Embedded 1394 LynxSoft Hardware Abstraction Layer (HAL) For MPEG2Lynx and GPLynx - Programmers Interface User's Guide

APPLICATION REPORT: SLLA022

Richard Solomon
BuS Solutions Software Group

Mixed Signal and Logic Productss
BuS Solutions Group
15 December 1997

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

GPLynx, PCILynx, Lynxsoft, Mpeg2Lynx, 1394 TImes are trademarks of Texas Instruments Incorporated

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com
US 1394 email	1394@ti.com
US 1394 web site	http://www.ti.com/sc/1394

Contents

Figures

Embedded 1394 LynxSoft Hardware Abstraction Layer (HAL) For MPEG2Lynx and GPLynx - Programmers Interface User's Guide

Abstract

This Guide describes the Hardware Abstraction Layer (HAL) software (LynxSoft) for Embedded Systems that use the GPLynx (TSB12LV31) and MPEG2Lynx (TSB12LV41) IEEE 1394 Link-Layer Controllers. Topics covered include a high level description of the HAL and a description of each public function provided by the HAL.

Product Support

Related Documentation

The following list specifies product names, part numbers, and literature numbers of corresponding TI documentation.

- *Data Manual TSB12LV31*, Literature number SLLS255
- *Data Manual TSB12LV41*, Literature number SLLS276A
- *IEEE Standard for a High Performance Serial Bus, IEEE Std 1394-1995, ISBN 1-55937-583-3*

World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

The URL specifically for the TI 1394 external web site is http://www.ti.com/sc/1394. On this page users can subscribe to 1394 TImes, which periodically updates subscribers on events, articles, products, and other news regarding 1394 developments.

Email

For technical issues or clarification on products, please send a detailed email to 1394@ti.com.

Overview

The **Embedded LynxSoft HAL** (Hardware Abstraction Layer) is the lowest layer of software before the Lynx Hardware. Its purpose is to provide an interface of the essential functions for Embedded Lynx hardware. These functions will isolate hardware details from the client software modules. This creates a stable programming environment for system-level software designers and a flexible environment for hardware designers.

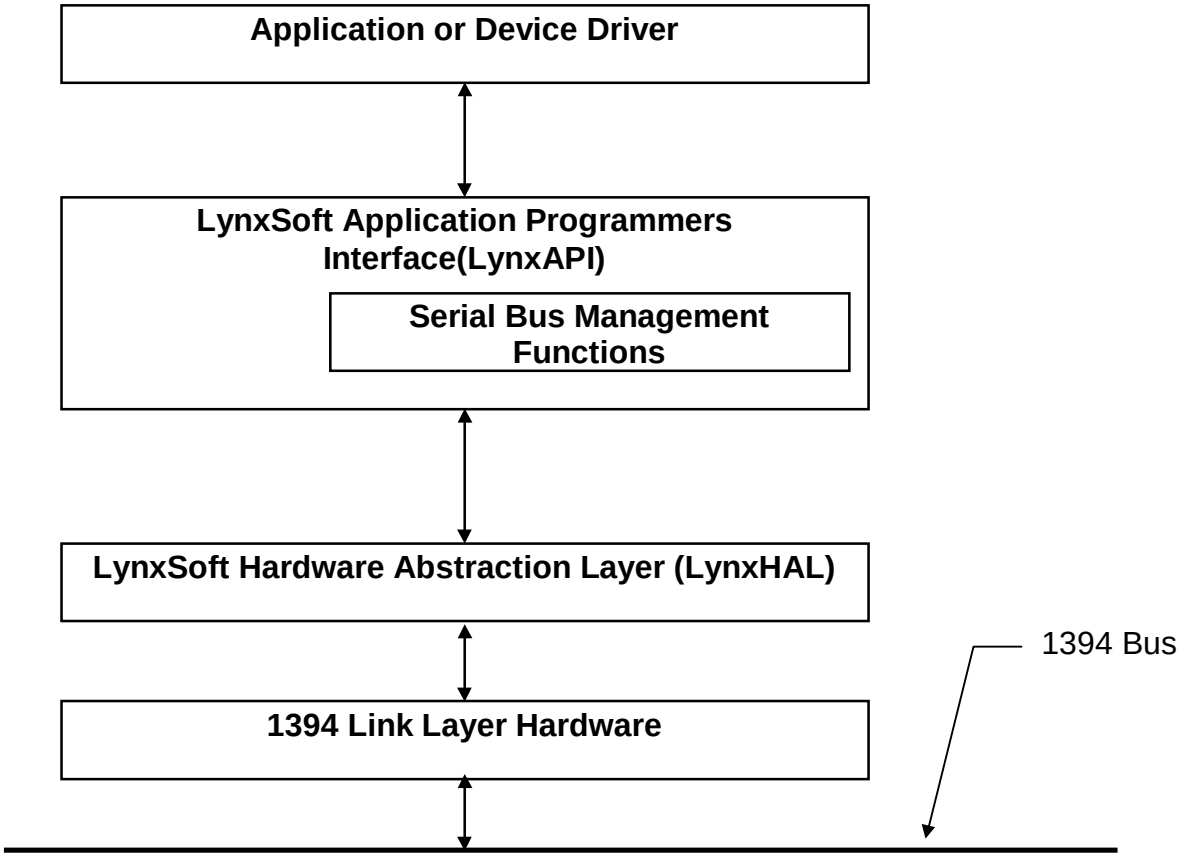
This document assumes that the reader is familiar with IEEE Std 1394-1995. Readers may need the Data Manuals for TSB12LV31 and/or TSB12LV41 for reference while considering the details of function prototypes and data structures presented here.

The HAL provides no 1394 Bus Management services. It provides minimal event notification services to client software modules.

Architecture

The architecture of the expected use for the LynxSoft HAL is shown below.

Figure 1. LynxSoft HAL Architecture



Software Portability

It is expected that the LynxHAL software will be used in any number of different hardware and software environments, and hence, software portability was a consideration during the development of LynxHAL. Some portability issues that an embedded system developer should consider are listed below.

Interface to Hardware.

The developer should understand the Link Layer Controller hardware interface with the target system. The file LynxHALReg.h has the macros, and LynxHALReg.c has the functions, used to read and write Lynx hardware registers. These macros and functions should be optimized for the target system.

Operating System and ANSI C.

LynxHAL software was written in ANSI C and without an Operating System. If an OS will be used in the target system, the developer needs to consider the most efficient use of the OS with the LynxHAL. How the OS handles interrupts, queued processes, and the time line to use these OS features are some of the things to consider.

Interrupts and Callbacks.

Whether an OS will be used or not, the developer needs to consider how interrupts and callback functions will be handled by the target system.

Memory Usage.

Because LynxHAL software does not know the memory limitation of the target system, the LynxHAL expects the calling function to provide all run time memory space. The calling function needs to create and return the structures used for sending async packets, and create and return the space needed for incoming async packets.

Callback Services

There are two levels of callbacks needed for the LynxHAL to operate. One level connects the node interrupts into the LynxHAL software and the other connects the LynxHAL into the software layer above, the API layer. Callbacks used for connecting interrupts are named LynxHALxxxxCallback and these functions are a part of the LynxHAL software. Callbacks into the API layer are passed into LynxHAL during initialization. They are passed as pointers to functions that are a part of the API layer. These API layer callback functions are called by the LynxHAL to notify the API layer that it needs to deal with a bus event.

There are four types of callback functions:

- Bus Reset - happens when the bus changes
- I/O Complete - when an Async packet is sent to the bus(GRF goes empty)
- Bus Event Indication - incoming Async packets
- Error Detected - error caused by incoming packets(Iso or Async) and Iso cycle conditions that cause Iso packets not to be sent.

These functions will be discussed in detail in the following sections.

Callback Services

The following are more detailed descriptions of the callback functions.

These callback functions need to be connected to the interrupts that are detected by LynxHALNodeISR, the 1394 interrupt handler. How this is accomplished is dependent on the environment in which the LynxHAL will run.

Some (but not all) examples are: function calls, software interrupts, or queuing a process if an OS is in use.

LynxHAL Bus Reset Callback

Description:

LynxHAL callback for bus resets.

Action:

Whenever a bus reset occurs, an interrupt is generated to notify software that the bus has been reconfigured.
LynxHALBusResetCallback is the function that is called to notify LynxHAL of the bus reset.This function will wait for the Self-ID Period to end, check for errors, fill in the Bus Reset structureprovided by the API and then call the Bus Reset Callback function provided by the API.

Syntax:

```
void LynxHALBusResetCallback();
```

Parameters:

None.

Return Status:

None.

LynxHAL Indication Callback

Description:

LynxHAL callback for bus event indications.

Action:

When an Async packet is received from a remote node, an interrupt is generated to notify software.
LynxHALIndicationCallback is the function that is called when this interrupt occurs. This function will call the API Callback function that will handle the incoming packet.

Syntax:

```
void LynxHALIndicationCallback();
```

Parameters:

None.

Return Status:

None.

LynxHAL I/O Complete Callback

Description:

LynxHAL callback when an Async packet has been sent.

Action:

When an Async packet has been sent to the bus, the ATF becomes empty. **LynxHALIoCompleteCallback** is the function that is called to notify LynxHAL of the Async packet leaving the ATF. This function will verify that the ATF is empty, that the packet was acknowledged, and call the API Callback function that needs to know that the packet was sent.

Syntax:

```
void LynxHALIoCompleteCallback();
```

Parameters:

None.

Return Status:

None.

LynxHAL Error Detected Callback

Description:

LynxHAL callback when an error is detected.

Action:

When the Link layer hardware generates an interrupt indicating that an error was detected, the Interrupt Service Routine (ISR) determines which one of the errors caused this interrupt and saves an error code for that error. The function **LynxHALErrorDetectedCallback** is called by the ISR with this error code. **LynxHALErrorDetectedCallback** then calls the API error callback function using the callback pointer with the error code as a prameter.

Syntax:

```
void LynxHALErrorDetectedCallback( int iIntrErrorCode );
```

Parameters:

```
int iIntrErrorCode   /* Interrupt error code.*/
```

Return Status:

None.

Initialization and Termination Functions

LynxHALInit should be the first function called and **LynxHALTerminate** should be the last.

LynxHALInit

Description:

Initialize LynxHAL software and Lynx Hardware.

Action:

This function must be called prior to issuing any calls to the HAL.

The **LynxHALInit** routine prepares the LynxHAL software and hardware for use. This function accepts a pointer to a **LYNXHAL_BUS_RESET_BLOCK** as a parameter, which is described below. The caller is expected to create this structure. The HAL will fill in the values of this structure after each bus reset and then call LynxHALBusResetCallback.

Syntax:

```
int  LynxHALInit( LYNXHAL_BUS_RESET_INFO*  pInitInfo );
```

Parameters:

```
typedef struct
   {
/* Inputs from caller:*/
void     (*pBusResetCallback)(); /* Pointers to caller's callback
                                                        functions.*/
void     (*pIoCompleteCallback)();
void     (*pIndicationCallback)();
void     (*pErrorCallback)();

/* Outputs to caller:*/
ULONG  ulBusResetCount;   /* The number of bus resets this node has
                                                        detected.*/
ULONG  ulResetCycleTime;  /* Contents of the CYCLE_TIME register at
                                                        bus reset.*/
UINT   uiBusResetError;   /* Indicates error in Bus Reset process.*/
UINT   uiSelfIDErrorCode; /* Self ID packet error code detected by
                                                        Lynx chip.*/
UINT   uiNodeID;          /* This node's current node ID(phy ID).*/
UINT   uiBusID;           /* This node's current bus ID.*/
UINT   uiIRMNodeID;       /* The current IRM' node ID.*/
ULONG  ulNodeCount;       /* Number of nodes on the 1394 bus.*/
BOOL   bRoot;             /* Set True if this node is root. */
```

```
BOOL   bContender;        /* Set True if this node is a contender.*/

   } LYNXHAL_BUS_RESET_INFO;
```

Return Status:

This function returns zero to indicate no errors. Non zero return indicates an error occurred.

LynxHALTerminate

Description:

Turn off HAL SW and HW.

Action:

If the upper software layers must perform an orderly shutdown, it must de-initialize the HAL during its shutdown operation using **LynxHALTerminate.** The HAL removes any interrupt that it installed and disables the Lynx interface hardware so that it is no longer able to generate CPU interrupts or access system memory.

Syntax:

```
void LynxHALTerminate();
```

Parameters:

None.

Return Status:

None.

Asynchronous Data Transfer Services

The Lynx HAL provides Asynchronous data transfer services in the form of sending Asynchronous packets, saving incoming Asynchronous packets, saving Self-ID packets and sending Phy packets.

LynxHALSendAsyncPacket

Description:

Send Async packet.

Action:

This routine causes the HAL to send an Async transaction packet.

When this function returns , without error, the packet has been loaded into the ATF(Asynchronous Transmit FIFO) by the HAL. If the ATF is not empty, because another packet is waiting to be sent, this function will return with an error code indicating this condition.

The HAL will call the API's I/O Complete CallBack function when the packet has been sent.

Syntax & Parameters:

```
int LynxHALSendAsyncPacket(
   int     iHeaderSizeQuad, /* Size in quads of following header. */
   ULONG*  pulHeader,      /*Points to the Async header to be sent*/
   int     iDataSizeQuad,  /* Size in quads of following data.    */
   ULONG*  pulData,        /* Points to the data to be sent.      */
   ULONG*  pulAckReturned  /* Points to place to put returned ack.*/
   );
```

Return Status:

This function returns a zero when the packet has been loaded into the ATF. Non zero return indicates an error occurred.

LynxHALSendPhyPacket

Description:

Send a PHY packet

Action:

This function will send a PHY packet. Only the PHY Configuration and Link On packets can be sent. These packets are described in clause 4.3.4.2 and 4.3.4.3 of the 1394 Spec.

When this function returns, without error, the packet has been loaded into the ATF (Asynchronous Transmit FIFO) by the HAL. If the ATF is not empty, because another packet is waiting to be sent, this function will return with an error code indicating this condition.

Syntax:

```
int LynxHALSendPhyPacket( ULONG    ulPhyPacket );
```

Parameters:

```
ulPhyPacket /*contains the first 32 bit word of the PHY packet*/
```

Return Status:

This function returns a zero when completed with no errors. Non zero return indicates an error occurred.

LynxHALSaveAsyncPacket

Description:

Remove and save Async packet from receive FIFO.

Action:

This function will move Async and Self-ID packets from the receive FIFO into the callers memory space.

Syntax:

```
int LynxHALSaveAsyncPacket( ULONG* pAPacketStorage );
```

Parameters:

```
ULONG* pAPacketStorage /* points to caller memory space. Size   */
                       /* will be max receive Async packet size.*/
```

Return Status:

This function returns a zero when completed with no errors. Non zero return indicates an error occurred.

LynxHALSaveSelfIDPacket

Description:

Save Self ID packets.

Action:

This function will move Self-ID packets from the receiving FIFO into the callers memory space.

Syntax:

```
int LynxHALSaveSelfIDPacket( ULONG* pAPacketStorage );
```

Parameters:

```
ULONG* pAPacketStorage /* points to caller memory space.  Size  */
                       /* will be max receive Async packet size.*/
```

Return Status:

This function returns a zero when completed with no errors. Non zero return indicates an error occurred.

Isochronous Data Transfer Services

The Lynx HAL provides Isochronous data transfer services in the form of sending (talk) Isochronous packets/data streams, receiving (listen) incoming Isochronous packets/data streams, stopping talking or listening and scausing this node to be ycle master.

LynxHALStartCycleMaster

Description:

Start Cycle Master.

Action:

This function causes the node to become Cycle Master and provides the option for the node to become Cycle Master automatically after a bus reset if the node is still root.

Syntax:

```
int LynxHALStartCycleMaster( BOOL  bSetAutoStart );
```

Parameters:

```
bSetAutoStart - Set true to cause automatic start of
                    Cycle Master after a bus reset if node is root.
```

Return Status:

This function returns a zero when completed with no errors. Non zero return indicates an error occurred.

LynxHALTalk – GPLynx Only

Description:

Start Isochronous data stream transmit.

Action:

This function configures the Lynx Hardware to begin transmitting Isochronous packets/data stream.

Syntax:

```
int LynxHALTalk( int   iChannelNumber,
                 int   iTag,
                 int   iSyncBits,
                 int   iPacketDataSizeBytes,
                 long  lQuadletsPerFrame,
                 int   iSpeed
                 );
```

Parameters:

```
int   iChannelNumber,        /* Part of Iso header. 0-63        */
int   iTag,                  /* Part of Iso header.             */
int   iSyncBits,             /* Part of Iso header.             */
int   iPacketDataSizeBytes,  /* Part of Iso header.             */
long  lQuadletsPerFrame,     /* Number of data quadlets between */
                             /*  setting sync field.            */
int   iSpeed             /* Speed at which packet will be sent. */
```

Return Status:

This function returns a zero when completed with no errors. Non zero return indicates an error occurred.

LynxHALListen – GPLynx Only

Description:

Start Isochronous data stream receive

Action:

This function configures the Lynx Hardware to begin receiving Isochronous packets/data stream.

Syntax:

```
int LynxHALListen(  int   iFirstChannelNumber,
                    int   iFirstTag,
                    int   iSecondChannelNumber,
                    int   iSecondTag,
                    int   iSyncBits,
                    BOOL  bReceiveAl
                  );
```

Parameters:

```
int  iFirstChannelNumber,    /* Channel number of Iso packet   */
                             /*    excepted by receiver. 0-63. */
int  iFirstTag,              /* Tag number of Iso packet       */
                             /*    excepted by receiver.       */
int  iSecondChannelNumber,   /* Channel number of Iso packet   */
                             /*    excepted by receiver. 0-63. */
int  iSecondTag,             /* Tag number of Iso packet       */
                             /*    excepted by receiver.       */
int  iSyncBits,              /* When SyncBits match Sy field   */
           /* of incoming Iso packet, hardware pin ISORST pulses.*/
BOOL bReceiveAll             /* When true, All Iso packets are */
                             /*    received.                   */
```

Return Status:

This function returns a zero when completed with no errors. Non zero return indicates an error occurred.

LynxHALStop – GPLynx Only

Description:

Stop sending or receiving Isochronous data stream

Action:

This function configures the Lynx Hardware to stop sending or receiving Isochronous packets/data stream.

Syntax:

```
int LynxHALStop( void );
```

Parameters:

None.

Return Status:

This function returns a zero when completed with no errors. Non zero return indicates an error occurred.

LynxHALAllocateLynxResources – MPEG2Lynx Only

Description:

Allocates MPEG2Lynx's FIFOs.

Action:

This function configures sizes of the MPEG2Lynx's FIFOs.

Syntax:

```
int LynxHALAllocateLynxResources( BULKY_FIFO_SIZE_INFO* pSizeInfo );
```

Parameters:

```
/* Structure contains sizes of Lynx FIFOs. Each FIFO can not be */
/* greater than 4092 bytes and total of all FIFOs must be less */
/* than or equal to 8192 bytes.*/

typedef struct
{
                int iMpegDssTalkFifoSizeQuads;
                int iMpegDssListenFifoSizeQuads;
                int iIsoTalkSizeQuads;
                int iIsoListenSizeQuads;
                int iAsyncTalkFifoSizeQuads;
                int iAsyncListenFifoSizeQuads;
} BULKY_FIFO_SIZE_INFO;
```

Return Status:

This function returns a zero when completed with no errors. Non zero return indicates an error occurred.

NOTE:
This is in development.

LynxHALMpegDssMode – MPEG2Lynx Only

Description:

Setup mode of MPEG2 data stream to be either MPEG2 or DSS.

Action:

This function configures the MPEG2Lynx Hardware MPEG2 FIFOs to handle either MPEG2 or DSS data.

Syntax:

```
void LynxHALMpegDssMode( enum E_MODE eMode );
```

Parameters:

```
typedef enum E_MODE { MPEG2, DSS };

enum E_MODE eMode;
/* eMode is set to either MPEG2 or DSS.*/
```

Return Status:

None.

NOTE:
This is in development.

LynxHALMpegDssListen – MPEG2Lynx Only

Description:

Setup to receive MPEG2/DSS data stream.

Action:

This function configures the MPEG2Lynx Hardware MPEG2 FIFO to receive data.

Syntax:

```
void LynxHALMpegDssListen(
     int      iListenChannelNumber,
     ULONG    ulListenTimestampOffset,
     BOOL     bStripHeaders, /*Set True to remove headers*/
     BOOL     bListenAging   /* Set True to turn on Listen aging.*/
);
```

Parameters:

```
int    iListenChannelNumber,    /* Iso channel to listen.*/
ULONG  ulListenTimestampOffset, /* Timestamp offset.*/
BOOL   bStripHeaders,         /* Set True to remove headers.*/
BOOL   bListenAging           /* Set True to turn on Listen aging.*/
```

Return Status:

None.

NOTE:
This is in development.

LynxHALMpegDssListenStop – MPEG2Lynx Only

Description:

Stop MPEG2/DSS data stream listening.

Action:

This function configures the MPEG2Lynx Hardware MPEG2 FIFO to stop receiving MPEG2 or DSS data.

Syntax:

```
void LynxHALMpegDssListenStop( void );
```

Parameters:

None.

Return Status:

None.

NOTE:
This is in development.

LynxHALMpegDssTalk – MPEG2Lynx Only

Description:

Setup to send MPEG2/DSS data stream.

Action:

This function configures the MPEG2Lynx Hardware MPEG2 FIFO to send data.

Syntax:

```
void LynxHALMpegDssTalk(
       int     iMpegClass,  /* Class 0 through 7 */
       enum    E_INSERTS  eInserts,
       ULONG   ulTalkTimestampOffset,
       BOOL    bTalkAging,  /* Set TRUE to turn on Talk Aging.*/
       ULONG   ulTalkIsoHeader,
       ULONG   ulTalkCIPHeaderDataBlock,
       ULONG   ulTalkCIPHeaderFormat
            );
```

Parameters:

```
typedef enum E_INSERTS { TIMESTAMP, ISO_CIP_HEADER, TIME_ISO_CIP };

       int     iMpegClass,     /* Class 0 through 7 */
       enum    E_INSERTS  eInserts,
       ULONG   ulTalkTimestampOffset,
       BOOL    bTalkAging,     /* Set TRUE to turn on Talk Aging.*/
       ULONG   ulTalkIsoHeader,
       ULONG   ulTalkCIPHeaderDataBlock,
       ULONG   ulTalkCIPHeaderFormat
```

Return Status:

None.

NOTE:
This is in development.

LynxHALMpegDssTalkStop – MPEG2Lynx Only

Description:

Stop MPEG2/DSS data stream sending.

Action:

This function configures the MPEG2Lynx Hardware MPEG2 FIFO to stop sending MPEG2 or DSS data.

Syntax:

```
void LynxHALMpegDssTalkStop( void );
```

Parameters:

None.

Return Status:

None.

NOTE:
This is in development.

LynxHALStreamIsoListen – MPEG2Lynx Only

Description:

Start receiving Isochronous data stream using the Bulky Data Interface.

Action:

This function configures the MPEG2Lynx Hardware to receive Isochronous data.

MPEG2Lynx can receive from one to seven isochronous channels. This function can be called up to seven times with iPort set to indicate the number of the call(1, 2,... or 7). sbiStreamIsoListenStop must be called to stop listening for a isochronous channel.

Syntax:

```
int LynxHALStreamIsoListen  (
                            int   iPort,
                            int   iChannelNumber,
                            BOOL  bMatchOnTag,
                            int   iTag
                            );
```

Parameters:

```
int   iPort,        /* 1 to 7.*/
int   iChannelNumber, /* Channel number of Iso /*
           /* packet excepted by receiver. 0-63.*/
BOOL  bMatchOnTag,  /* Set true to receive when*/
                 /* Tag and channel matches,*/
int   iTag       /*0 to 3, Tag number of Iso*/
                 /* packet excepted by receiver. */
```

Return Status:

This function returns a zero when completed with no errors. Non zero return indicates an error occurred.

NOTE:
This is in development.

LynxHALStreamIsoListenStop – MPEG2Lynx Only

Description:

Stop receiving Isochronous data stream for a iPort number.

Action:

This function configures the MPEG2Lynx Hardware Isochronous FIFO to stop sending.

Syntax:

```
int LynxHALStreamIsoListenStop( int   iPort );
```

Parameters:

```
int   iPort /* Port number to stop listening.*/
```

Return Status:

This function returns a zero when completed with no errors. Non zero return indicates an error occurred.

NOTE:
This is in development.

LynxHALStreamIsoTalk – MPEG2Lynx Only

Description:

Start sending an Isochronous data stream.

Action:

This function configures the MPEG2Lynx Hardware Isochronous FIFO to sending data that comes from the Bulky Data Interface.

Syntax:

```
void  LynxHALStreamIsoTalk(
        int   iChannelNumber,
        int   iTag,
        int   iSyncBits,
        int   iPacketDataSizeBytes,
        int   iSpeed
        );
```

Parameters:

```
int   iChannelNumber,         /* Part of Iso header. 0-63*/
int   iTag,                   /* Part of Iso header.*/
int   iSyncBits,              /* Part of Iso header.*/
int   iPacketDataSizeBytes,   /* Part of Iso header.*/
int   iSpeed                  /* Speed at which packet will be
sent.*/
```

Return Status:

None.

NOTE:
This is in development.

LynxHALStreamIsoTalkStop – MPEG2Lynx Only

Description:

Stop sending an Isochronous data stream.

Action:

This function configures the MPEG2Lynx Hardware Isochronous FIFO to stop sending data that comes from the Bulky Data Interface.

Syntax:

```
void LynxHALStreamIsoTalkStop( void );
```

Parameters:

None.

Return Status:

None.

NOTE:
This is in development.

LynxHALStreamAsyncListen – MPEG2Lynx Only

Description:

Start receiving an Asynchronous data stream.

Action:

This function configures the MPEG2Lynx hardware Asynchronous FIFO to start receiving asynchronous 1394 data and transfering that data to the Bulky Data Interface.

Syntax:

```
void LynxHALStreamAsyncListen( void );
```

Parameters:

None.

Return Status:

None.

NOTE:
This is in development.

LynxHALStreamAsyncListenStop – MPEG2Lynx Only

Description:

Stop receiving an Asynchronous data stream.

Action:

This function configures the MPEG2Lynx hardware Asynchronous FIFO to stop receiving asynchronous 1394 data and transferring that data to the Bulky Data Interface.

Syntax:

```
void LynxHALStreamAsyncListenStop( void );
```

Parameters:

None.

Return Status:

None.

NOTE:
This is in development.

LynxHALStreamAsyncTalk – MPEG2Lynx Only

Description:

Start sending an Asynchronous data stream.

Action:

This function configures the MPEG2Lynx Hardware Asynchronous FIFO to start sending data that comes from the Bulky Data Interface.

Syntax:

```
void LynxHALStreamAsyncTalk( void );
```

Parameters:

None.

Return Status:

None.

NOTE:
This is in development.

LynxHALStreamAsyncTalkStop – MPEG2Lynx Only

Description:

Stop sending an Asynchronous data stream.

Action:

This function configures the MPEG2Lynx Hardware Asynchronous FIFO to stop sending data that comes from the Bulky Data Interface.

Syntax:

```
void LynxHALStreamAsyncTalkStop( void );
```

Parameters:

None.

Return Status:

None.

NOTE:
This is in development.

Management Functions

The following functions will be used to manage the 1394 bus.

LynxHALGetThisNodesID

Description:

Returns this node's Self-ID packet.

Action:

After each bus reset, this node will have received a self-id packet from every node on the bus except this node. This function creates this Node's Self-ID packet. This will be used by the Bus Manager for the Topology map and generation of the Speed map.

Syntax:

```
ULONG LynxHALGetThisNodesID ();
```

Parameters:

None.

Return Status:

This function returns the Self-ID packet for this node. See clause 4.3.4.1 of the IEEE 1394-1995 Standard for a description of the Self-ID packet.

LynxHALPresentVersion

Description:

Returns the version of the LynxHAL software and hardware.

Action:

Returns the version of the LynxHAL software and hardware.

Syntax:

```
void LynxHALPresentVersion(ULONG* pulSoftwareVersion, ULONG*
                                                    ulHardwareVersion );
```

Parameters:

```
pSoftwareVersion /* Pointer to caller provided memory location */
                 /* that, on return, will contains the start   */
                 /* address to the SW Version character string.*/

pHardwareVersion /* Points to caller provided memory location     */
                 /* that, on return, will contains the HW Version.*/
```

Return Status:

None.

LynxHALGetGapCount

Description:

Returns Gap Count contained in the PHY.

Action:

Reads PHY Gap Count and returns it to the caller.

Syntax:

```
int LynxHALGetGapCount();
```

Parameters:

None.

Return Status:

This function returns an integer value which is the PHY Gap Count.

LynxHALCauseBusReset

Description:

Cause a bus reset.

Action:

Does the writes to the PHY to cause a bus reset.

Syntax:

```
void LynxHALCauseBusReset();
```

Parameters:

None.

Return Status:

None.

LynxHALGetCycleTime

Description:

Returns value in Lynx's Cycle Timer register.

Action:

This function reads and returns the value in Lynx's Cycle Timer register.

Syntax:

```
ULONG LynxHALGetCycleTime();
```

Parameters:

None.

Return Status:

This function returns the value in the Lynx's Cycle Timer register.

LynxHALGetBusID

Description:

Returns value of Bus ID.

Action:

This function reads and returns the value that the Lynx's Hardware is using for Bus ID. Read Clause 8.3.2.2.3 in the 1394 Standard.

NOTE:
This function will be used when bridges between buses are available.

Syntax:

```
ULONG LynxHALGetBusID();
```

Parameters:

None.

Return Status:

This function returns the value in the Bus ID register.

LynxHALSetBusID

Description:

Set Bus ID in Lynx Bus Reset register.

Action:

This function sets the value that the Lynx's Hardware will use for Bus ID. Read Clause 8.3.2.2.3 in the 1394 Standard.

NOTE:
This function will be used when bridges between buses are available.

Syntax:

```
void LynxHALSetBusID( ULONG ulNewBusID );
```

Parameters:

```
ULONG ulNewBusID   /* New Bus ID */
```

Return Status:

None.