current state of the RingIO client.

Constraints

None.

See Also

`None.`

5.3.4 RingIO_create

This function creates a RingIO instance in Shared memory using the creation params specified.

Syntax

```
DSP_STATUS RingIO_create (Char8 * name, RingIO_Attrs * attrs) ;
```

Arguments

`IN`	`Char8 *`	`Name`
	Unique name identifying the RingIO instance.	
`IN`	`RingIO_Attrs *`	`attrs`
	Attributes to use for creating the RingIO instance	

Return Value

`RINGIO_SUCCESS`	The operation has been successfully completed.
`RINGIO_EFAILURE`	General failure
`DSP_EINVALIDARG`	Invalid arguments.
`DSP_EACCESSDENIED`	The RINGIO component has not been initialized.
`DSP_EMEMORY`	Operation failed due to a memory error.
`RINGIO_EALREADYEXISTS`	The specified RINGIO name is already in use.(i.e. RingIO or MPCS name already exists)
`DSP_EFAIL`	General failure.
`DSP_ENOTFOUND`	A specified entity was not found. Address translation between user space and DSP address space failed. See POOL_translateAddr() in POOL design document.

Comments

‘name’ is a null terminated ASCII string.

Constraints

‘name’ and ‘attrs’ should not be NULL

See Also

`RingIO_delete ()`

5.3.5 RingIO_delete

This function deletes a RingIO instance in shared memory between the reader and writer.

Syntax

```
DSP_STATUS RingIO_delete (Char8*  name) ;
```

Arguments

`IN`	`Char8 *`	`name`
	Name of the RingIO instance to be deleted	

Return Value

`RINGIO_SUCCESS`	The operation has been successfully completed.
`RINGIO_EFAILURE`	General failure
`DSP_EINVALIDARG`	Invalid arguments.
`DSP_EACCESSDENIED`	The RINGIO component has not been initialized.
`DSP_EFAIL`	General failure.

Comments

‘name’ is a null terminated ASCII string.

Constraints

‘name’ should not be NULL

See Also

`RingIO_create ()`

5.3.6 RingIO_open

This function opens a RingIO instance handle which can be used for either reading or writing. Only one reader and one writer can be opened on a RingIO.

Syntax

```
RingIO_Handle RingIO_open (Char8 *         name,
                           RingIO_OpenMode openMode,
                           Uint32          flags) ;
```

Arguments

`IN`	`Char8 *`	`Name`
	Name of the RingIO channel to be opened.	
`IN`	`RingIO_OpenMode`	`openMode`
	Mode with which the RingIO channel is to be opened (Reader/Writer)	
`IN`	`Uint32`	`Flags`
	Cache usage and other notification flags.	

Return Value

`<valid handle>`	Operation successfully completed.
`NULL`	General failure, name not found.

Comments

None

Constraints

‘name’ should not be NULL.

See Also

```
RINGIO_DATABUF_CACHEUSE
RINGIO_ATTRBUF_CACHEUSE
RINGIO_CONTROL_CACHEUSE
RINGIO_NEED_EXACT_SIZE
RingIO_OpenMode
RingIO_close ()
```

5.3.7 RingIO_close

This function closes an already open RingIO reader/writer.

Syntax

```
DSP_STATUS RingIO_close (RingIO_Handle handle) ;
```

Arguments

IN	`RingIO_Handle`	`handle`
	Handle of the RingIO instance to close	

Return Value

`RINGIO_SUCCESS`	The operation has been successfully completed.
`RINGIO_EFAILURE`	General failure
`DSP_EINVALIDARG`	Invalid arguments.
`DSP_EACCESSDENIED`	The RINGIO component has not been initialized.

Comments

None

Constraints

‘handle’ should not be NULL

See Also

`RingIO_open ()`

5.3.8 RingIO_acquire

This function acquires a data buffer from RingIO for reading or writing, depending on the mode in which the client (represented by the handle) has been opened.

Syntax

```
DSP_STATUS RingIO_acquire (RingIO_Handle   handle,
                           RingIO_BufPtr * dataBuf,
                           Uint32 *        size) ;
```

Arguments

`IN`	`RingIO_Handle`	`Handle`
	Handle of the RingIO Instance	
`OUT`	`RingIO_BufPtr *`	`dataBuf`
	Location to receive the pointer to the acquired data buffer	
`IN OUT`	`Uint32 *`	`size`
	Input: The number of bytes to acquire. Output: The number of bytes actually acquired	

Return Value

`RINGIO_SUCCESS`	The operation has been successfully completed.
`RINGIO_EFAILURE`	General failure
`RINGIO_SPENDINGATTRIBUTE`	No data buffer could be acquired because an attribute was present at the current read offset.
`RINGIO_EBUFWRAP`	Requested size of data buffer could not be returned because the available contiguous size till end of buffer is less than requested size. A smaller sized buffer may have been returned, if available. (Applicable only to RingIO writer client)
`RINGIO_EBUFFULL`	Requested size of data buffer could not be returned because the total available size is less than requested size. A smaller sized buffer may have been returned, if available. (Applicable only to RingIO writer client).
`RINGIO_EBUFEMPTY`	No valid size is available from the RingIO. (Applicable only to RingIO reader client)

`RINGIO_ENOTCONTIGUOUSDATA`	Requested size of data buffer could not be returned due to one of the following. 1. Attribute is present within, or at the end of the valid contiguous size available. 2.The available contiguous size till end of buffer is less than requested size (Applicable only to RingIO reader client)

Comments

None

Constraints

‘handle’, ‘dataBuf’ and ‘size’ should not be NULL.

‘size’ should be 4 Byte aligned.

See Also

`RingIO_release ()`

5.3.9 RingIO_release

This function releases a previously acquired buffer or part of it, of the specified size.

Syntax

```
DSP_STATUS RingIO_release (RingIO_Handle handle, Uint32 size) ;
```

Arguments

`IN`	`RingIO_Handle`	`Handle`
	Handle to the RingIO Client.	
`OUT`	`Uint32`	`Size`
	Size of data buffer to be released.	

Return Value

`RINGIO_SUCCESS`	The operation has been successfully completed.
`RINGIO_EFAILURE`	General failure.
`DSP_EINVALIDARG`	Invalid arguments.

Comments

None

Constraints

None

See Also

`RingIO_acquire ()`

5.3.10 RingIO_cancel

This function cancels any data buffers acquired by reader or writer. In the case of writer, all attributes that are set since the first acquire are removed. In the case of reader, all attributes that were obtained since the first acquired are re-instated in the RingIO instance.

Syntax

```
DSP_STATUS RingIO_cancel (RingIO_Handle handle) ;
```

Arguments

IN	RingIO_Handle	handle
	Handle to the RingIO Client.	

Return Value

RINGIO_SUCCESS	The operation has been successfully completed.
RINGIO_EFAILURE	General failure.
DSP_EINVALIDARG	Invalid arguments.

Comments

None

Constraints

‘handle’ must not be NULL

See Also

```
RingIO_acquire ()
RingIO_flush ()
```

5.3.11 RingIO_getAttribute

This function gets a fixed-size attribute from the attribute buffer. If an attribute is present, the attribute type and a related parameter are returned.

Syntax

```
DSP_STATUS RingIO_getAttribute (RingIO_Handle handle,
                                Uint16 *      type,
                                Uint32 *      param) ;
```

Arguments

IN	`RingIO_Handle`	`handle`
	Handle to the RingIO Client.	
OUT	`Uint16 *`	`type`
	Location to receive the user-defined type of attribute.	
OUT	`Uint32 *`	`param`
	Location to receive an optional parameter which depends on the attribute type.	

Return Value

`RINGIO_SUCCESS`	The operation has been successfully completed.
`RINGIO_EFAILURE`	No valid attributes are present or general failure.
`RINGIO_SPENDINGATTRIBUTE`	The operation has been successfully completed It also indicates that additional attributes are present at the current read offset.
`RINGIO_EVARIABLEATTRIBUTE`	A variable attribute exists. The application must call RingIO_getvAttribute () to get the variable attribute.
`RINGIO_EPENDINGDATA`	More data must be read before reading the attribute.
`DSP_EINVALIDARG`	Invalid arguments.

Comments

None

Constraints

'handle' must not be NULL

'type' and 'param' must point to valid memory locations

See Also

```
RingIO_getvAttribute ()
RingIO_setAttribute ()
RingIO_setvAttribute ()
```

5.3.12 RingIO_setAttribute

This function sets a fixed-size attribute at the offset provided in the acquired data buffer. If the offset is not in the range of the acquired data buffer, the attribute is not set, and an error is returned. One exception to this rule is when no data buffer has been acquired. In this case an attribute is set at the next data buffer offset that can be acquired

Syntax

```
DSP_STATUS RingIO_setAttribute (RingIO_Handle handle,
                                Uint32        offset,
                                Uint16        type,
                                Uint32        param) ;
```

Arguments

IN	RingIO_Handle	param1
	Handle to the RingIO Client.	
IN	Uint32	Offset
	Offset in the acquired data buffer to which the attribute corresponds	
IN	Uint16	Type
	User-defined type of attribute.	
IN OPT	Uint32	Param
	Optional parameter which depends on the attribute type.	

Return Value

RINGIO_SUCCESS	The operation has been successfully completed.
RINGIO_EFAILURE	General failure
DSP_EINVALIDARG	Invalid arguments
RINGIO_EWRONGSTATE	RingIO is in wrong state to set an attribute. The following scenarios will generate this error: - The data buffer is completely full. In this case, attribute can only be set at offset 0. But offset 0 falls into reader region. - The data buffer is completely acquired by the writer. Part or none of this buffer may be released. Writer is attempting to set an attribute at the end of its acquired range. In this case, end of writer buffer is the same as beginning of reader buffer. -Attribute buffer is full.

Comments

Writer client can try to set attributes after reader has released some data buffer. Incase of attribute buffer is full, reader has to read some attributes to allow writer to set attributes.

Constraints

'handle' must not be NULL

See Also

```
RingIO_getAttribute ()
RingIO_getvAttribute ()
RingIO_setvAttribute ()
```

5.3.13 RingIO_getvAttribute

This function gets an attribute with a variable-sized payload from the attribute buffer. If an attribute is present, the attribute type, the optional parameter, a pointer to the optional payload and the payload size are returned.

Syntax

```
DSP_STATUS RingIO_getvAttribute (RingIO_Handle handle,
                                 Uint16 *      type,
                                 Uint32 *      param,
                                 RingIO_BufPtr vptr,
                                 Uint32 *      pSize) ;
```

Arguments

`IN`	`RingIO_Handle`	`Handle`
	Handle to the RingIO Client.	
`OUT`	`Uint16 *`	`Type`
	Location to receive the user-defined type of attribute.	
`OUT`	`Uint32 *`	`Param`
	Location to receive an optional parameter which depends on the attribute type.	
`OUT`	`RingIO_BufPtr`	`Vptr`
	Pointer to buffer to receive the optional payload.	
`OUT`	`Uint32 *`	`pSize`
	Location with the size of the variable attribute. On return, this stores the actual size of the payload.	

Return Value

`RINGIO_SUCCESS`	The operation has been successfully completed.
`RINGIO_SPENDINGATTRIBUTE`	Additional attributes are present at the current read offset.
`RINGIO_EVARIABLEATTRIBUTE`	No buffer has been provided to receive the variable attribute payload.
`RINGIO_EPENDINGDATA`	More data must be read before reading the attribute.
`RINGIO_EFAILURE`	No valid attributes are present or general failure.
`DSP_EINVALIDARG`	Invalid arguments

Comments

None

Constraints

‘handle’ must not be NULL

‘type’, ‘param’, ‘vptr’ and ‘size’ must point to valid memory locations.

See Also

```
RingIO_getAttribute ()
RingIO_setAttribute ()
RingIO_setvAttribute ()
```

5.3.14 RingIO_setvAttribute

This function sets an attribute with a variable sized payload at the offset provided in the acquired data buffer. If the offset is not in the range of the acquired data buffer, the attribute is not set, and an error is returned. One exception to this rule is when no data buffer has been acquired. In this case an attribute is set at the next data buffer offset that can be acquired

Syntax

```
DSP_STATUS RingIO_setvAttribute (RingIO_Handle handle,
                                 Uint32        offset,
                                 Uint16        type,
                                 Uint32        param,
                                 RingIO_BufPtr pdata,
                                 Uint32        size) ;
```

Arguments

IN	RingIO_Handle	handle
	Handle to the RingIO Client.	
IN	Uint32	offset
	Offset in the acquired data buffer to which the attribute corresponds	
IN	Uint16	type
	User-defined type of attribute.	
IN OPT	Uint32	param
	Optional parameter which depends on the attribute type.	
IN	RingIO_BufPtr	pdata
	Pointer to attribute payload buffer.	
IN	Uint32	size
	Size of the attribute payload.	

Return Value

RINGIO_SUCCESS	The operation has been successfully completed.
RINGIO_EFAILURE	General failure
DSP_EINVALIDARG	Invalid arguments

`RINGIO_EWRONGSTATE`	RingIO is in wrong state to set an attribute. The following scenarios will generate this error: - The data buffer is completely full. In this case, attribute can only be set at offset 0. But offset 0 falls into reader region. - The data buffer is completely acquired by the writer. Part or none of this buffer may be released. Writer is attempting to set an attribute at the end of its acquired range. In this case, end of writer buffer is the same as beginning of reader buffer. -Attribute buffer is full.

Comments

Writer client can try to set attributes after reader has released some data buffer. Incase of attribute buffer is full, reader has to read some attributes to allow writer to set attributes.

Constraints

'handle', 'pdata' should not be NULL

'size' should be 4 byte aligned.

See Also

```
RingIO_getAttribute ()
RingIO_setAttribute ()
RingIO_getvAttribute ()
```

5.3.15 RingIO_flush

This function is used to flush the data from the RingIO. Behavior of this function depends on the value of hardFlush argument.

When hardFlush is false:

> If function is called for the writer, all the valid data in buffer after the first attribute location will be discarded. In case there are no attributes, no data will be cleared from the buffer. Note that this does not include the data that has been already acquired by the reader. Note that the attribute will also be cleared from the attribute buffer.
>
> For the reader, all the data till the next attribute location will be discarded. And if there is no attribute in the buffer, all valid data will get discarded. Note that the attribute will remain the attribute buffer. This is different from the behavior mentioned for the writer.

When hardFlush is true:

> If function is called from the writer, all committed data andattributes that is not acquired by reader are removed from the RingIO instance. The writer pointer is moved to point to reader's head pointer
>
> If function is called from the reader, all data and attributes that can be subsequently acquired from the reader are removed.

Syntax

```
DSP_STATUS RingIO_flush (RingIO_Handle handle,
                         Bool          hardFlush,
                         Uint16 *      type,
                         Uint32 *      param) ;
```

Arguments

IN	`RingIO_Handle`	`handle`
	Handle to the RingIO Client.	
IN	`Bool`	`hardFlush`
	Mode in which the flush operation discards committed data and attributes.	
OUT	`Uint16 *`	`type`
	Location to receive the User-defined type of attribute.	
OUT	`Uint32 *`	`param`
	Location to receive an optional parameter which depends on the attribute type.	

Return Value

`RINGIO_SUCCESS`	The operation has been successfully completed.
`RINGIO_EFAILURE`	General failure

`DSP_EINAVLIDARG`	Invalid arguments

Comments

None

Constraints

'handle' should not be NULL

'type' and 'param' should point to valid memory locations.

See Also

`None.`

5.3.16 RingIO_setNotifier

This function sets Notification parameters for the RingIO Client. Both the reader and writer can set their notification mechanism using this function

Syntax

```
DSP_STATUS RingIO_setNotifier (RingIO_Handle      handle,
                               RingIO_NotifyType  notifyType,
                               Uint32             notifyWatermark,
                               RingIO_NotifyFunc  notifyFunc,
                               RingIO_NotifyParam notifyParam) ;
```

Arguments

`IN`	`RingIO_Handle`	`Handle`
	Handle to the RingIO client.	
`IN`	`RingIO_NotifyType`	`notifyType`
	Type of notification.	
`IN`	`Uint32`	`notifyWatermark`
	Watermark for notification	
`IN`	`RingIO_NotifyFunc`	`notifyFunc`
	Function to call when notification is required	
`IN`	`RingIO_NotifyParam`	`notifyParam`
	Pointer to the notification parameter. The type of the pointer and its size depends on the notification function	

Return Value

`RINGIO_SUCCESS`	The operation has been successfully completed.
`RINGIO_EFAILURE`	General failure.
`DSP_EINVALIDARG`	Invalid arguments

Comments

None

Constraints

‘handle’ should not be NULL

See Also

```
RingIO_NotifyType
RingIO_NotifyFunc
RingIO_NotifyParam
```

5.3.17 RingIO_sendNotify

This function sends a notification to the other client with an associated message value.

The message value includes the RingIO Id only in the lower 16 bits of the payload and the upper 16 bits includes the message.

Syntax

```
DSP_STATUS RingIO_sendNotify (RingIO_Handle    handle,
                              RingIO_NotifyMsg msg) ;
```

Arguments

`IN`	`RingIO_Handle`	`handle`
	Handle to the RingIO client.	
`IN`	`RingIO_NotifyMsg`	`msg`
	Message to be sent along with notification.	

Return Value

`RINGIO_SUCCESS`	The operation has been successfully completed.
`RINGIO_EFAILURE`	General failure
`DSP_EINVALIDARG`	Invalid arguments
`DSP_ENOTREADY`	Failed .DSP is not ready to respond to requested command.

Comments

This API will return success only when other RingIO client is valid. Along with this, other client has to have a registered notification function.

Constraints

'handle' should not be NULL

See Also

```
RingIO_Handle
RingIO_NotifyMsg
RingIO_setNotifier ()
```

5.3.18 RingIO_getValidSize

This function returns the valid size in the RingIO.

Syntax

```
Uint32 RingIO_getValidSize (IN RingIO_Handle handle );
```

Arguments

`IN`	`RingIO_Handle`	`handle`
	Handle of the RingIO instance	

Return Value

`Valid data size`	The operation has been successfully completed.

Comments

This function is provided as a helper function to return information about the current state of the RingIO client.

Constraints

‘handle’ should not be NULL and must be valid handle.

See Also

```
RingIO_Handle
RingIO_ControlStruct
```

5.3.19 RingIO_getValidAttrSize

This function returns the valid attributes size in the RingIO.

Syntax

```
Uint32 RingIO_getValidAttrSize (IN RingIO_Handle handle );
```

Arguments

`IN`	`RingIO_Handle`	`handle`
	Handle of the RingIO instance	

Return Value

`Valid attribute size`	The operation has been successfully completed.

Comments

This function is provided as a helper function to return information about the current state of the RingIO client.

Constraints

'handle' should not be NULL and must be valid handle.

See Also

```
RingIO_Handle
RingIO_ControlStruct
```

5.3.20 RingIO_getEmptySize

This function returns the current empty data buffer size.

Syntax

```
Uint32 RingIO_getEmptySize (IN RingIO_Handle handle );
```

Arguments

`IN`	`RingIO_Handle`	`handle`
	Handle of the RingIO instance	

Return Value

`Empty data buffer size`	The operation has been successfully completed.

Comments

This function is provided as a helper function to return information about the current state of the RingIO client.

Constraints

'handle' should not be NULL and must be valid handle.

See Also

```
RingIO_Handle
RingIO_ControlStruct
```

5.3.21 RingIO_getEmptyAttrSize

This function returns the current empty attribute buffer size.

Syntax

```
Uint32 RingIO_getEmptyAttrSize (IN RingIO_Handle handle );
```

Arguments

`IN`	`RingIO_Handle`	`handle`
	Handle of the RingIO instance	

Return Value

`Empty attribute buffer size`	The operation has been successfully completed.

Comments

This function is provided as a helper function to return information about the current state of the RingIO client.

Constraints

‘handle’ should not be NULL and must be valid handle.

See Also

RingIO_Handle
RingIO_ControlStruct.