LEX ULM Design

GoldenEars Software

Authors:	Michelle La Haye
Project:	New Project
Date Saved:	11/25/2009 03:20:41 PM
Reviewed by:	
Approved by:	

Contents

1 Introduction

This document presents the LEX USB Link Module (ULM) software design for GoldenEars. This document also applies to the Virtual Hub demo design implemented on modified Lionsgate hardware.

1.1 Purpose

The purpose of this document is to identify the goals of the LEX ULM software and describe how it will achieve those goals. A detailed description of the design will show how it will function, how it will interact with other software components, and how it will interface to the hardware.

1.2 Scope

This document covers the LEX ULM software but not the REX nor the link component between any LEX and REX. It also does not state the details of how to control the underlying LEX hardware, see the ULM BDS document 90-00396.

1.3 Terminology References

ULM – USB Link Module

1.4 Document References

[R-1]	*90-00396 ULM BDS*, Section 6.3, Terry Sosniak, Feb, 2009

2 Design Overview

2.1 Background Information

The LEX hardware provides the interface between the host computer and any connected devices, be that a virtual function or a REX. The LEX ULM hardware notifies the software of events through interrupts. And in order for the LEX hardware to communicate to the host computer the software needs to setup the hardware with the required information and enable it at the right time.

2.2 User Characteristics

2.2.1 User Problem Statement

The LEX needs to know when to connect to the host computer, when to transmit USB traffic, and when to issue resets and other control requests to the virtual function or REX.

2.2.2 User Objectives

The LEX FSM software component is going to achieve the following listed goals:

1. Maintain connection status
2. Issue reset and other control commands to connected devices (virtual function or REXs)
3. Issue reset and other commands to the topology tree
4. Inform LEX hardware of it's connection speed and other required data
5. Enable/disable static queue event interrupts at appropriate times

2.3 Proposed Process

An event handler is going to process the interrupts received from the LEX ULM and take appropriate actions based on which interrupt is received. In addition, messages from other software components or REX will be processed by the event handler and the required actions taken immediately after decoding the message.

2.4 Constraints

The LEX ULM software is responsible for the connection status to the host computer but not of the connection status of any of the downstream devices through any connected REXs.

2.5 Design Trade-offs

This code is written for simplicity and readability.

3 Design Architecture

There are two external modules in this design, one dealing with the hardware interrupts from the local side and the other dealing with messages received from the remote side.

3.1 ULM General ISR

The ULM general ISR is triggered when there is a local ULM event. It then reads the ULM interrupt register and determines which interrupt occurred and what actions to take. Below is the list of the interrupts handled by the general ISR.

- Connect
- Disconnect
- Suspend
- Host resume
- Host resume done
- Remote resume done
- Bus reset
- Negotiate done
- Bus reset done
-

3.2 ULM Message Handler

The ULM message handler is called when a message from either the remote side is received over the link or from the virtual function on LEX that is destined for the ULM. It decodes the message and takes the required actions. The following list covers the ULM messages received by LEX.

- Root device connect
- Root device disconnect
- Remote wakeup

4 External Module interface

4.1 Use model

The LEX ULM software component is intended to be used through event notifications. When an event, either a local interrupt or a message from a remote REX or the virtual function listed in section 3, occurs the ULM event handler will be informed by directly calling the appropriate API to handle an interrupt or a message.

4.2 API/Data Structures

The following API are exposed to the system.

```
void LEX_Init(void)
void LEX_UlmGeneralIsr(void)
void LEX_UlmMessageHandler
(
    UsbMessageT eventMsg // current event message received lex needs to handle
)
```

If a virtual function is configured, LEX_Init() is called on start up by the virtual function initialization code to set the local function pointer for message passing. If a virtual function exists, messages for the root device are passed to it, otherwise they are sent to the remote REX over the link.

5 Detailed Design

The LEX ULM software component is composed of two main modules: the general ISR and message handler. The LEX general ISR handles all ULM interrupts and the message handler deals with messages received from REX over the link.

5.1 LEX ULM General ISR

5.1.1 Processing

On an interrupt event (connect, disconnect, bus reset, negotiate done, bus reset done, suspend, resume, resume done, and remote resume done) the LEX ULM general ISR is called. The LEX ULM general ISR decodes the interrupt and The actions the LEX ULM takes are based strictly on the current event. The table below lists the interrupts generated by the ULM and the actions that are taken.

Table 1: LEX Interrupt Event Handling

Interrupt	**Action**
Host connect	Enable LEX Control and LEX Control Response queue interrupts Turn on host LED
Host disconnect	Disable LEX Control and LEX Control Response queue interrupts Disconnect LEX from host by disconnecting the LEX USB port Turn off host LED
Bus reset	Reset all devices in topology Setup root device for connection Clear data path
Negotiate done	Set speed register settings in ULM (inter packet gap) and CLM (back to back timeout) Store root device speed in topology Send bus reset message to root device Enable forwarding of USB traffic to root device
Bust reset done	Flush LEX Control and Control Response queues Enable control endpoint (0,0)
Host suspend	Send suspend message to root device Flash host LED
Host resume	Send resume message to root device Enable forwarding of USB traffic to root device
Host resume done	Send resume done message to root device Turn on host LED
Remote wake up done	Send remote wake up done message to root device Turn on host LED

5.1.2 Local data structures

None

5.1.3 Inputs

None

5.1.4 Outputs

None

5.2 LEX ULM Message Handler

5.2.1 Processing

The LEX ULM message handler operates in nearly the same way as the general ISR. A message is received and specific actions are taken. Below is a list of the messages and the required actions.

Table 2: LEX Message Event Handling

Message	**Action**
Root device connect	Allow LEX to connect to the host by connecting the LEX USB port
Root device disconnect	Disconnect LEX from the host by disconnecting the LEX USB port
Remote wakeup	Set the remote wakeup bit in the ULM and enable USB traffic

5.2.2 Local data structures

None

5.2.3 Inputs

None

5.2.4 Outputs

None

6 Test Plan

6.1 Unit Tests

The entire component can be tested at once. In order to test the LEX ULM design, a simulated test environment can be constructed that will go through every interrupt and message event signal ensuring that the appropriate actions are taken.

The entire component can be tested together; there is no sense in testing the exit and entry modules separately. In order to test the LEX FSM, s simulated test environment will be constructed that will initialize the state machine to it's starting point and then it will go through every event signal while the FSM is in each of its four states. The test will check that the state machine ends up in the correct state and sends the correct messages out to the other components.

6.2 Limitations

The test environment will not test timing, such as when hardware events occur quickly and what consequences this may incur. This type of testing will be taken care of when tested in the entire system.

6.3 Integration Plan

As the LEX ULM for GoldenEars software will be a replacement for the LEX FSM in Lionsgate software, integration into the system should be mostly a drop-in exercise and should not require much integration. And testing should be straightforward because it can be tested first with the original Lionsgate software to ensure it functions as before, then it will be used in the Virtual Hub software such that by the time it is used in GoldenEars all integration should be complete.