HiSysLink API

Development Reference

Issue 00B08

Date 2019-04-30

HiSilicon (Shanghai) Technologies Co., Ltd.

Address:	New R&D Center, 49 Wuhe Road, Bantian, Longgang District, Shenzhen 518129 P. R. China
Website:	http://www.hisilicon.com/en/
Email:	support@hisilicon.com

About This Document

Purpose

This document provides guides to developing dual-core applications. It describes main functions and development reference of the IPCMSG and DATAFIFO modules, which can be used to solve the dual-core communication and data transmission problems.

Related Version

The following table lists the product version related to this document.

Product Name	Version
Hi3559A	V100ES
Hi3559A	V100
Hi3559C	V100
Hi3519A	V100
Hi3556A	V100
Hi3516C	V500
Hi3516D	V300
Hi3516A	V300
Hi3559	V200
Hi3556	V200

Intended Audience

This document is intended for:

- Technical support engineers
- Software development engineers

Change History

Changes between document issues are cumulative. The latest document issue contains all changes made in previous issues.

Issue 00B08 (2019-04-30)

This issue is the eighth draft release, which incorporates the following changes:

In section 4.1, the descriptions in the **Note** fileds of HI_IPCMSG_DestroyMessage and HI_IPCMSG_SendSync are modified.

Issue 00B07 (2018-11-13)

This issue is the seventh draft release, which incorporates the following changes:

In section 4.2, HI_DATAFIFO_Close and HI_DATAFIFO_CMD are modified.

Issue 00B06 (2018-06-15)

This issue is the sixth draft release, which incorporates the following changes:

In section 4.1, HI_IPCMSG_SendOnly is added.

In section 4.2, the description in the **Note** field of HI_DATAFIFO_Write is modified.

Issue 00B05 (2018-05-15)

This issue is the fifth draft release, which incorporates the following changes:

In section 5.2, the description in the **Note** field of HI_DATAFIFO_PARAMS_S is modified.

In section 5.3, Table 5-1 is modified.

Issue 00B04 (2018-01-10)

This issue is the fourth draft release, which incorporates the following changes:

Chapter 4 API Reference

In section 4.1, HI_IPCMSG_DestroyMessage, HI_IPCMSG_SendAsync, and HI_IPCMSG_SendSync are modified.

In section 4.2, HI_DATAFIFO_CMD is modified.

Chapter 5 Data Structures

In section 5.1, HI_IPCMSG_CONNECT_S is modified, and CPU_ID_E is deleted.

Section 5.2 is modified.

Issue 00B03 (2017-09-20)

This issue is the third draft release, which incorporates the following changes:

In section 5.3, table 5-1 is modified

Issue 00B02 (2017-05-27)

This issue is the second draft release, which incorporates the following changes:

In section 5.1, the descriptions in the **Syntax** and **Member** fields of HI_IPCMSG_CONNECT_S are modified, and CPU_ID_E is added.

Issue 00B01 (2017-04-28)

This issue is the first draft release.

Contents

Figures

Tables

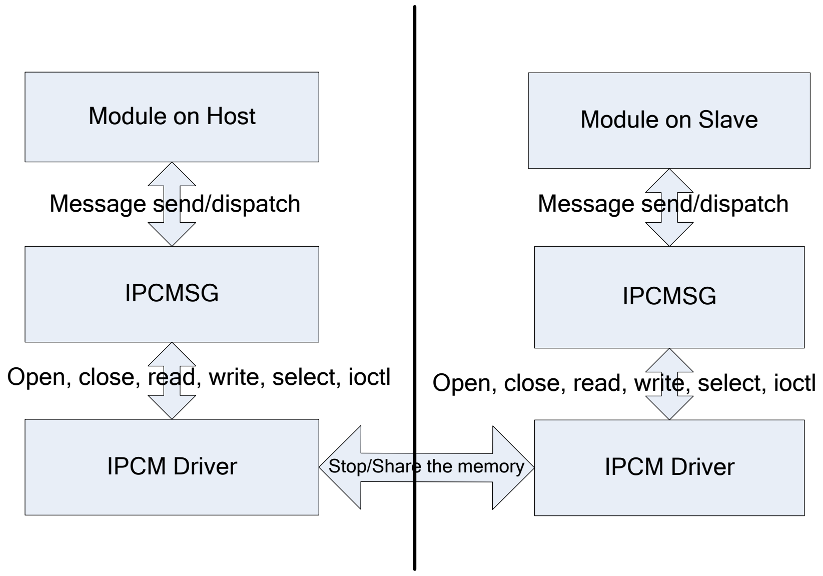
1 Overview

The HiSysLink includes the IPCMSG and DATAFIFO modules. The IPCMSG module is used for inter-core communication while the DATAFIFO module is used for inter-core data transmission.

1.1 IPCMSG

The IPCMSG module is used to solve communication problems between modules deployed in two systems in the dual-core dual-system environment. The communication data cannot exceed 1024 bytes each time. In terms of performance, the end-to-end delay of the message with high priority is restricted at the micro-second level, and that of the message with ordinary priority is restricted at the millisecond level.

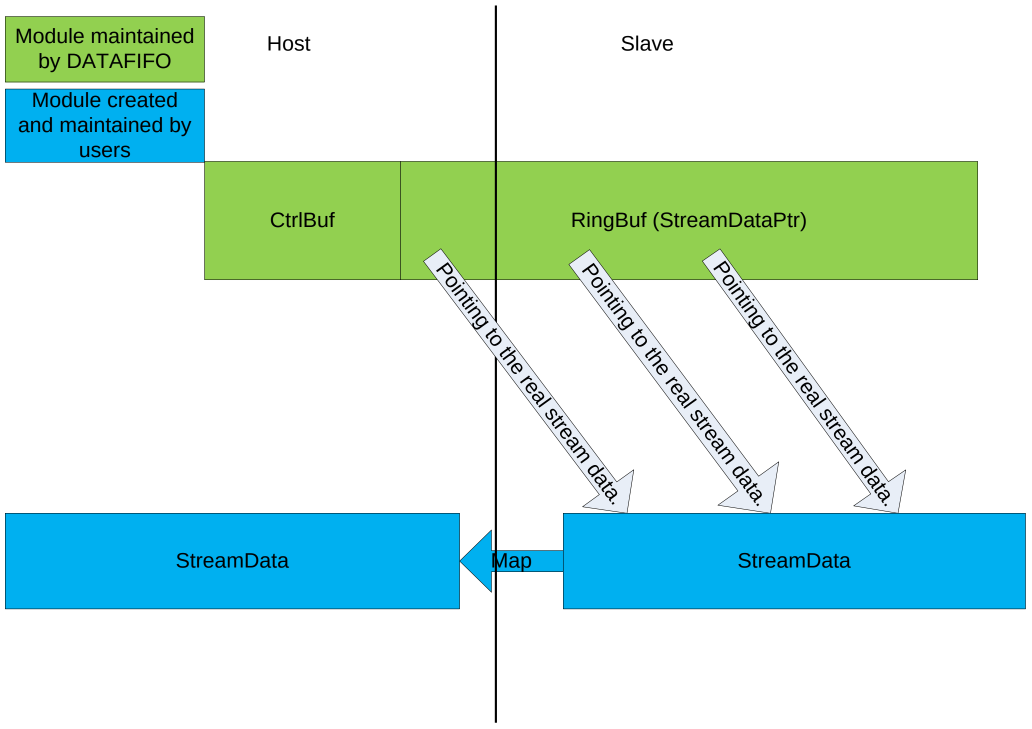
Figure 1-1 Dual-core communication



1.2 DATAFIFO

For frequent massive data transmission (such as encoding and decoding), the IPCMSG module cannot transmit it efficiently and effectively, but the DATAFIFO module provides the mechanism to implement the data transmission. Figure 1-2 shows the basic process of massive data transmission by the DATAFIFO module. The pointer to stream data is maintained in the DATAFIFO module. After the write end has applied for the stream data, it sends the pointer to the ring buffer of the DATAFIFO module. When the read end needs to read the data, the module transfers the stored data pointer to the read end. During dual-core data transmission, one end can only write data and the other end can only read data.

Figure 1-2 Dual-core data transmission


Module maintained by DATAFIFO
Module created and maintained by users
Host
Slave
CtrlBuf
RingBuf (StreamDataPtr)
Pointing to the real stream data.
Pointing to the real stream data.
Pointing to the real stream data.
StreamData
Map
StreamData

2 Important Concepts

- IPCM

  Driver layer for inter-core communication. It sends and receives messages by using the interrupts and sharing the memory, and provides standard device read and write interfaces for the upper layer.
- IPCMSG

  IPCM packaging. It provides message interfaces for users and realizes communication between two cores by sending and receiving messages.
- Physical address

  Absolute address on the DDR which is visible to both ends of the dual-core dual-system.
- Virtual address

  The virtual address varies from system to system. In the Linux system, it is visible only in the same process. In Huawei LiteOS, it is the same as the physical address.
- MMZ

  In dual-system environment, the media memory zone (MMZ) memory is allocated outside the memory occupied by the two systems. Therefore, data write and read operations on the MMZ do not affect the running of the two systems.
- Mapping

  The mapping in this document indicates the mapping from the physical address to the virtual address.
- Ring buffer and stream data

  To facilitate address offset, it is specified that the length of the data stored in the ring buffer is fixed. Therefore, the ring buffer stores only the pointer to the stream data, but not the stream data.

3 Function Description

3.1 IPCMSG

The IPCMSG module can create and destroy messages, add and delete services, establish and disconnect the connections, and send messages. For connection establishing, the module supports block and non-block modes. For message sending, it supports the sending of synchronous and asynchronous messages. For synchronization messages, timeout mechanism is supported. For sending both synchronous and asynchronous messages, if the reply message is greater than 60s, the reply message will be discarded.

3.2 DATAFIFO

The DATAFIFO module is used to transmit data across cores. The data which is input first is fetched first. The two systems implement data transmission by sharing memory. One end is responsible for data writing, and the other for data reading. The DATAFIFO module maintains four pointers (write head pointer, write tail pointer, read head pointer, and read tail pointer) internally and performs operations in the cyclic buffer at one end to implement data transmission.

The DATAFIFO module contains commands used to enable and disable channels and write and read the data, and other control commands.

4 API Reference

4.1 IPCMSG

The following APIs are provided:

- HI_IPCMSG_CreateMessage: Creates a message.
- HI_IPCMSG_CreateRespMessage: Creates a reply message.
- HI_IPCMSG_DestroyMessage: Destroys a message.
- HI_IPCMSG_AddService: Adds a service.
- HI_IPCMSG_DelService: Deletes a service.
- HI_IPCMSG_TryConnect: Establishes a non-block connection.
- HI_IPCMSG_Connect: Establishes a block connection.
- HI_IPCMSG_Disconnect: Disconnects the connection.
- HI_IPCMSG_IsConnected: Obtains the current connection state.
- HI_IPCMSG_SendAsync: Sends asynchronous messages.
- HI_IPCMSG_SendSync: Sends synchronous messages.
- HI_IPCMSG_Run: Defines the message handling function.
- HI_IPCMSG_SendOnly: Sends only messages.

HI_IPCMSG_CreateMessage

[Description]

Creates a message.

[Syntax]

```
HI_IPCMSG_MESSAGE_S* HI_IPCMSG_CreateMessage(HI_U32 u32Module, HI_U32
u32CMD, HI_VOID *pBody, HI_U32 u32BodyLen);
```

[Parameter]

Parameter	Description	Input/Output
u32Module	Module ID, which is created by users to distinguish different messages of different modules	Input

Parameter	Description	Input/Output
u32CMD	Command ID, which is created by users to distinguish different commands of a module	Input
pBody	Pointer to the message body	Input
u32BodyLen	Size of the message body	Input

[Return Value]

Return Value	Description
HI_IPCMSG_MESSAGE_S*	Pointer to the message structure
HI_NULL	Message creation failure

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

None

[Example]

None

[See Also]

HI_IPCMSG_DestroyMessage

HI_IPCMSG_CreateRespMessage

[Description]

Creates a reply message.

[Syntax]

```
HI_IPCMSG_MESSAGE_S* HI_IPCMSG_CreateRespMessage(HI_IPCMSG_MESSAGE_S
*pstRequest, HI_S32 s32RetVal, HI_VOID *pBody, HI_U32 u32BodyLen);
```

[Parameter]

Parameter	Description	Input/Output
pstRequest	Pointer to the request message	Input
s32RetVal	Return value	Input

Parameter	Description	Input/Output
pBody	Pointer to the reply message body	Input
u32BodyLen	Size of the reply message body	Input

[Return Value]

Return Value	Description
HI_IPCMSG_MESSAGE_S*	Pointer to the message structure
HI_NULL	Message creation failure

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

None

[Example]

None

[See Also]

HI_IPCMSG_DestroyMessage

HI_IPCMSG_DestroyMessage

[Description]

Destroys a message.

[Syntax]

```
HI_VOID HI_IPCMSG_DestroyMessage(HI_IPCMSG_MESSAGE_S * pstMsg);
```

[Parameter]

Parameter	Description	Input/Output
pstMsg	Message pointer	Input

[Return Value]

Return Value	Description
HI_VOID	None

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

A message cannot be destroyed repeatedly. Otherwise, a system exception may occur.

[Example]

None

[See Also]

- HI_IPCMSG_CreateMessage
- HI_IPCMSG_CreateRespMessage

HI_IPCMSG_AddService

[Description]

Adds a service.

[Syntax]

```
HI_S32 HI_IPCMSG_AddService(const HI_CHAR* pszServiceName, const
HI_IPCMSG_CONNECT_S* pstConnectAttr);
```

[Parameter]

Parameter	Description	Input/Output
pszServiceName	Pointer to the service name Maximum length of the service name: HI_IPCMSG_MAX_SERVICENAME_LEN	Input
pstConnectAttr	Attribute structure connecting to the peer server	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

Multiple services can be added. However, different services cannot use the same port number. As the client communicates with the service using the same port number, one service can only correspond to one client.

[Example]

None

[See Also]

HI_IPCMSG_DelService

HI_IPCMSG_DelService

[Description]

Deletes a service.

[Syntax]

```
HI_S32 HI_IPCMSG_DelService(const HI_CHAR *pszServiceName);
```

[Parameter]

Parameter	Description	Input/Output
pszServiceName	Pointer to the service name Maximum length of the service name: HI_IPCMSG_MAX_SERVICENAME_LEN	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

None

[Example]

None

[See Also]

HI_IPCMSG_AddService

HI_IPCMSG_TryConnect

[Description]

Establishes a non-block connection.

[Syntax]

```
HI_S32 HI_IPCMSG_TryConnect(HI_S32 *ps32Id, const HI_CHAR *pszServiceName,
HI_IPCMSG_HANDLE_FN_PTR pfnMessageHandle);
```

[Parameter]

Parameter	Description	Input/Output
ps32Id	Pointer to the message communication ID	Output
pszServiceName	Pointer to the service name	Input
pfnMessageHandle	Message handling callback function	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

None

[Example]

None

[See Also]

HI_IPCMSG_Disconnect

HI_IPCMSG_Connect

[Description]

Establishes a block connection.

[Syntax]

```
HI_S32 HI_IPCMSG_Connect(HI_S32 *ps32Id, const HI_CHAR *pszServiceName,
HI_IPCMSG_HANDLE_FN_PTR pfnMessageHandle);
```

[Parameter]

Parameter	Description	Input/Output
ps32Id	Pointer to the message communication ID	Output
pszServiceName	Pointer to the service name	Input
pfnMessageHandle	Message handling function	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

None

[Example]

None

[See Also]

HI_IPCMSG_Disconnect

HI_IPCMSG_Disconnect

[Description]

Disconnects the connection.

[Syntax]

```
HI_S32 HI_IPCMSG_Disconnect(HI_S32 s32Id);
```

[Parameter]

Parameter	Description	Input/Output
s32Id	Message communication ID	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

None

[Example]

None

[See Also]

- HI_IPCMSG_TryConnect
- HI_IPCMSG_Connect

HI_IPCMSG_IsConnected

[Description]

Obtains the current connection state.

[Syntax]

```
HI_BOOL HI_IPCMSG_IsConnected(HI_S32 s32Id);
```

[Parameter]

Parameter	Description	Input/Output
s32Id	Message communication ID	Input

[Return Value]

Return Value	Description
HI_TRUE	Connected
HI_FALSE	Disconnected

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

None

[Example]

None

[See Also]

- HI_IPCMSG_Connect
- HI_IPCMSG_TryConnect

HI_IPCMSG_SendAsync

[Description]

Sends asynchronous messages. This interface is a non-block interface. After a message is sent to the peer end, a value is returned directly without waiting for the processing of the message command.

If this API is called for sending a reply message, the peer end does not need to reply. Otherwise, the peer end must reply.

[Syntax]

```
HI_S32 HI_IPCMSG_SendAsync(HI_S32 s32Id, HI_IPCMSG_MESSAGE_S *pstMsg,
HI_IPCMSG_RESPHANDLE_FN_PTR pfnRespHandle);
```

[Parameter]

Parameter	Description	Input/Output
s32Id	Message service ID	Input
pstMsg	Message pointer	Input
pfnRespHandle	Message reply handling function This parameter can be **NULL** only when a reply message is sent. Otherwise, the parameter cannot be **NULL**.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

None

[Example]

None

[See Also]

HI_IPCMSG_SendSync

HI_IPCMSG_SendSync

[Description]

Sends synchronous messages. This interface is block interface. A value is returned after the peer end has processed the message command.

[Syntax]

```
HI_S32 HI_IPCMSG_SendSync(HI_S32 s32Id, HI_IPCMSG_MESSAGE_S *pstMsg,
HI_IPCMSG_MESSAGE_S **ppstMsg, HI_U32 s32TimeoutMs);
```

[Parameter]

Parameter	Description	Input/Output
s32Id	Message service ID	Input
pstMsg	Message pointer	Input
ppstMsg	Pointer to the reply message pointer	Output
s32TimeoutMs	Timeout period (in ms)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

When this API times out, HI_IPCMSG_DestroyMessage is called internally to destroy ***ppstMsg** (reply message). A message cannot be destroyed repeatedly. Therefore, you do not need to destroy the reply message after this API exits due to timeout.

[Example]

None

[See Also]

HI_IPCMSG_SendAsync

HI_IPCMSG_Run

[Description]

Defines the message handling function.

[Syntax]

```
HI_VOID HI_IPCMSG_Run(HI_S32 s32Id);
```

[Parameter]

Parameter	Description	Input/Output
s32Id	Message service ID	Input

[Return Value]

Return Value	Description
HI_VOID	None

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

None

[Example]

None

[See Also]

None

HI_IPCMSG_SendOnly

[Description]

Sends only messages to the peer end and does not receive return values from the peer end.

[Syntax]

```
HI_S32 HI_IPCMSG_SendOnly(HI_S32 s32Id, HI_IPCMSG_MESSAGE_S *pstRequest);
```

[Parameter]

Parameter	Description	Input/Output
s32Id	Message service ID	Input
pstRequest	Pointer to the message structure	Input

[Return Value]

Return Value	Description
0	Success

Return Value	Description
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header files: hi_comm_ipcmsg.h, hi_ipcmsg.h
- Library files: libipcmsg_big-little.a, libipcmsg_single.a

[Note]

None

[Example]

None

[See Also]

None

4.2 DATAFIFO

The following APIs are provided:

- HI_DATAFIFO_Open: Enables a data channel.
- HI_DATAFIFO_OpenByAddr: Enables a data channel using the physical address.
- HI_DATAFIFO_Close: Disables a data channel.
- HI_DATAFIFO_Read: Reads the data.
- HI_DATAFIFO_Write: Writes the data.
- HI_DATAFIFO_CMD: Defines other control commands.

HI_DATAFIFO_Open

[Description]

Enables a data channel.

[Syntax]

```
HI_S32 HI_DATAFIFO_Open(HI_DATAFIFO_HANDLE *Handle, HI_DATAFIFO_PARAMS_S
*pstParams);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Handle to the data channel	Output

Parameter	Description	Input/Output
pstParams	Pointer to the data channel parameters	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header file: **hi_datafifo.h**
- Library files: **libdatafifo_big-little.a** and **libdatafifo_single.a**

[Note]

None

[Example]

None

[See Also]

HI_DATAFIFO_Close

HI_DATAFIFO_OpenByAddr

[Description]

Enables a data channel using the physical address.

[Syntax]

```
HI_S32 HI_DATAFIFO_OpenByAddr(HI_DATAFIFO_HANDLE *Handle,
HI_DATAFIFO_PARAMS_S *pstParams, HI_U32 u32PhyAddr);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Handle to the data channel	Output
pstParams	Pointer to the data channel parameters	Input
u32PhyAddr	Physical address of the data buffer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header file: **hi_datafifo.h**
- Library files: **libdatafifo_big-little.a** and **libdatafifo_single.a**

[Note]

None

[Example]

None

[See Also]

HI_DATAFIFO_Close

HI_DATAFIFO_Close

[Description]

Disables a data channel.

[Syntax]

```
HI_S32 HI_DATAFIFO_Close(HI_DATAFIFO_HANDLE Handle);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Handle to the data channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header file: **hi_datafifo.h**
- Library files: **libdatafifo_big-little.a** and **libdatafifo_single.a**

[Note]

During data FIFO disabling, to enable normal release of the read and write data, you need to ensure that all data in the data FIFO is read. After data writing is finished, the write end needs to call HI_DATAFIFO_Write(Handle, NULL) again to trigger the release of the write data and the update of the read pointer.

[Example]

None

[See Also]

- HI_DATAFIFO_Open
- HI_DATAFIFO_OpenByAddr

HI_DATAFIFO_Read

[Description]

Reads the data.

[Syntax]

```
HI_S32 HI_DATAFIFO_Read(HI_DATAFIFO_HANDLE Handle, HI_VOID **ppData);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Handle to the data channel	Input
ppData	Pointer to the read data pointer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header file: **hi_datafifo.h**
- Library files: **libdatafifo_big-little.a** and **libdatafifo_single.a**

[Note]

None

[Example]

None

[See Also]

- HI_DATAFIFO_Write
- HI_DATAFIFO_CMD

HI_DATAFIFO_Write

[Description]

Writes the data.

[Syntax]

```
HI_S32 HI_DATAFIFO_Write(HI_DATAFIFO_HANDLE Handle, HI_VOID *pData);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Handle to the data channel	Input
pData	Written data	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header file: **hi_datafifo.h**
- Library files: **libdatafifo_big-little.a** and **libdatafifo_single.a**

[Note]

When **pData** is set to **NULL**, the data release callback function of the write end is triggered and the read tail pointer of the write end is updated.

[Example]

None

[See Also]

- HI_DATAFIFO_Read
- HI_DATAFIFO_CMD

HI_DATAFIFO_CMD

[Description]

Defines other control commands.

[Syntax]

```
HI_S32 HI_DATAFIFO_CMD(HI_DATAFIFO_HANDLE Handle, HI_DATAFIFO_CMD_E enCMD,
HI_VOID * pArg);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Handle to the data channel	Input
enCMD	Operation command	Input
pArg	For details about this parameter, see the **Note** field.	Input/Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The value is an error code. For details, see section 5.3 "Error Codes."

[Chip Difference]

None

[Requirement]

- Header file: **hi_datafifo.h**
- Library files: **libdatafifo_big-little.a** and **libdatafifo_single.a**

[Note]

For details about the control commands and corresponding parameters, see the table below.

Command	Parameter and Description
DATAFIFO_CMD_GET_PHY_ADDR	Returns the physical address of the DATAFIFO module (in type of HI_U32).
DATAFIFO_CMD_READ_DONE	Updates the head and tail pointers of the read operation after data reading is finished. There is no return value. The parameter can be HI_NULL.
DATAFIFO_CMD_WRITE_DONE	Updates the tail pointer of the write operation after data writing is finished. There is no return value. The parameter can be HI_NULL.
DATAFIFO_CMD_SET_DATA_RELEASE_CALLBACK	Data release callback function
DATAFIFO_CMD_GET_AVAIL_WRITE_LEN	Returns the amount of the writable data (in type of HI_U32).
DATAFIFO_CMD_GET_AVAIL_READ_LEN	Returns the amount of the readable data (in type of HI_U32).

[Example]

None

[See Also]

None

5 Data Structures

5.1 IPCMSG

The data structures are defined as follows:

- HI_IPCMSG_MAX_CONTENT_LEN: Defines the maximum length of the message body.
- HI_IPCMSG_PRIVDATA_NUM: Defines the maximum amount of the private data in the message body.
- HI_IPCMSG_INVALID_MSGID: Defines the invalid message ID.
- HI_IPCMSG_MAX_SERVICENAME_LEN: Defines the maximum length of the service name.
- HI_IPCMSG_PORT_HIMPP: Defines the service port number used by HiMPP.
- HI_IPCMSG_PORT_FAKEFS: Defines the service port number used by FAKEFS.
- HI_IPCMSG_CONNECT_S: Defines the structure connecting to the peer server.
- HI_IPCMSG_MESSAGE_S: Defines the message structure.
- HI_IPCMSG_HANDLE_FN_PTR: Defines the message handling function.
- HI_IPCMSG_RESPHANDLE_FN_PTR: Defines the reply message handling function.

HI_IPCMSG_MAX_CONTENT_LEN

[Description]

Defines the maximum length of the message body.

[Syntax]

```
#define HI_IPCMSG_MAX_CONTENT_LEN (1024)
```

[Note]

None

[See Also]

HI_DATAFIFO_Close

HI_IPCMSG_PRIVDATA_NUM

[Description]

Defines the maximum amount of the private data in the message body.

[Syntax]

```
#define HI_IPCMSG_PRIVDATA_NUM (8)
```

[Note]

None

[See Also]

HI_DATAFIFO_Close

HI_IPCMSG_INVALID_MSGID

[Description]

Defines the invalid message ID.

[Syntax]

```
#define HI_IPCMSG_INVALID_MSGID (0xFFFFFFFFFFFFFFFF)
```

[Note]

None

[See Also]

None

HI_IPCMSG_MAX_SERVICENAME_LEN

[Description]

Defines the maximum length of the service name.

[Syntax]

```
#define HI_IPCMSG_MAX_SERVICENAME_LEN (16)
```

[Note]

None

[See Also]

None

HI_IPCMSG_PORT_HIMPP

[Description]

Defines the service port number used by HiMPP.

[Syntax]

```
#define HI_IPCMSG_PORT_HIMPP (1)
```

[Note]

None

[See Also]

None

HI_IPCMSG_PORT_FAKEFS

[Description]

Defines the service port number used by FAKEFS.

[Syntax]

```
#define HI_IPCMSG_PORT_FAKEFS (2)
```

[Note]

None

[See Also]

None

HI_IPCMSG_CONNECT_S

[Description]

Defines the structure connecting to the peer server.

[Syntax]

```
typedef struct hiIPCMSG_CONNECT_S
{
    HI_U32 u32RemoteId;
    HI_U32 u32Port;
    HI_U32 u32Priority;
} HI_IPCMSG_CONNECT_S;
```

[Member]

Member	Description
u32RemoteId	Enumerated value of the remotely connected CPU • 0: master CPU on which main applications run • 1: slave CPU on which the media drive runs
u32Port	User-defined port number used for message communication Value range: [0, 512]

Member	Description
u32Priority	Priority of message transfer Value range: [0, 1] 0: ordinary priority 1: high priority Default value: 0

[Note]

- If the message transfer with a high priority is required, **u32Priority** of the TX and RX ends must be set to **1**.
- Messages with high priorities are transferred in interrupt mode. If the frequency of sending high-priority messages is high, the overall system performance may deteriorate.

[See Also]

None

HI_IPCMSG_MESSAGE_S

[Description]

Defines the message structure.

[Syntax]

```
typedef struct hiIPCMSG_MESSAGE_S
{
   HI_BOOL bIsResp;    /**<Identify the response message*/
   HI_U64 u64Id;      /**<Message ID*/
   HI_U32 u32Module;   /**<Module ID, user-defined*/
   HI_U32 u32CMD;     /**<CMD ID, user-defined*/
   HI_S32 s32RetVal;   /**<Retrun Value in response message*/
   HI_S32 as32PrivData[HI_IPCMSG_PRIVDATA_NUM]; /**<Private data, can be
modified directly after ::HI_IPCMSG_CreateMessage
or ::HI_IPCMSG_CreateRespMessage*/
   HI_VOID* pBody;     /**<Message body*/
   HI_U32 u32BodyLen;  /**<Length of pBody*/
} HI_IPCMSG_MESSAGE_S
```

[Member]

Member	Description
bIsResp	Member indicating whether the message body is a reply message ● HI_TRUE: reply ● HI_FALSE: not reply

Member	Description
u64Id	Message ID
u32Module	Module ID
u32CMD	Command ID
s32RetVal	Return value
as32PrivData	Private data
pBody	Pointer to the message body
u32BodyLen	Length of the message body (in byte)

[Note]

None

[See Also]

None

HI_IPCMSG_HANDLE_FN_PTR

[Description]

Defines the message handling function.

[Syntax]

```
typedef void (*HI_IPCMSG_HANDLE_FN_PTR)(HI_S32 s32Id, HI_IPCMSG_MESSAGE_S
*pstMsg);
```

[Member]

Member	Description
s32Id	Message service ID
pstMsg	Pointer to the message body

[Note]

None

[See Also]

HI_IPCMSG_DestroyMessage

HI_IPCMSG_RESPHANDLE_FN_PTR

[Description]

Defines the reply message handling function.

[Syntax]

```
typedef void (*HI_IPCMSG_RESPHANDLE_FN_PTR)(HI_IPCMSG_MESSAGE_S *pstMsg);
```

[Member]

Member	Description
pstMsg	Pointer to the message body

[Note]

None

[See Also]

None

5.2 DATAFIFO

The data structures are defined as follows:

- HI_DATAFIFO_HANDLE: Defines the handle to the DATAFIFO module.
- MAX_NAME_LEN: Defines the maximum length of the data channel name.
- HI_DATAFIFO_INVALID_HANDLE: Defines the invalid handle to the data channel.
- HI_DATAFIFO_RELEASESTREAM_FN_PTR: Defines the stream release function of the data channel.
- HI_DATAFIFO_OPEN_MODE_E: Defines the open mode of the data channel.
- HI_DATAFIFO_PARAMS_S: Defines the configuration parameters of the data channel.
- HI_DATAFIFO_CMD_E: Defines the control type of the data channel.

HI_DATAFIFO_HANDLE

[Description]

Defines the handle to the DATAFIFO module.

[Syntax]

```
typedef HI_U32 HI_DATAFIFO_HANDLE;
```

[Note]

None

[See Also]

None

MAX_NAME_LEN

[Description]

Defines the maximum length of the data channel name.

[Syntax]

```
#define MAX_NAME_LEN (16)
```

[Note]

None

[See Also]

None

HI_DATAFIFO_INVALID_HANDLE

[Description]

Defines the invalid handle to the data channel.

[Syntax]

```
#define HI_DATAFIFO_INVALID_HANDLE (-1)
```

[Note]

None

[See Also]

None

HI_DATAFIFO_RELEASESTREAM_FN_PTR

[Description]

Defines the stream release function of the data channel.

[Syntax]

```
typedef void (*HI_DATAFIFO_RELEASESTREAM_FN_PTR)(void *pStream)
```

[Note]

None

[See Also]

None

HI_DATAFIFO_OPEN_MODE_E

[Description]

Defines the open mode of the data channel.

[Syntax]

```
typedef enum hiDATAFIFO_OPEN_MODE_E
{
    DATAFIFO_READER,
    DATAFIFO_WRITER
} HI_DATAFIFO_OPEN_MODE_E
```

[Member]

Member	Description
DATAFIFO_READER	Member only used for reading data
DATAFIFO_WRITER	Member only used for writing data

[Note]

None

[See Also]

None

HI_DATAFIFO_PARAMS_S

[Description]

Defines the configuration parameters of the data channel.

[Syntax]

```
typedef struct hiDATAFIFO_PARAMS_S
{
    HI_U32 u32EntriesNum; /**<The number of items in the ring buffer*/
    HI_U32 u32CacheLineSize; /**<Item size*/
    HI_BOOL bDataReleaseByWriter; /**<Whether the data buffer release by
writer*/
    HI_DATAFIFO_OPEN_MODE_E enOpenMode; /**<READER or WRITER*/
} HI_DATAFIFO_PARAMS_S
```

[Member]

Member	Description
u32EntriesNum	Amount of data in the cyclic buffer
u32CacheLineSize	Size of each data item
bDataReleaseByWriter	Member used to indicate whether the writer needs to release the data
enOpenMode	Member used to open a data channel

[Note]

The value ranges of **u32EntriesNum** and **u32CacheLineSize** are not restricted. As long as the MMZ memory is large enough, a data FIFO can be created successfully. Therefore, ensure that these two parameters are within a proper range.

[See Also]

None

HI_DATAFIFO_CMD_E

[Description]

Defines the control type of the data channel.

[Syntax]

```
typedef enum hiDATAFIFO_CMD_E
{
    DATAFIFO_CMD_GET_PHY_ADDR, /**<Get the physic address of ring buffer*/
    DATAFIFO_CMD_READ_DONE, /**<When the read buffer read over, the reader
should call this function to notify the writer*/
    DATAFIFO_CMD_WRITE_DONE, /**<When the writer buffer has finished the write
operation, the writer should call this function*/
    DATAFIFO_CMD_SET_DATA_RELEASE_CALLBACK, /**<When bDataReleaseByWriter
is HI_TRUE, the writer should call this to register release callback*/
    DATAFIFO_CMD_GET_AVAIL_WRITE_LEN, /**<Get available write length*/
    DATAFIFO_CMD_GET_AVAIL_READ_LEN /**<Get available read length*/
} HI_DATAFIFO_CMD_E;
```

[Member]

Member	Description
DATAFIFO_CMD_GET_PHY_ADDR	Obtaining the physical address of the data channel
DATAFIFO_CMD_READ_DONE	Notifying that the read operation is completed
DATAFIFO_CMD_WRITE_DONE	Notifying that the write operation is completed
DATAFIFO_CMD_SET_DATA_RELEASE_CALLBACK	Setting the data release callback function
DATAFIFO_CMD_GET_AVAIL_WRITE_LEN	Obtaining the length of the writable data
DATAFIFO_CMD_GET_AVAIL_READ_LEN	Obtaining the length of the readable data

[Note]

None

[See Also]

None

5.3 Error Codes

Table 5-1 lists the error codes of IPCMSG APIs.

Table 5-1 Error codes of IPCMSG APIs

Error Code	Macro Definition	Description
0x1901	HI_IPCMSG_EINVAL	The configuration parameter is invalid.
0x1902	HI_IPCMSG_ETIMEOUT	A timeout error occurs.
0x1903	HI_IPCMSG_ENOOP	The driver fails to be enabled.
0x1904	HI_IPCMSG_EINTER	An internal error occurs.
0x1905	HI_IPCMSG_ENULL_PTR	The pointer is null.
0x00000000	HI_SUCCESS	The operation is successful.
0xFFFFFFFF	HI_FAILURE	The operation fails.

Table 5-2 lists the error codes of DATAFIFO APIs.

Table 5-2 Error codes for DATAFIFO APIs

Error Code	Macro Definition	Description
0x00000000	HI_SUCCESS	The operation is successful.
0xFFFFFFFF	HI_FAILURE	The operation fails.