Design of the GPLynx HAL

APPLICATION REPORT: SLLA026

Richard Solomon
Mixed Signal Processing

Digital Signal Processing Solutions
June 1998

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 and complete.

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 and LynxSoft 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

Contents

Figures

Design of the GPLynx HAL

Abstract

This document describes a high level design of the GPLynx Hardware Abstraction Layer (HAL) software that runs on the Texas Instruments (TI™) TMS320C52-based GPLynx evaluation module (EVM). This design shows the control and data paths. Note that the purpose of the HAL is to isolate the LynxSoft™ Application Program Interface (API) or user supplied Driver software from the hardware.

HAL operation depends on the driving API calling and using the HAL properly. This design includes a description of what is expected of the driving API.

The design has been divided into Initialization, Bus Reset, Outgoing Asynchronous Packet, and Incoming Asynchronous Packet. A Porting section is added to emphasize the parts of the HAL that may change as the HAL is moved to another system/processor environment.

Product Support

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.

Initialization

After power on or hardware reset, the driving API must initialize the HAL by calling the LynxHALInit function. The API should keep the system's 1394 interrupts turned off until the return from LynxHALInit.

1) The API/Driver must first create a LYNXHAL_BUS_RESET_INFO structure. The HAL will place bus-reset information in this structure after a bus reset. This structure also contains pointers to callback functions that will be called by the HAL to notify the API of certain bus events and errors.

2) The API calls the LynxHALInit function. This function initializes the HAL software.

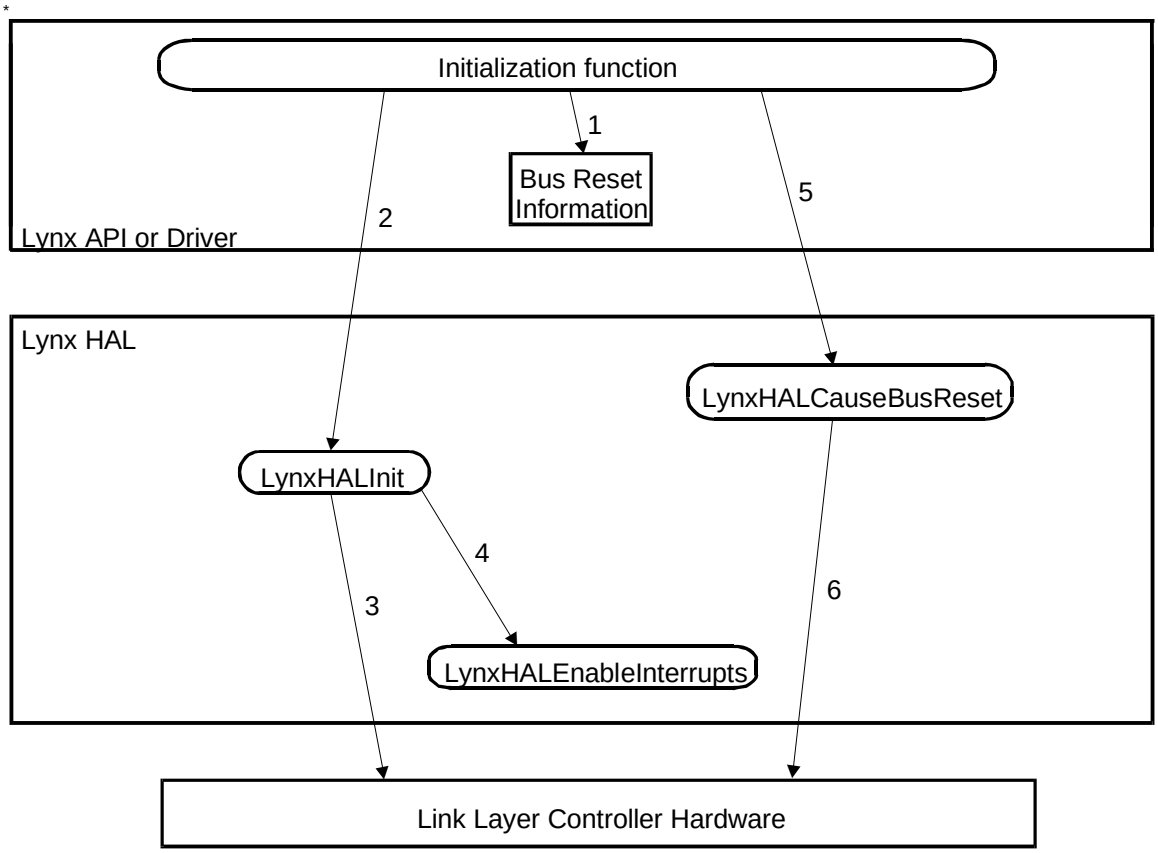
3) LynxHALInit does the initial setup of the GPLynx hardware.

4) The last thing LynxHALInit does is call LynxHALEnableInterrupts to unmask the GPLynx interrupts.

5) After the HAL and the API have been initialized, the API should call LynxHALCauseBusReset.

6) LynxHALCauseBusReset causes the GPLynx Link Layer Controller to generate a 1394 Bus Reset.

Figure 1. Initialization

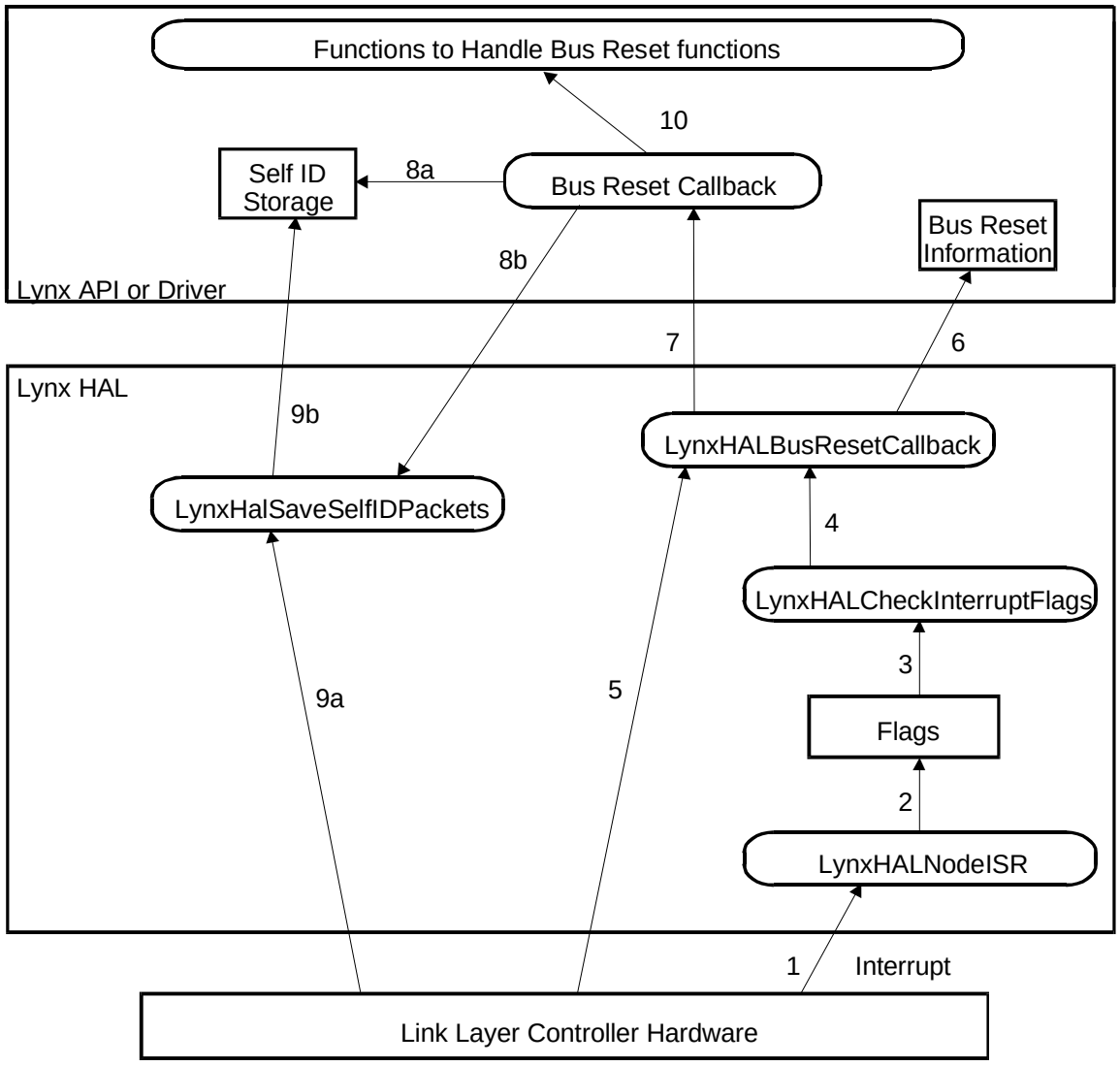


Bus Reset

The 1394 bus will be reset whenever a node is removed or added to the bus, or by software at anytime. The 1394 Hardware will detect that a 1394 bus reset has started and will cause an interrupt.

1) The hardware interrupt causes LynxHALNodeISR to execute.
2) LynxHALNodeISR detects that the PhRst interrupt is set and sets a flag indicating a bus reset has started.
3) LynxHALCheckInterruptFlags is caused to run and checks all of the interrupt flags.
4) LynxHALBusResetCallback is called by LynxHALCheckInterruptFlags.
5) LynxHALBusResetCallback polls hardware registers to determine that Self ID packets have been collected.
6) LynxHALBusResetCallback updates the Bus Reset Information structure.
7) LynxHALBusResetCallback calls the API Bus Reset Callback function.
8) API Bus Reset Callback function creates a place to store Self ID packets and calls LynxHALSaveSelfIDPackets.
9) LynxHALSaveSelfIDPackets removes packets from GRF and puts them into the API provided Self ID storage.
10) The API's Bus Reset Callback function will notify the API's Handle Bus Reset functions that the bus reset is complete and Self ID and Bus Reset information is available.

Figure 2. Bus Reset

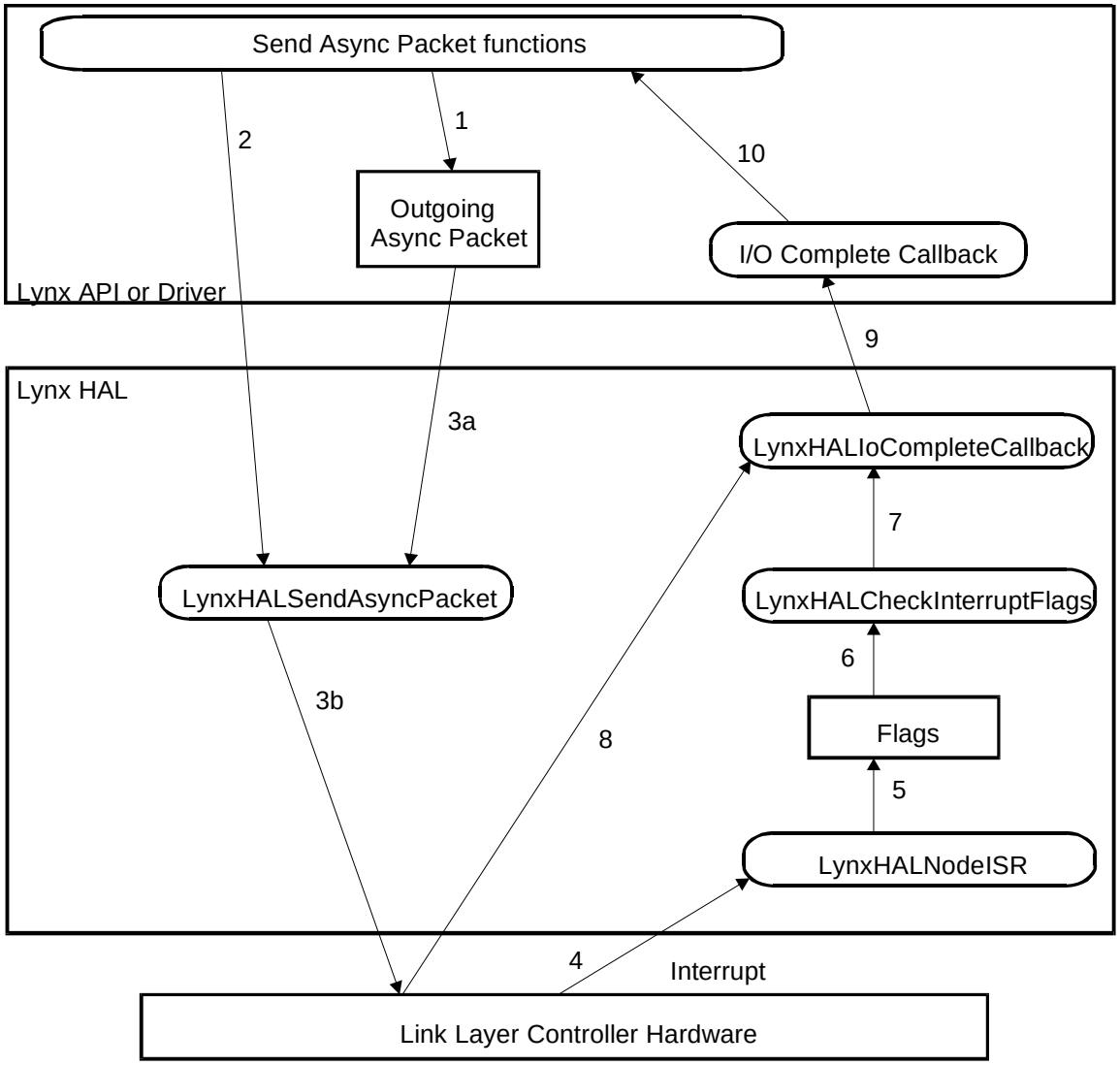


Outgoing Asynchronous Packet

The API/Driver creates the asynchronous packet and then gives it to the HAL to be sent.

1) The API creates the async packet header and data. The hardware will insert the CRC for the header and the data.
2) API calls LynxHALSendAsyncPacket with pointers to the async packet's header and data.
3) LynxHALSendAsyncPacket moves the packet from the API's memory into the GPLynx Transmit FIFO.
4) The packet being sent causes an interrupt. This interrupt causes LynxHALNodeISR to be executed.
5) LynxHALNodeISR sets a flag indicating a packet was sent.
6) When LynxHALCheckInterruptFlags runs, it sees the flag set.
7) LynxHALCheckInterruptFlags calls LynxHALIoCompleteCallback.
8) LynxHALIoCompleteCallback reads GPLynx registers to verify that the transmit FIFO is empty.
9) If the FIFO is empty, LynxHALIoCompleteCallback calls the API's I/O Complete Callback function.
10) The API's I/O Complete Callback function will notify the API's packet sending functions that the packet was sent.

Figure 3. Outgoing Packet



Incoming Asynchronous Packet

The HAL notifies the API/Driver of a packet received into the GPLynx General Receive FIFO (GRF). The API/Driver must process the packet and take the correct action.

1) An interrupt is caused by a packet coming into the GRF. The interrupt causes LynxHALNodeISR to run.

2) LynxHALNodeISR sets a flag to indicate a packet is in the GRF.

3) When LynxHALCheckInterruptFlags runs, it sees the packet in the GRF flag set.

4) LynxHALCheckInterruptFlags calls LynxHALIndicationCallback.

5) LynxHALIndicationCallback calls the API's Bus Event Indication Callback function.

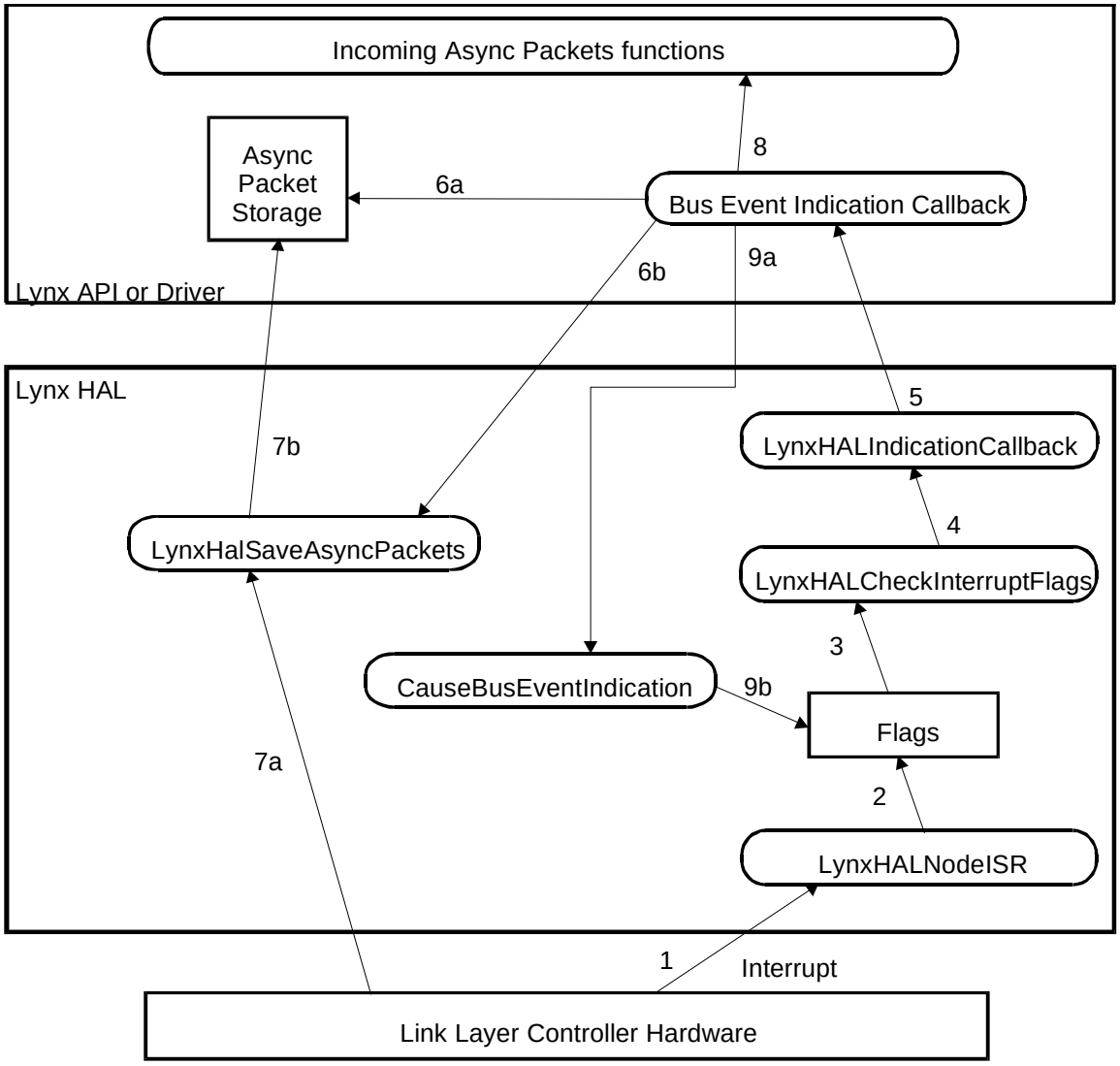
6) The API's Bus Event Callback function creates storage space for the packet and then calls LynxHALSaveAsyncPackets.

7) LynxHALSaveAsyncPackets will move the packet from the GRF to the API's storage space.

8) The API's Callback function will then call the API Incoming Async Packet functions to process the incoming packet.

9) If the GRF is not empty, API's Bus Event Indication Callback function will call CauseBusEventIndication to set the flag that will cause the process to start again.

Figure 4. Incoming Async Packet



Porting

The two major areas to consider when porting the HAL to a different system is how to read or write GPLynx Link Layer Controller Registers and how interrupts are handled.

Reading and Writing GPLynx Registers

All of the register functions and macros are contained in the files LynxReg.c and LynxReg.h.

Some porting considerations:

- The start physical address of GPLynx Configuration Registers contained in the defined StartAddressGPLynxCFR will be unique to each system.
- GPLynx reads a 32-bit register in either 8 or 16 bit increments. Keep in mind that an interrupt can occur at any time while trying to read the complete 32 bits of a register. Because of that, it is recommended to turn off interrupts at the start of a 32-bit read.
- GPLynx is a Big Endian device, meaning the Most Significant Byte is at offset 0. When a register is read from offset xx0, the MSB part will come out first followed by the LSB.

Handling Interrupts

All interrupt-related functions are contained in the files LynxISR.c and LynxISR.h.

Some porting considerations:

- How does the system connect the LynxHALNodeISR to the hardware interrupt vector? In the case of the GPLynx EVM using a TMS320C52, the address of the ISR was placed into the interrupt vector map.
- It may be possible to miss an interrupt that occurs while in the ISR. Note that LynxHALNodeISR checks the Interrupt Register for set bits before it exits and does not exit until there are no bits set.

The interrupt functions will be greatly affected if an operating system is used. The operating system will have a preferred way to handle interrupts, such as setting semaphores, queuing function calls, or starting tasks.