

*Make sure to update all fields on Title page from File->Properties Menu.*

*Once updated right click and select update field to force update*

*\*Note: This text will not print. Do not delete.*

|  |  |
| --- | --- |
|  |  |
| **Authors:** |  |
| **Project:** |  |
| **Date Saved:** |  |

|  |  |
| --- | --- |
| **Reviewed by:** |  |
| **Approved by:** |  |

CONFIDENTIAL

The information contained herein is confidential and shall not be revealed to any third party without the expressed written consent of Icron Technologies Corporation. The document and all copies shall be returned to Icron promptly upon request by Icron.

Contents

1 Introduction 1

1.1 Purpose 1

1.2 Scope 1

1.3 Terminology References 1

1.4 Document References 1

2 Design Overview 2

2.1 Background Information 2

2.2 System Evolution Description 2

2.3 Current Process 2

2.4 Design Objectives 2

2.4.1 Primary 2

2.4.2 Secondary 2

2.5 Proposed Process 3

2.6 Constraints 3

2.7 Design Trade-offs 3

3 Design Architecture 4

3.1 Golden Ears Software Block Diagram 5

3.2 Golden Ears Software Initialization Sequence 6

3.3 Flash Writer Block Diagram 7

3.4 Golden Ears ROM Block Diagram 7

3.5 Icron file format 8

3.6 Jump Table 8

4 Detailed Design 9

4.1 Project Component 9

4.2 Development Environment 10

4.2.1 tigger 10

4.2.2 icmd 10

4.2.3 ibase 11

4.2.4 ibuild 11

4.3 Hardware 12

4.3.1 spectareg 12

4.3.2 leon 12

4.4 Logging 12

4.4.1 ilog 13

4.5 Drivers 13

4.5.1 GRG 13

4.5.2 CLM 14

4.5.3 ULM 15

4.5.4 XCSR 16

4.5.5 XLRC 19

4.6 Utilties 20

4.6.1 Rex Scheduler 20

4.6.2 System Monitor 20

4.6.3 PLL Xilinx/ChipX 21

4.6.4 Xmodem 21

4.7 High Level Components 22

4.7.1 Device Topology Tree 22

4.7.2 VFunction/VHub 23

4.7.3 LexULM 23

4.7.4 RexULM 25

4.7.5 DevMgr 26

4.7.6 DescParser 26

4.7.7 LinkMgr 27

4.7.8 SysQ 28

4.8 FlashWriter 29

4.9 ROM ( BootLoader) 29

5 Development Timeline 31

5.1 Phase 1 - GE code structure on LG1 chip 31

5.1.1 Phase 1A - driver updates 31

5.1.2 Phase 1B - low level code updates 31

5.1.3 Phase 1C - high level code updates 31

5.2 Phase 2 - SW on limited GE FPGA (single link CLEI, no DisplayPort Aux, no MAC address) 31

5.2.1 Phase 2A - Establish link 31

5.2.2 Phase 2B - Full USB enumeration 31

5.2.3 Phase 2C - Full USB functionality 32

5.3 Phase 3 - Full functionality of SW on GE 32

# Introduction

The Golden Ears Software High Level design and the migration path from the Lionsgate1 Software are described in this document. The Golden Ears Software design attempts to implement a more organized structure within the Lionsgate1 Software to allow a more encapsulated design and easier migration between the platforms.

## Purpose

The software goal is to have an easy migration path to the Golden Ears chip, with a solid software foundation.

## Scope

This document covers the high level design of the Golden Ears Software, and its migration path from the Lionsgate1 Software. Detailed designs of the individual components can be found in each component’s design documentation, however depending on the component there are various levels of documentation. For example the leon drivers include a full detailed design, but simple components such as ibase only provide a quick user’s guide in a readme.txt file.

## Terminology References

XUSB – eXtreme USB. Icron’s term for extending USB

Lex – Local USB extender

Rex – Remote USB extender

GRG – Global ReGisters

CLM – Communication Link Module

ULM – USB Link Module

XCSR –XUSB Control Status Registers

XSST – XUSB System Status Table

XICS - XUSB Internal Cache System

XLRC – XUSB Lex Rex Control

## Document References

# Design Overview

## Background Information

The Golden Ears project and previous Lionsgate1 project are USB extension ASIC’s that have an embedded processor. While all of the data path functionality of these chips is in the RTL, the management and configuration of the RTL blocks must be done in software. The software is required to initialize hardware, establish links, negotiate USB speed, handle suspend/resumes, schedule downstream USB packets, configure downstream USB devices, run garbage collection on the RTL queue usage, etc.

## System Evolution Description

The Golden Ears Software is intended to be implemented piece by piece in place of the Lionsgate1 Software. By providing a stable lower level API, higher level code is intended to run on both platforms mostly unmodified. The Golden Ears Software components will be developed for future projects as well.

## Current Process

The current Lionsgate1 Software was implemented in an as needed fashion, without much upfront design, where the design was driven by transactions seen on the USB analyzer instead of designing to the USB specification. As each new USB issue was discovered the current software at that time would be modified to address the current issue. This lead to a large number of cases where low level components would call back into high level components, where the high level components would just call right back into the low level components. This created a small problem with numerous function calls for large stack usage, and a larger issue with very entangled dependencies that make it very difficult to address any issue in isolation.

## Design Objectives

### Primary

1. Initialize all of the hardware in the chip.
2. Create a connection between the Lex and Rex & maintain the link.
3. Create a USB connection between Lex and Rex, by forwarding speed negotiating, bus resets, suspends, and resumes.
4. Configure the hardware to let it know which devices are connected and what their configuration is.
5. Schedule the USB packets to be sent out the Rex USB link.
6. Run garbage collection on the USB transactions.

### Secondary

1. Provide an upgrade path for future software builds.
2. Provide an easy debug path while developing software.
3. Provide a development environment for quick development.
4. Provide a debug environment for the RTL developers.

## Proposed Process

The Golden Ears Software design is broken up into components with a well defined interface between them to allow easy encapsulation and identify all the relationships between components in the earliest stages of design.

## Constraints

## Design Trade-offs

Code correctness is far more important than anything else. To achieve code correctness this design has tried to encapsulate all the various components as much as possible to allow changes to be designed and verified in a controlled environment.

# Design Architecture

The Golden Ears Software structure is shown in the block diagram in Figure 1. Every component on the diagram only depends on components below it, which ensures no circular dependencies, and also allows us to more easily measure an upper bound on stack usage and latency as well as a quick visual estimate of window usage. The diagram does not include links to drivers or lower level utilities to maintain visual simplicity. The components listed are all used for the following design purposes.

Primary Purpose:

1. Initialize Hardware. This involves configuring interrupts, creating static Q’s, initializing data structures:

Components: All

1. Create a connection between the Lex and Rex & maintain the link.

Components: LinkMgr, PLL, CLM, GRG, XCSR, Spectareg, Leon

1. Create a USB connection between Lex and Rex, by forwarding speed negotiating, bus resets, suspends, and resumes.

Components: RexULM, LexULM, ULM, XCSR, XLRC, Spectareg, Leon

1. Configure the hardware to let it know which devices are connected and what their configuration is.

Components: SysQ, DevMgr, DescParser, Topology, XCSR, Spectareg, Leon

1. Schedule the USB packets to be sent out the Rex USB link.

Components: RexScheduler, Spectareg, Leon

1. Run garbage collection on the USB transactions.

Components: SystemMonitor, XCSR, Spectareg, Leon

Secondary purposes:

1. Provide an upgrade path for future software builds

Components: Xmodem, flashwriter

1. Provide an easy debug path while developing software

Components: tigger, ilog, icmd

1. Provide a development environment for quick development

Components: ibase, ibuild, tigger

External tools: git, gcc, binutils, make, tar, bzip2

1. Provide a debug environment for the RTL developers.

Components: tigger, ilog, icmd, ulm, xcsr, xlrc, clm, grg

## Golden Ears Software Block Diagram



Figure - Block Diagram

\* icmd is useable by all components, but at runtime it calls ilog

\*\* The PLL component may be replaced by different versions depending on the target

\*\*\* The VFunction/VHub is left as a placeholder as the functionality is not required in Golden Ears

## Golden Ears Software Initialization Sequence

The Golden Ears Software components are initialized in the following order

* Jump Table - Depends on boot sequencing. Does the i2c load preset this up? Or is it setup from flash?
* Leon – Uart, Timers
* GRG – Verify chip versions, GPIOs
* XCSR – Verify chip versions, initialize the static queues, start stat collection ISR, etc
* ULM – Verify chip revisions
* XLRC– Verify chip revisions
* CLM – Verify chip revisions, configure link, start stat collection
* PLL – setup PLL registers
* Lex Only: System Monitor – Start the system monitor timers
* Rex Only: Rex Scheduler – Initialize structures
* Lex Only: Topology – Initialize structures
* Rex Only: RexULM – Register and start ULM interrupt handlers
* Lex Only: LexULM – Register and start ULM interrupt handlers
* Lex Only: DevMgr – Initialize structures
* Lex Only: DescParser – Initialize structures
* Lex Only: SysQ – Register and start interrupt handler
* LinkMgr – Startup the link, phy configuration, start sending pings
* Project – enable CPU2CPU message interrupt and jump to the loop forever functions for Rex and Lex

## Flash Writer Block Diagram



Figure 2 – Flash Writer Block Diagram

## Golden Ears ROM Block Diagram



Figure 3 – ROM Block Diagram

## Icron file format

All firmware images are in a \*.icron file format. Icron files are .tar.bz2 files that contain at least a file called icron\_header. The icron\_header lists all the other files, the project the .icron file is for, and the version of the .icron file. The version numbers are defined per project.

Here is an example icron\_header file

version:3

project:lg1\_vhub

icomponent:icomponent

ilog:ilog

icmd:icmd

flash\_writer:flash\_writer.icr

flash\_writer2:flash\_writer.bin

main\_firmware:AsicFirmware.bin

This icron header lets Tigger know the project and version, as well as the various other files within the .icron file for updating the firmware. The 1st flash writer is a stewie compatible with the Lionsgate1 Boot ROM, and the 2nd flash writer is a binary compatible with raw file transfers such as X-Modem. This .icron file also contains the main firmware, ilog & icmd files. Each ilog and icmd file is bundled with the .icron file so Tigger knows which commands are available with every build, and how to interpret each logging message from different builds.

## Jump Table

Brief

This is a layer in the code that separates blocks at the link layer. This allows the system to be later patched without having to build a complete new image.

Details

Yet to be defined. This is at a pre-planning stage

Current Status (April 8, 2010)

No work has been done

Work Estimate:

3 weeks (bit of a guesstimate)

# Detailed Design

The following documents all of the individual components with an overview as well as the interfaces between components. Note that since just about every component uses the lower level components such as ilog, icmd, ibase, and leon, they are omitted from several interfaces to keep the document to a reasonable size.

## Project Component

Brief

This component defines the non-reusable code for this project, and depends on all of the other components.

Defines

* gpios.h
* interrupts.h
* project\_components.h
* lexrex\_msgs.h – GE Sample shown. To be updated when this component goes through a detailed design.

Lionsgate interim stage

To maintain compatibility with current Lionsgate software, the interfaces for pings and software version messages must remain the same. lexrex\_msgs.h will contain the replacement for CpuOpcodeT with the same enum values. The Lex and Rex ULM messages and the Rex upstream response message can be modified.

Project needs C file to provide replacement for CLM\_ProcessRxMessage, this will be moved into the LinkMgr for Goldenears.

GoldenEars sample

2 bytes in the header for a message

struct lexrex\_msg { uint16:4 msg\_id; uint16:12 msg\_data; }; // exposed without bitfields, msg\_id is an enum exposed at top level, everything else in XUSB

defines: MAJOR\_VERION // 4 bits

defines: MINOR\_VERION // 4 bits

defines: TINY\_VERION // 4 bits //TODO: this required for compatibility?

Exposes

* define: PROJECT\_VERSION ((MAJOR\_VERSION) << 8) | etc. – Sample shown. To be updated when this component goes through a detailed design

Calls

all initialization functions

configures jump table

void GRG\_GpioInit();

void ULM\_Init();

void CLM\_Init(linkType, enableRetries, fullDuplex);

void LINKMGR\_Init(enum linkT); //which calls PLL init functions

void XCSR\_Init();

void XLRC\_Init();

void SYSMON\_Init();

void LINKMGR\_IncomingPing(uint8 rexId);

Arg not needed if there is no virtual hub, 0 could mean Lex

void LINKMGR\_IncomingPong(uint8 rexId);

Arg not needed if there is no virtual hub, 0 could mean Lex

void REXULM\_IncomingMsg(enum);

void LEXULM\_IncomingMsg(enum, uint8 rexId);

Arg not needed if there is no virtual hub

Current Status (April 8, 2010)

Only minimal initialization is done as it is mostly buried in the old LG1 code. Also the messages between Rex and Lex are also still buried in the LG1 code.

Work Estimate:

2 weeks

## Development Environment

The following components make up the development environment used by the Golden Ears Software.

### tigger

Brief

Tigger is a development/debug tool that connects to the target via a UART cable. Tigger is able to decode logging messages from the target, encode messages to the target, and send the target firmware images.

### icmd

Brief

Icmd provides a binary interface for sending debugging commands to the target. The icmd macros can be called by all components at build time. At run time icmd is set by the project to be called on received UART characters and that is its only exposed runtime interface.

Exposes

void ICMD\_ProcessByte(uint8 rxByte);

ICMD\_FUNCTIONS\_CREATE(\_component\_code\_)

ICMD\_FUNCTIONS\_ENTRY(\_function\_, \_help\_string\_, \_args\_...)

ICMD\_FUNCTIONS\_END(\_component\_code\_)

Calls

Ilog

leon timers

Current Status (April 8, 2010)

Complete

Work Estimate:

Nil

### ibase

Brief

An Icron base common set of low level C definitions. It defines memcpy/memset, types, components, fifos, structure macros, and casting macros. This is the lowest level C library.

External

The project should expose project\_components.h to use component functionality.

Exposes

Fifo API's

memset/memcpy

(u/s)int(8/16/32)

boolT

TRUE/FALSE/NULL

component\_t

build time component parser engine

CAST(var, old\_type, new\_type)

structure macros

etc.

Calls

Nothing

Current Status (April 8, 2010)

Complete

Work Estimate:

Nil

### ibuild

Brief

Makefile rules to build Icron software projects.

Exposes

A set of variables to be set by makefiles, which then include ibuild to build the project.

Current Status (April 8, 2010)

Complete, except

1) Doesn't automatically build unit tests. These are currently built manually.

2) Building project files should be more automated. This would reduce some manual work on each test harness

Work Estimate:

1 week (ultra-low priority)

## Hardware

The following components describe the lowest level hardware interfaces in the Golden Ears Software Design.

### spectareg

Brief

A set of auto-generated macros for accessing RTL registers.

Exposes

A whole set of macros for all drivers GRG, XCSR, XLRC, ULM, CLM & the Rex Scheduler.

Current Status (April 8, 2010)

Auto-generated and updated as the RTL team updates their code.

Work Estimate:

No SW work needed

### leon

Brief

Drivers for the Leon SPARC processor system which contain CPU, timers, uart, and flash.

External

The project exposes interrupts.h to allow all components to know the interrupt mapping.

Exposes

Trap/Interrupt/Critical Section API

UART API for both polling & interrupt modes

Timer API with both time stamp reading and callback functions

Flash transport API. The user needs to know the Serial Flash commands

CPU internals API to help assert handlers

Calls

ibase

Current Status (April 8, 2010)

Complete for LG1 in GE structure. GE is going to have a new trap handling RTL implementation, which SW has yet to code

Work Estimate:

1 week

## Logging

The following component describes the logging capabilities of the Golden Ears Software.

### ilog

Brief

A binary logging mechanism to assist debugging

External

This relies on project\_components.h from the project.

Exposes

void ilog\_<COMPONENTNAME>\_0(ilog\_level\_t, code\_t);

void ilog\_<COMPONENTNAME>\_1(ilog\_level\_t, code\_t, uint32 arg1);

void ilog\_<COMPONENTNAME>\_2(ilog\_level\_t, code\_t, uint32 arg1, uint32 arg2);

void ilog\_<COMPONENTNAME>\_3(ilog\_level\_t, code\_t, uint32 arg1, uint32 arg2, uint32 arg3);

void iassert\_<COMPONENTNAME>\_0(expr, code\_t);

void iassert\_<COMPONENTNAME>\_1(expr, code\_t, uint32 arg1);

void iassert\_<COMPONENTNAME>\_2(expr, code\_t, uint32 arg1, uint32 arg2);

void iassert\_<COMPONENTNAME>\_3(expr, code\_t, uint32 arg1, uint32 arg2, uint32 arg3);

Calls

Leon uarts

Leon locks

Leon CPU assert helper API

Current Status (April 8, 2010)

Complete

Work Estimate:

Nil

## Drivers

The following component describes the Icron drivers of the Golden Ears Chip. Note the Icron drivers do not include the Leon component.

### GRG

Brief

Contains the configuration, GPIO, PLL , MDIO, and module reset registers. As the RTL design solidifies this may also include an i2c master/slave interface and a DisplayPort configuration and softpath enable.

External

Requires the project to expose gpio.h, which defines enum gpio.

Exposes

void GRG\_Reset(enum moduleResetT);

void GRG\_ClearReset(enum moduleResetT);

void GRG\_Init(uint8 \* pMajor, uint8 \* pMinor, uint8 \* pDebug); // init + returns chip revision

enum linkType GRG\_GetLinkType(void);

boolT GRG\_IsStreamingMode(void);

boolT GRG\_IsHSJumperSelected(void);

boolT GRG\_IsDeviceRex(void);

boolT GRG\_IsDeviceLex(void);

boolT GRG\_IsDeviceAsic(void);

boolT GRG\_IsDeviceFpga(void);

struct gpioInitStates { enum gpioT; enum { INPUT, OUTPUT\_CLEAR, OUTPUT\_SET }; };

void GRG\_GpioInit(struct gpioInitStates \*, uint8 numOfGpioInitStates);

boolT GRG\_GpioRead(enum gpioT);

void GRG\_GpioSet(enum gpioT);

void GRG\_GpioClear(enum gpioT);

void GRG\_GpioPulse(enum gpioT); //For LEDs. Pulse time should be defined in options.h

void GRG\_MdioWrite(uint8 device, uint8 address, uint16 data);

uint16 GRG\_MdioRead(uint8 device, uint8 address);

void GRG\_CLMPLLWrite(uint32);

void GRG\_CRMPLLWrite(uint32);

void GRG\_CTMPLLWrite(uint32);

uint32 GRG\_CLMPLLRead(void);

uint32 GRG\_CRMPLLRead(void);

uint32 GRG\_CTMPLLRead(void);

Calls

Spectareg macros

Leon timers

ilog

Current Status (May 21, 2010)

The API and code implementation is updated from the LG1 code. No work has yet being done for the minor change to update to the GE registers, although this is currently expected to be very minor. The code also needs a code review before closing it off.

If i2c support or DisplayPort Aux support is added, then this is additional work not currently estimated.

Work Estimate:

3 days, in its current form, 1 day for GE changes, 1 day for test harnesses, and 1 day for code review. More for i2c and DisplayPort configuration if added.

Test Harnesses

Blink the GPIO LEDs – ensure input LEDs can't be blinked

Read & write an MDIO phy – this is done through icmd & may not necessarily need a separate test

Read configuration – This only needs implementation if issues are discovered, since we work with all configurations currently

higher level code must test PLLs & resets/enables

### CLM

Brief

Contains the link configuration options.

Exposes

// Verifies chip Id and Rev, and configures interrupts and registers

void CLM\_Init(enum linkType ltype, boolT fullDuplex);

// called by LexULM/RexULM

void CLM\_SetUSBHighSpeed(void);

void CLM\_SetUSBLowFullSpeed(void);

Other

Stats collection:

started by initialization

controlled by icmd

uses leon timers & ilog

Calls

Spectareg macros

Leon timers, and interrupts

ilog

Current Status (May 27, 2010)

Only missing stats for LG1, Not updated for GE

Work Estimate:

2 weeks

### ULM

Brief

Provides the interrupts, and connection registers for the USB link to host or device.

Exposes

void ULM\_Init(boolT isDeviceLex);

ULM\_InterruptBitMaskT ULM\_GetAndClearInterrupts(void);

boolT ULM\_CheckInterruptBit(ULM\_InterruptBitMaskT, enum ULM\_InterruptBit);

void ULM\_ConnectRexUsbPort(UsbSpeedT usbSpeed);

void ULM\_GenerateRexUsbReset(UsbSpeedT usbSpeed);

void ULM\_ConnectLexUsbPort(UsbSpeedT usbSpeed);

void ULM\_RexExtendResetBegin(void);

void ULM\_RexExtendResetRestore(void);

void ULM\_GenerateRexUsbSuspend(void);

void ULM\_EnableDisconnectInterrupt(void);

void ULM\_EnableConnectInterrupt(void);

void ULM\_SetUSBHighSpeed(void);

void ULM\_SetUSBFullSpeed(void);

void ULM\_SetUSBLowSpeed(void);

UsbSpeedT ULM\_ReadDetectedSpeed(void);

// USB Tests

void ULM\_SetTestMode(boolT hostTest);

void ULM\_GenJ(void);

void ULM\_GenK(void);

// Disconnect the USB port on the Lex

void ULM\_DisconnectLexUsbPort(void);

// Disconnect the USB port on the Rex

void ULM\_DisconnectRexUsbPort(void);

// Generate a USB RESUME

void ULM\_GenerateRexUsbResume(void);

// Generate a Rex extend resume

void ULM\_GenerateRexUsbExtResume(void);

// Clear the Extend Reset bit

void ULM\_RexExtendResetClear(void);

// Clear Rex extend resume

void ULM\_ClrRexUsbExtResume(void);

void ULM\_GenerateLexUsbRemoteWakeup(void);

Calls

Spectareg macros

Leon timers

ilog

Current Status (May 27, 2010)

Updated for LG1, needs updates for GE

Work Estimate:

1 weeks

### XCSR

Brief

Consists of several submodules: XUSB, XSST (system status table), XICS (cache), XCRM, and XCTM.

External

The project must expose lexrex\_msgs.h, which provides enum lexRexMsgT.

The project must expose queues.h, which provides queueT (probably an enum).

Exposes

**XUSB**

eXCSR\_StaticQidT

void XCSR\_Init(void);

//verfies chip Id and Rev. Clears the hardware, allocates static queues, sets flow control rules,

//setsup BCO settings, enables xctm and xcrm modules, etc

void XCSR\_XUSBAllocateStaticQueues(void);

void XCSR\_XUSBHostTestmodeEnable(void);

void XCSR\_XUSBXctmEnable(void);

void XCSR\_XUSBXctmDisable(void);

void XCSR\_XUSBXcrmEnable(void);

void XCSR\_XUSBXcrmDisable(void);

void XCSR\_XUSBEnableSystemQueueInterrupts(void);

void XCSR\_XUSBDisableSystemQueueInterrupts(void);

void XCSR\_XUSBClearDataPath(void); // may not be needed in GoldenEars

void XCSR\_XUSBEnableInterruptLexFlushQueue(void);

void XCSR\_XUSBDisableInterruptLexFlushQueue(void);

void XCSR\_XUSBClearInterruptLexFlushQueue(void);

void XCSR\_XUSBEnableInterruptLexCtrl(void);

void XCSR\_XUSBDisableInterruptLexCtrl(void);

uint8 XCSR\_XUSBReadInterruptLexCtrl(void);

void XCSR\_XUSBClearInterruptLexCtrl(void);

void XCSR\_XUSBDisableInterruptRexSof(void);

void XCSR\_XUSBEnableInterruptLexCtrlResp(void);

void XCSR\_XUSBDisableInterruptLexCtrlResp(void);

uint8 XCSR\_XUSBReadInterruptLexCtrlResp(void);

void XCSR\_XUSBClearInterruptLexCtrlResp(void);

void XCSR\_XUSBEnableInterruptCpuRx(void);

void XCSR\_XUSBDisableInterruptCpuRx(void);

uint8 XCSR\_XUSBReadInterruptCpuRx(void);

void XCSR\_XUSBClearInterruptCpuRx(void);

uint32 XCSR\_XUSBReadInterrupts(void);

uint8 XCSR\_XUSBReadIrq2CpuRx(void);

**XSST**

struct XCSR\_XusbAddress

XUSB address type. Some sort of combination of logical and physical packed together. Includes an insys bit.

void XCSR\_XSSTInit(void);

void XCSR\_XSSTAddress0Enable(void);

void XCSR\_XSSTAddress0Disable(void);

void XCSR\_XSSTUpdateAddress(struct XCSR\_XusbAddress \*pAddress);

//sets the logical address in the logical address look up table

void XCSR\_XSSTClearInsys(struct XCSR\_XusbAddress \*pAddress);

void XCSR\_XSSTClearAddress(struct XCSR\_XusbAddress \*pAddress);

void XCSR\_XSSTReadLogicalAddressTable(struct XCSR\_XusbAddress \*pAddress);

void XCSR\_XSSTRead(struct XCSR\_XusbAddress \*pAddress, uint8 endPoint, uint32 \*pStatus);

//all writes to the XSST will do a read-modify-write when needed

void XCSR\_XSSTWriteEndpoint(struct XCSR\_XusbAddress \*pAddress, uint8 endpointNumber, uint8 endpointType, uint8 endpointExtension, uint32\* oldStatus);

void XCSR\_XSSTMoveEndpoints(struct XCSR\_XusbAddress \*pAddress, uint8 maxEndpoint);

void XCSR\_XSSTResetEndpoint(struct XCSR\_XusbAddress \*pAddress, uint8 endpointNumber);

//clear everything except the endpoints for the given address

void XCSR\_XSSTClearEndpoints(struct XCSR\_XusbAddress \*pAddress, uint8 maxEndpoint);

void XCSR\_XSSTSetEndpointExtension(struct XCSR\_XusbAddress \*pAddress, uint8 endpointNumber);

void XCSR\_XSSTClearBCO(struct XCSR\_XusbAddress \*pAddress);

void XCSR\_XSSTClearBCI(struct XCSR\_XusbAddress \*pAddress);

void XCSR\_XSSTClearOutBlocking(struct XCSR\_XusbAddress \*pAddress);

//all of the following are set by software, cleared by hardware

void XCSR\_XSSTSetInClear(struct XCSR\_XusbAddress \*pAddress, uint8 endpoint);

void XCSR\_XSSTSetOutClear(struct XCSR\_XusbAddress \*pAddress, uint8 endpoint);

void XCSR\_XSSTSetInDetect(struct XCSR\_XusbAddress \*pAddress, uint8 endpoint);

void XCSR\_XSSTSetOutDetect(struct XCSR\_XusbAddress \*pAddress, uint8 endpoint);

void XCSR\_XSSTWriteAlternateResponse(struct XCSR\_XusbAddress \*pAddress, uint8 endpoint, uint8 alternateResponse);

void XCSR\_XSSTSendNakResponse(struct XCSR\_XusbAddress \*pAddress, uint8 endpoint);

void XCSR\_XSSTSendErrResponse(struct XCSR\_XusbAddress \*pAddress);

void XCSR\_XSSTSetupTestmode(struct XCSR\_XusbAddress \*pAddress);

void XCSR\_XSSTSetupTestmodeStall(struct XCSR\_XusbAddress \*pAddress);

//will be more writes/clears to the XSST as the bits are finalized such as ovrLay, rtyUsage

void XCSR\_XSSTWriteAccel(<XUSB address type> addr, uint8 endpointNum, accel);

void XCSR\_XSSTSetSplitDev(<XUSB address type> addr);

void XCSR\_XSSTClearSplitDev(<XUSB address type> addr);

//used for UI commands and bug fixes

void XCSR\_XSSTWrite(struct XCSR\_XusbAddress \*pAddress, uint8 endPoint, uint32 writeValue, uint32 writeMask, struct XCSR\_Xsst \*pOldValue);

**XICS**

eXCSR\_PacketT

eXCSR\_QueueReadStatusT

eXCSR\_QueueStatusT

struct XCSR\_MessageData

struct XCSR\_CacheFrame

struct XCSR\_QueueStats

void XCSR\_XICSQQueueFlushInit(void);

void XCSR\_XICSSetFlowCtrlThresholds(void);

uint8 XCSR\_XICSQQueueAllocate(void);

void XCSR\_XICSQQueueDeallocate(uint8 qid);

void XCSR\_XICSQQueueFlush(uint8 qid);

eXCSR\_QueueStatusT XCSR\_XICSQQueueGetEmptyStatus(uint8 qid);

void XCSR\_XICSQQueueGetStatistics(uint8 qid, struct XCSR\_QueueStats \*pStats);

void XCSR\_XICSSendMessage(struct XCSR\_MessageData \*pData);

eQueueReadStatusT XCSR\_XICSQueueRead(uint8 qid, struct XCSR\_CacheFrame \*);

eQueueWriteStatusT XCSR\_XICSQueueWrite(uint8 qid, struct XCSR\_CacheFrame \*);

//the only write error is if there was an overflow when trying to write to the queue, enum might be overkill

?void XCSR\_XICSClearCache();

//TODO: adds stats calls. Should be from icmd? And through the initialization

**XCRM**

? - See Aaron’s comments

**XCTM**

?

Calls

Spectareg macros

ilog

GRG\_IsFullDuplex();

Current Status (April 8, 2010)

Significantly intertwined with multiple components in the LG1 code base. Needs cleanup for LG1 and for GE needs to match the new RTL changes.

Work Estimate:

7 weeks

### XLRC

Brief

Contains the drivers for the lex and rex components of the Golden Ears hardware

Exposes

void XLRC\_Init(void);

void XLRC\_XlexEnable(void);

void XLRC\_XlexDisable(void);

void XLRC\_XrexEnable(void);

void XLRC\_XrexDisable(void);

void XLRC\_ClearQueueTracker(void);

void XLRC\_XrexSOFEnable(void);

// Used for testmode

void XLRC\_EnableNakAllRequests(void);

void XLRC\_DisableNakAllRequests(void);

void XLRC\_StartSendingTestModePacket(void);

void XLRC\_SetQueueTrakerQid(uint8 qid);

?void XLRC\_AddGenericBco/Bci(uint32 mask, uint32 filter);

//Will not be needed right away, but may be used later for potential bug fixing or unanticipated scenarios

//TODO: adds stats calls. Should be from icmd? And through the initialization

Calls

Spectareg macros

ilog

Leon interrupts

XCSR\_XICSQQueueFlush();

XCSR\_XUSBClearInterruptLexFlushQueue();

Current Status (April 8, 2010)

This split off of XCSR for GE and part of XCSR for LG1. See XCSR for status

Work Estimate:

1 week

## Utilties

The following section contains the low level utilities of the Golden Ears Software.

### Rex Scheduler

Brief

Schedules all USB transactions out the Rex.

Exposes

void REXSCH\_Enable(boolT highSpeed);

void REXSCH\_Init(boolT fastLink);

void REXSCH\_Disable(void);

Calls

leon interrupts

Spectareg macros

ilog

Current Status (May 27, 2010)

GE updates are to be done by the RTL team.

Work Estimate:

Done for the SW team

### System Monitor

Brief

Monitors the XUSB state for various deadlock type scenarios, which it then cleans up.

Exposes

void SYSMON\_Init(void);

void SYSMON\_Check(void);

void SYSMON\_Clear3Strikes(<XUSB address type> addr);

void SYSMON\_Clear3StrikesAll(void);

**Calls**

void XCSR\_XSSTRead(struct XCSR\_XusbAddress \*pAddress, uint8 endPoint, uint32 \*pStatus);

void XCSR\_XICSQQueueFlush(uint8 qid);

void XCSR\_XSSTClearOutBlocking(struct XCSR\_XusbAddress \*pAddress);

void XCSR\_XSSTWriteAlternateResponse(struct XCSR\_XusbAddress \*pAddress, uint8 endpoint, uint8 alternateResponse);

void XCSR\_XSSTSetInClear(struct XCSR\_XusbAddress \*pAddress, uint8 endpoint);

void XCSR\_XSSTSetOutClear(struct XCSR\_XusbAddress \*pAddress, uint8 endpoint);

void XCSR\_XSSTSetInDetect(struct XCSR\_XusbAddress \*pAddress, uint8 endpoint);

void XCSR\_XSSTSetOutDetect(struct XCSR\_XusbAddress \*pAddress, uint8 endpoint);

//will be more writes/clears to the XSST as the bits are finalized such as ovrLay, rtyUsage

void XCSR\_XSSTSendNakResponse(struct XCSR\_XusbAddress \*pAddress, uint8 endpoint);

Current Status (April 8, 2010)

XSST is significantly changing for GE however the high level functionality hasn't changed. Most of the work is just modifying the lower level code to the new HW.

Work Estimate:

1 week

### PLL Xilinx/ChipX

Brief

Has knowledge of the chip or FPGA's PLL register internals

Exposes

void PLL\_CHIPXInit(enum linkT); //For ChipX version //called by project initialization

void PLL\_XILINXInit(enum linkT); //For Xilinx version //called by project initialization

boolT PLL\_SetupCRM(void); // returns TRUE on success, FALSE otherwise

// caller should re-check link status if PLL could not be acquired.

// Should we blindly do this for all links? Or just when configured for TBI?

Calls

void GRG\_CLMPLLWrite(uint32);

void GRG\_CRMPLLWrite(uint32);

void GRG\_CTMPLLWrite(uint32);

void GRG\_Reset(enum moduleResetT);

void GRG\_ClearReset(enum moduleResetT);

Current Status (April 8, 2010)

Works in the LG1 structure but is not very readable. It needs a complete overhaul. Also needs extracting from link acquisition code.

Work Estimate:

4 weeks (guesstimate, based on the current volume of code)

### Xmodem

Brief

Provides basic XModem support for grabbing images over the uart.

Exposes

// For GE main SW & ROM

boolT XMODEM\_PolledModeReceive(uint32 \* (\*rxDataHandlerFunction)(uint32 \* buf, uint32 bufSize), uint32 \* buffer);

// For Flashwriter

void XMODEM\_InterruptModeInit(void);

boolT XMODEM\_InterruptModeReceive(uint32 \* (\*rxDataHandlerFunction)(uint32 \* buf, uint32 bufSize), uint32 \* buffer1, uint32 \* buffer2);

Calls

Leon timers

Leon uart

Test harnesses

Grab text over the uart and echo it back. Really just helps development. Higher level code is going to find out if it works or not fairly quickly

TODO: add test harnesses to test error conditions: timeouts, checksum errors, etc.

Limitations

As the xmodem code may be running from serial flash, the software may have trouble keeping up with a fast uart speed. We currently support 115200. Increasing the speed could probably only occur with a faster serial flash, or a significantly sized UART buffer.

Current Status (April 8, 2010)

Complete, except for extra test harnesses to test error conditions.

Work Estimate:

Nil, unless we implement all the test harnesses, then a week.

## High Level Components

The following components are the high level components of the Golden Ears Software

### Device Topology Tree

Brief

Keeps track of the current USB devices attached and their status and updates the XSST. Gets its data from the system queue.

Exposes

void DTT\_Init(void);

void DTT\_CreateXusbAddr(???);

uint8 DTT\_GetLAddr(<XUSB address type> addr);

uint8 DTT\_GetUsbAddr(<XUSB address type> addr);

uint8 DTT\_ResetDevice(<XUSB address type> parentAddr, uint8 portNum);

//Searches for the device in the topology, if it exists it returns it's logical address otherwise it adds a new device node in the tree and returns the assigned logical address.

//Passing in parent=0, port=0 indicates this is for the root hub.

void DTT\_UpdateAddress(<XUSB address type> currAddr, <XUSB address type> newAddr);

//Updates the device node at the given logical address with its assigned USB address.

void DTT\_DisableDevice(<XUSB address type> parentAddr, uint8 portNum);

//Marks the device identified by parent, port and all its descendents as disconnected and cleans up their XSST entries.

void DTT\_SetPortStatusPort(<XUSB address type> addr, uint8 portNum);

uint8 DTT\_GetPortStatusPort(<XUSB address type> addr, uint8 portNum);

void DTT\_SetDeviceSpeed(<XUSB address type> addr, eUsbSpeedT speed);

void DTT\_UpdateEndpoint(<XUSB address type> addr, uint8 endpointNum, uint8 attributes,

uint8 config, uint8 interface);

uint8 DTT\_GetHighEndpoint(<XUSB address type> addr);

?void DTT\_ClearHighEndpoint(<XUSB address type> addr);

?boolT DTT\_IsHub(<XUSB address type> addr);//TODO: should this be internal to DTT?

void DTT\_SetHub(<XUSB address type> addr);

void DTT\_ClearHub(<XUSB address type> addr);

?void DTT\_GetDisconnetStatus(<XUSB address type> addr);

void DTT\_HandleSetInterface(??);

void DTT\_HandleSetConfiguration(??);

Calls

void XCSR\_XSSTUpdateAddress(struct XCSR\_XusbAddress \*pAddress);

void XCSR\_XSSTWriteEndpoint(struct XCSR\_XusbAddress \*pAddress, uint8 endpointNumber, uint8 endpointType, uint8 endpointExtension, uint32\* oldStatus);

void XCSR\_XSSTMoveEndpoints(struct XCSR\_XusbAddress \*pAddress, uint8 maxEndpoint);

void XCSR\_XSSTSetEndpointExtension(struct XCSR\_XusbAddress \*pAddress, uint8 endpointNumber);

void XCSR\_XSSTResetEndpoint(struct XCSR\_XusbAddress \*pAddress, uint8 endpointNumber);

void XCSR\_XSSTClearEndpoints(struct XCSR\_XusbAddress \*pAddress, uint8 maxEndpoint);

void XCSR\_XSSTClearInsys(struct XCSR\_XusbAddress \*pAddress);

void XCSR\_XSSTClearAddress(struct XCSR\_XusbAddress \*pAddress);

void XCSR\_XICSQQueueFlush(uint8 qid);

void XCSR\_XICSQQueueDeallocate(uint8 qid);

void XCSR\_XSSTReadLogicalAddressTable(struct XCSR\_XusbAddress \*pAddress);

void XCSR\_XSSTRead(struct XCSR\_XusbAddress \*pAddress, uint8 endPoint, uint32 \*pStatus);

// Not clear who will be responsible for this, topology or device manager

void XCSR\_XSSTAddress0Enable(void);

void XCSR\_XSSTAddress0Disable(void);

void SYSMON\_Clear3Strikes(<XUSB address type> addr);

void SYSMON\_Clear3StrikesAll();

Current Status (April 8, 2010)

The topology currently doesn't control everything related to topology, so there are a number of other blocks that are going to push their code into the topology including the DevMgr and parser. As well the USB address to logical address mapping is currently contained in the XCSR as well as the decision logic to assign a device a logical address. This code will also be moved into the topology. And this is going to mean a lot of code within the topology is also going to be updated.

Work Estimate:

5 weeks (more)

### VFunction/VHub

The Virtual Hub or generic Virtual Function is a placeholder in the software design for a future date when a virtual device is present.

### LexULM

Brief

Handles all of the events received from XUSB hardware and USB events relayed from the Rex. Manages host USB connection.

Exposes

void LEXULM\_GeneralIsr(void);

void LEXULM\_LexRexLinkUp(void);

void LEXULM\_LexRexLinkDown(void);

void LEXULM\_ForceEnumeration(void);

//To take care of quick enumeration.

void LEXULM\_IncomingMsg(enum ?, uint8 rexId);

Calls

uint32 ULM\_ReadInterrupts();

void GRG\_GpioSetLed(eGpioT led);

void GRG\_GpioClearLed(eGpioT led);

void GRG\_GpioPulseLed(eGpioT led);

?void XCSR\_XUSBEnableStaticQEvents(); // not sure if these are needed

?void XCSR\_XUSBDisableStaticQEvents(); // not sure if needed

uint8 DTT\_ResetDevice(<XUSB address type> parentAddr, uint8 portNum);

//Parent and port will be 0 in this case

uint8 DTT\_SetDeviceSpeed(<XUSB address type> addr);

void ULM\_ConnectLexUsbPort(eUsbSpeedT speed);

void ULM\_GenerateLexUsbRemoteWakeup();

eUsbSpeedT ULM\_ReadDetectedSpeed();

void XLRC\_XlexEnable(void);

void XLRC\_XlexDisable(void);

void XCSR\_XUSBXctmEnable(void);

void XCSR\_XUSBXctmDisable(void);

void XCSR\_XUSBXcrmEnable(void);

void XCSR\_XUSBXcrmDisable(void);

void XCSR\_XUSBClearDataPath(void); // may not be needed in GoldenEars

void XLRC\_ClearQueueTracker(void);

void XCSR\_ClearXICSCache(void);

void XCSR\_XUSBAllocateStaticQueues(void);

void XCSR\_XICSFlushQueue(uint8 qid);

void XCSR\_XUSBEnableSystemQueueInterrupts(void);

void XCSR\_XUSBDisableSystemQueueInterrupts(void);

void XCSR\_XICSSendLexRexMsg(enum lexRexMsgT, msg\_data);

void XCSR\_SetUSBHighSpeed(void);

void XCSR\_SetUSBLowFullSpeed(void);

void ULM\_SetInterPacketGap(eUsbSpeedT speed);

void CLM\_SetBackToBackTimeOut(eUsbSpeedT speed);

Current Status (April 8, 2010)

Complete, however some of the lower level driver API is changing

Work Estimate:

1 week

### RexULM

Brief

Handles all of the events received from XUSB hardware and USB events relayed from the Lex. Manages device USB connection.

Exposes

void REXULM\_GeneralIsr(void);

void REXULM\_LexRexLinkUp(void);

void REXULM\_LexRexLinkDown(void);

void LEXULM\_IncomingMsg(enum ?);

Calls

uint32 ULM\_ReadInterrupts();

void ULM\_DisconnectRexUsbPort();

void ULM\_ConnectRexUsbPort();

void ULM\_RexExtendResetClear();

void ULM\_RexExtendResetRestore();

void ULM\_RexExtendResetBegin();

void ULM\_GenerateRexUsbExtendResume();

void ULM\_ClearRexUsbExtendResume();

void ULM\_GenerateRexUsbReset(eUsbSpeedT speed);

void ULM\_GenerateRexUsbSuspend(eUsbSpeedT speed);

void ULM\_GenerateRexUsbResume();

void ULM\_SetInterPktGap(eUsbSpeedT speed);

void CLM\_SetBackToBackTimeOut(eUsbSpeedT speed);

void GRG\_GpioSetLed(eGpioT led);

void GRG\_GpioClearLed( eGpioT led);

void GRG\_GpioPulseLed( eGpioT led);

void XCSR\_XUSBClearDataPath(void); // may not be needed in GoldenEars

void XCSR\_SetUSBHighSpeed(void);

void XCSR\_SetUSBLowFullSpeed(void);

void XLRC\_XrexSOFEnable(void);

void XLRC\_XrexEnable(void);

void XLRC\_XrexDisable(void);

void XCSR\_ClearXICSCache(void);

void XCSR\_XUSBAllocateStaticQueues(void);

void XCSR\_XICSQQueueFlush(uint8 qid);

void XCSR\_XICSSendLexRexMsg(enum lexRexMsgT, msg\_data);

void XLRC\_RexSOFEnable();

void XLRC\_RexEnable();

void XCSR\_XUSBXctmEnable(void);

void XCSR\_XUSBXctmDisable(void);

void REXSCH\_Enable(eUsbSpeedT speed);

void REXSCH\_Disable();

Current Status (April 8, 2010)

Not started. No work to build on

Work Estimate:

5 weeks

### DevMgr

Brief

Manages device connection, disconnection, status, and address assignments passing the required information to the Device Topology Tree. Also manages the USB address 0 control endpoint during enumeration.

**Exposes**

void DEVMGR\_ProcessSetAddr(<USB address type> currAddr, <USB address type> newAddr);

void DEVMGR\_PostProcessSetAddr(<XUSB address type> oldAddr);

void DEVMGR\_InitHubPortStatusHandler(<XUSB address type>, uint8 portNum);

void DEVMGR\_HubPortStatusHandler(<XUSB address type>, uint8 dataBuf[8]);

void DEVMGR\_ProcessPortReset(<XUSB address type>, uint8 portNum);

void DEVMGR\_DisablePort(<XUSB address type>, uint8 portNum);

//formely processClearFeature

Calls

uint8 DTT\_ResetDevice(<XUSB address type> parentAddr, uint8 portNum);

void DTT\_UpdateAddress(<XUSB address type> currAddr, <XUSB address type> newAddr);

void DTT\_DisableDevice(<XUSB address type> parentAddr, uint8 portNum);

void DTT\_SetPortStatusPort(<XUSB address type> addr, uint8 portNum);

uint8 DTT\_GetPortStatusPort(<XUSB address type> addr, uint8 portNum);

void DTT\_SetDeviceSpeed(<XUSB address type> addr, eUsbSpeedT speed);

void XCSR\_XSSTAddress0Enable(void);

void XCSR\_XSSTAddress0Disable(void);

void XCSR\_XSSTClearBCO(struct XCSR\_XusbAddress \*pAddress);

void LEXULM\_ForceEnumeration();

Current Status (April 8, 2010)

Needs an overhaul from its current implementation. There are a number of changes occurring between the DevMgr and Topology that are for both LG1 and GE. In addition, for GE there will be changes between SysQ and DevMgr.

Work Estimate:

3 weeks (maybe more but will overlap with topology changes)

### DescParser

Brief

Parses USB descriptor packets to determine what kind of devices are downstream.

Exposes

void PARSER\_InitProcessSetupPacket(<XUSB address type>, uint16 requestedLength);

boolT PARSER\_ProcessData(<XUSB address type>, uint8 \*dataBuf, uint16 size);

//returns TRUE when done, FALSE otherwise: used by caller to clear BCO

Calls

void DTT\_UpdateEndpoint(<XUSB address type>, uint8 endPointNum, uint8 attributes);

void DTT\_SetHub(<XUSB address type>);

//work around LG1 bug?

state (stored internally or by topology): requested amount of data, how many bytes have being parsed, ctrl ep0 packet size, cfg descriptor size

Current Status (April 8, 2010)

Currently implemented but contains more knowledge than it should of the system. Need to move some things down to the topology and others back up to the SysQ. Half the changes can be done on LG1 chip.

Work Estimate:

1 week

### LinkMgr

Brief

Manages the link between the Lex and Rex. Includes pings, phy checking, activity monitoring, etc.

Thoughts:

* Pings should always be replied to with a pong. Pongs are indicative of a working link, but discarded. Pings are also indicative of a working link
* Pings contain version info. We only respond to a ping, when the version matches.
* Pings are only sent initially to establish a link, and when there is no activity.
* When pings are used, timers should also be set and activity ignore, while waiting to see if the version matches.

Exposes

void LINKMGR\_Init(enum linkT);

configures ISR and enables the ISR

void LINKMGR\_IncomingPing(uint8 rexId);

//Arg not needed if there is no virtual hub, 0 could mean Lex

void LINKMGR\_IncomingPong(uint8 rexId);

//Arg not needed if there is no virtual hub, 0 could mean Lex

**Calls**

void XCSR\_XICSSendLexRexMsg(enum lexRexMsgT, msg\_data);

//To send pings and pongs

void GRG\_GpioSet(LINK\_LED);

void GRG\_GpioClear(LINK\_LED);

void LEXULM\_LexRexLinkUp(void);

void REXULM\_LexRexLinkUp(void);

void VHUB\_RexLinkUp(uint8 rexid);

void LEXULM\_LexRexLinkDown(void);

void REXULM\_LexRexLinkDown(void);

void VHUB\_RexLinkDown(uint8 rexid);

void GRG\_MdioWrite(uint8 device, uint8 address, uint16 data);

uint16 GRG\_MdioRead(uint8 device, uint8 address);

leon timers. For polling the phy status on some links & for polling the link stats

leon interrupts. For configuring the interrupts on some links

Current Status (April 8, 2010)

The LG1 code is quite a bit more complicated than it needs to be. It is also very intertwined with the PLL code. It needs a new implementation.

Work Estimate:

5 weeks

### SysQ

Brief

Has knowledge of the USB packet structure & BCO/BCI. Calls various subsystems as needed.

Exposes

void SYSQ\_ISR(void);

//configured by project initialization

**Calls**

boolT XCSR\_XICSReadQueue(<queue type identifier>, <pointer to a structure for a Q frame>);

//Update name

DTT to get <XUSB address type> from a physical address

void PARSER\_InitProcessSetupPacket(<XUSB address type>, uint16 requestedLength);

void PARSER\_ProcessData(<XUSB address type>, uint8 \*dataBuf, uint16 size);

void DEVMGR\_ProcessSetAddr(<XUSB address type> currAddr, <XUSB address type> newAddr);

void DEVMGR\_PostProcessSetAddr(<XUSB address type> oldAddr);

void DEVMGR\_InitHubPortStatusHandler(<XUSB address type>, uint8 port);

void DEVMGR\_HubPortStatusHandler(<XUSB address type>, uint8 dataBuf[8]);

TODO: dbl check in C book, how arrays are passed (value? reference?)

void DEVMGR\_ProcessPortReset(<XUSB address type>, uint8 port);

void DEVMGR\_ProcessClearFeature(<XUSB address type>, uint8 port);

//rename to DEVMGR\_DisablePort

void DTT\_HandleSetInterface();//args?

void DTT\_HandleSetConfiguration();//args?

void XCSR\_XSSTClearBCO(struct XCSR\_XusbAddress \*pAddress);

void <some vhub function>();//args?

Current Status (April 8, 2010)

In LG1 the SysQ is represented by 2 queues, and there is significant code to ensure that these queues are in sync. Now that there is only a single queue, there is no need for any synchronization code. There are cases where we flip between endians, and flip again later, which should all be cleaned up. This code will also change as the lower level XCSR driver changes.

Work Estimate:

3 weeks

## FlashWriter

Brief

Writes to the flash. Not part of the main project, as it is loaded by Xmodem and replaces the code with its own.

Exposes

Nothing, it’s a top level component.

Calls

void XMODEM\_InterruptModeInit(void);

boolT XMODEM\_InterruptModeReceive(uint32 \* (\*rxDataHandlerFunction)(uint32 \* buf, uint32 bufSize), uint32 \* buffer1, uint32 \* buffer2);

leon timers

leon flash interface

leon interrupts

leon uart

Current Status (April 8, 2010)

Complete for LG1. Almost done for GE. GE has some register changes for the flash interface.

Work Estimate:

30 minutes

## ROM ( BootLoader)

**Brief**

This is for the entry point for the processor after a reset (including power on reset). The component checks where it should boot from, and then boots from there. If the boot location is the uart, it starts an xmodem transfer to obtain an image, and then runs that image from IRAM.

With the use of a jump table and the option of putting some of the main firmware into ROM, this could significantly change

**Exposes**

Nothing, it’s a top level component.

**Calls**

Spectareg boot configuration pins read macro

Leon uart and timer initialization

boolT XMODEM\_PolledModeReceive(uint32 \* (\*rxDataHandlerFunction)(uint32 \* buf, uint32 bufSize), uint32 \* buffer);

Current Status (April 8, 2010)

Complete for GE’s current register files. As the boot pins change it will need updating

Work Estimate:

30 minutes for just the boot pin changes

# Development Timeline

## Phase 1 - GE code structure on LG1 chip

### Phase 1A - driver updates - done

### Phase 1B - low level code updates - done

### Phase 1C - high level code updates – mostly done

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Weeks of Work | Priority | Dependencies | Notes |
| LinkMgr | 3 | P5 |  | Will not be done for a Display Port based project, as this would require a new display port link manager, which wouldn’t share code with LG1 SW. |
| RexULM | 5 | P3 |  |  |
| Topology | 2 | P3 |  |  |
| Set Intf/Cfg - SW | 3 | P4 |  | This is the SW solution for tracking as many differences as possible |
| Update GCC | 1 | P3 |  | The new GCC 4.5.x improvements should directly affect our SW. I’m mostly looking at the new enum-compare warning, and whole program optimization. |

## Phase 2 - SW on limited GE FPGA (GMII only)

### Phase 2A – Base drivers

On completion we should be able to create basic links both CLM & ULM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Weeks of Work | Priority | Dependencies | Notes |
| XCSR | 2 | P1 |  | Critical Path |
| XLRC | 1 | P2 |  |  |
| CPU Messages | 1 | P2 |  |  |
| ULM | 0.2 | P2 |  | Just needs testing |
| CLM | 0.2 | P2 |  | Just needs testing |

### Phase 2B – Basic USB traffic

On completion we should be receiving USB traffic, however this is not a functional system

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Weeks of Work | Priority | Dependencies | Notes |
| Topology/XCSR | 2 | P2 | XCSR | Logical address increase, BCI, XSST interface, etc. |
| SysQ | 2 | P2 | XCSR |  |

### Phase 2C - Full USB functionality w/o VHub

This would include testing for both MII & GMII

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Weeks of Work | Priority | Dependencies | Notes |
| DevMgr | 1 | P2 | Topology/XCSR | Set Address updates |
| SW Testing |  |  |  |  |
| QA spot checking |  |  |  |  |
| QA regression |  |  |  |  |

### Phase 2D – GE with VHub

This could also be pushed onto Atlas dev. board, if available

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Weeks of Work | Priority | Dependencies | Notes |
| VHub | 4 | P3 | SysQ | Also would require updating the linkmgr for handling multiple Rexes |
| SW Testing |  |  |  |  |
| QA spot checking |  |  |  |  |
| QA regression |  |  |  |  |

## Phase 3 - Full functionality of SW on GE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Weeks of Work | Priority | Dependencies | Notes |
| Set Intf/Cfg – RTL |  | P4 |  | RTL solution for us to inject get descriptor requests. |
| Test and Verify the rest of the links |  | P4 |  | Unknown what would be prioritized here |
| Jump Table |  | P3 to determine feasibility.  P5 for implementation |  | Brainstorming exists, with a few days of work, we could work out a estimated schedule. |

## Phase 4 – Atlas FPGA board development

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Weeks of Work | Priority | Dependencies | Notes |
| DisplayPort (F)Aux Support |  |  |  | Not yet defined |
|  |  |  |  |  |