cyMUX API

December 17, 2015

Revision 0.02

Intel DPCAE

Revision History

| Date | Rev | Modifications |
| --- | --- | --- |
| 10/29/15 | 0.01 | Initial creation |
| 12/17/15 | 0.02 | Change Responder not available from timeout to 0xD3, not available |

Disclaimers

Copyright ©2015 Intel Corporation**.** All Rights Reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Intel Corporation.

Table of Contents

Revision History ii

Disclaimers ii

Table of Contents iii

List of Figures iv

List of Tables iv

1 Introduction 5

2 Connection Model 6

2.1 Registration 6

2.2 Responder Mode 6

2.3 Client Mode 7

2.4 No Registered Responder 7

2.1 Duplicate Responder Registration 8

3 Message format 9

3.1 Example Request 9

3.2 Example Response 9

3.3 Response failures 9

3.4 MUX timing out on expected responses 9

List of Figures

Figure 1: Responder data flow 6

Figure 2: Client data flow 7

Figure 3: No Registered Responder data flow 7

Figure 4: Duplicate Responder Registration Request 8

Figure 5: MUX timeout to outbound request 10

Figure 6: MUX timeout to inbound request 10

List of Tables

Table 1: Example client request 9

Table 2: Example response to client 9

# Introduction

This document provides an API of how to speak with the CY7C65215 multiplex daemon. cyMUX is a dedicated daemon for the BDC-R CPP, which will manage all asynchronous IPMB messages with and from the CY7C65215. cyMUX will receive all OOB requests route them to a registered client. It will allow multiple clients to attach to send out bound requests and will return the responses to those requests back to the proper client. It will be solely responsible for communication with the CY7C65215 and perform the necessary serialization of communication to the CY7C65215. It will properly tag and handle the asynchronous (and potentially out of order) responses to ensure that packets are routed to the correct client.

To allow flexibility and ease of refactoring, the API to cyMUX will assume fully qualified IPMB messages as defined by the IPMI specification.

# Connection Model

## Registration

A socket connection is made to cyMUX on port 4623/tcp. A registration byte is sent to tell the MUX its primary mode (‘C’lient or ‘R’esponder). The MUX will return either ‘OK’ or ‘NOK’ (with the later closing the connection). Only one server may be registered at a time. A server closing the connection will automatically deregister it from the MUX.

## Responder Mode

After registration, the MUX will write any incoming IPMB requests received to the socket of the registered server. The MUX will then wait for/read any appropriate response to write back to the requestor. As long as the socket is active, the new IPMB requests will be received and any response provided.



Figure 1: Responder data flow

## Client Mode

After registration, the MUX will wait for/read any request to send. When a response is received, it will be written.



Figure 2: Client data flow

## No Registered Responder

While there is no registered responder, the MUX will reject in request (e.g., invalid NetFn/Cmd).



Figure 3: No Registered Responder data flow

## Duplicate Responder Registration

At this time, keep it simple. There can only be one registered responder. While there is an active socket registered as a responder, any future requests are rejected with a ‘NOK’ and the socket closed.



Figure 4: Duplicate Responder Registration Request

# Message format

Registration when a socket is first connected to the MUX should be a ASCII byte of either ‘C’ or ‘R’. The MUX should return an ASCII string of either ‘OK’ or ‘NOK’ to acknowledge/reject the registration.

To minimize the work already done and make refactoring of existing systems, the raw binary IPMB message is to be transferred. Requests and responses should be preceded by a single byte of the slave address of where to send the message. However, the RqSA (and related checksums) of outbound requests may be rewritten to match the configuration of the hardware (0x08 by default).

## Example Request

In this example, the request is for the device id of the MMP (no bridging). MMP has slave address of 0x24. The default slave address of the Cypress CY7C65215 is 0x10. A sequence number of 3 is used in this example. Get device id is net function 6, command 1, (no data).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Slave Addr (7-bit) | NetFn/Lun  NetFn=0x6  Lun=0  0001 10 00 | ChkSum (Slave Addr + NetFn/Lun) 2's complement % 256 -(0x24+0x18) % 256  =0xC4 | RqSA Our Slave Addr | Seq / RqLun  0111 10 00 | IPMI cmd 0x1 = DeviceID | ChkSum (RqSa + Seq/RqLun) 2's complement % 256 -(0x10+0x78+0x01) % 256 =0x77 |
| 0x12 | 0x18 | 0xC4 | 0x10 | 0x78 | 0x01 | 0x77 |

Table 1: Example client request

## Example Response

For our above example, we may receive a response of the following form:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Slave Addr (7-bit) | NetFn/Lun  NetFn=0x7  Lun=2  0001 11 10 | ChkSum (Slave Addr + NetFn/Lun) 2's complement % 256 -(0x10+0x1E) % 256  =0xC4 | RqSA Their Slave Addr | Seq / RqLun  0111 10 00 | IPMI cmd 0x1 = DeviceID | CC | Data | ChkSum (RqSa + Seq/RqLun) 2's complement % 256 -(0x24+0x78+0x01+0x00 + <data>) % 256 =0x95 |
| 0x08 | 0x1E | 0xC4 | 0x24 | 0x78 | 0x01 | 0x00 | 0x25 0x00 0x01 0x08 0x02 0xB8 0x4C 0x1C 0x00 0x4D 0x31 0 0 0 0 | 0x95 |

Table 2: Example response to client

## Response failures

When a time out, device failure, or bus failure occur, the MUX will craft a reasonable IPMB response with the CC flagging the failure (e.g., 0xC3 for timeout, 0xD3 for destination not available, 0xFF catastrophic failure, etc. [to be expanded as corner cases handled]). If a client requests a packet for the I2C and a NAK is returned (e.g., no slave device), a 0xD3 CC will be returned to the client (not available).

## MUX timing out on expected responses

The MUX will return a corresponding IPMB packet with the CC set to 0xC3 when it internally times out waiting for the requested device to respond after X amount of time (possibly make configurable via configuration or command line execution). This time out will be for both devices on the I2C as well as for the responder daemon.



Figure 5: MUX timeout to outbound request



Figure 6: MUX timeout to inbound request