Media Agnostic Universal Serial Bus

BITS ZG628T: Dissertation

by

Nagaraju Kadiri

2014HT13190

Dissertation work carried out at

LG Soft India Private Limited
Bangalore.

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI (RAJASTHAN)

November 2016

Media Agnostic Universal Serial Bus

BITS ZG628T: Dissertation

by

Nagaraju Kadiri

2014HT13190

Dissertation work carried out at

LG Soft India Private Limited
Bangalore.

Submitted in partial fulfillment of M.Tech. Software Systems Degree programme

Under the Supervision of
Kishore PV, Project Manager,
LG Soft India Pvt. Ltd, Bangalore.

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI (RAJASTHAN)

November 2016

CERTIFICATE

This is to certify that the Dissertation entitled **Media Agnostic Universal Serial Bus** and submitted by **Nagaraju Kadiri** having ID-No. **2014HT13190** for the partial fulfillment of the requirements of M.Tech. Software Systems degree of BITS, embodies the bonafide work done by him under my supervision.

Signature of the Supervisor

Place : Bangalore
Date : 17/10/2016

Name: Kishore PV,
Designation: Project Manager,
Organization: LG Soft India Pvt Ltd
Location: Bangalore.

Birla Institute of Technology & Science, Pilani

Work-Integrated Learning Programmes Division

First Semester 2016-2017

BITS ZG628T: Dissertation

ABSTRACT

BITS ID No. : **2014HT13190**

NAME OF THE STUDENT : **Nagaraju Kadiri**

EMAIL ADDRESS : **raju.kadiri@lge.com**

STUDENT'S EMPLOYING ORGANIZATION & LOCATION : **LG Soft India PVT LTD, Bangalore.**

SUPERVISOR'S NAME : **Kishore PV**

SUPERVISOR'S EMPLOYING ORGANIZATION & LOCATION : **LG Soft India PVT LTD, Bangalore.**

SUPERVISOR'S EMAIL ADDRESS: Kishore.pv@lge.com

DISSERTATION TITLE : **Media Agnostic Universal Serial Bus**

Signature of the Student

Name: NagarajuKadiri

Date: 17/10/2016

Place: Bangalore

Signature of the Supervisor

Name: Kishore PV

Date: 17/10/2016

Place: Bangalore

ABSTRACT

USB micro or mini port is common in every embedded device. In case of wearable devices manufacturer will provide separate cradle system for USB functionality. The cradle system will have USB micro port. The cradle system is required to do any USB operation in wearable devices. Media Agnostic (MA) Universal Serial Bus is to provide USB connectivity over medium other than USB. Medium can be either WiFi or ZigBee or Bluetooth. Using MAUSB protocol we will get same functionality as if USB micro port is available in wearable devices. No need of extra cradle system for USB functionality in wearable devices. MAUSB can be used for water proof device (USB port not available) firmware flashing, remote debugging and diagnostic the device. No more USB data cables are required. MAUSB specifications are developed by USB forum.

Broad Academic Area of Work:

Embedded Systems, Wireless and Mobile Computing

Key words

USB (Universal Serial Bus), MAUSB (Media Agnostic USB), Wearable Devices, WiFi,ZigBe

ACKNOWLEDGEMENTS

The satisfaction and euphoria that accompanies the successful completion of any task would be incomplete without mentoring the people who made it possible, because success is the epitome of hard work, perseverance, undeterred missionary zeal, steadfast determination and most of all "Encouraging Guidance".

I express my gratitude to my supervisor **Mr. Kishore PV** for providing me a means of attaining my most cherished goals.

I record my heart full of thanks and gratitude to my additional examiner **Mr. Suresh Kumar J** for providing me an opportunity to carry this project, along with purposeful guidance and moral support extended to me throughout the duration of the project work.

Nagaraju Kadiri

Table of contents

List of Figures

1. Introduction

This document describes the design of Media Agnostic Universal Serial Bus Driver for Linux platform. The driver will provide Connectivity over medium other than USB, for example wireless or IP links, while making the maximum use of existing USB infrastructure.

2. Definitions, Abbreviations, and Acronyms

EP handle: A 16-bit number uniquely identifying a USB endpoint within an MA USB device. The endpoint belongs to a USB device The combination of MA USB device address and EP handle uniquely identifies a USB device endpoint within an MA USB Service Set.

Universal Serial Bus (USB) address: A unique address in the form of a seven-bit value assigned by the USB system software to the USB device.

USB Device Handle: A 16-bit number uniquely identifying a USB device behind an MA USB device; also called device handle for short.

Media Agnostic Universal Serial Bus (MA USB): The Protocol Adaptation Layer (PAL) defined in this specification, which enables connectivity between a USB host and one or more USB devices, including USB hubs, over medium other than USB, including wireless and IP links.

MA USB device: An MA USB architectural element that integrates a USB device, and manages USB transfers targeting the USB device over a network connection. The integrated device may be connected through wired USB (USB cable, USB chip-to-chip interconnect, etc.) or other technologies, but through the MA USB device, it is presented to the host system as a USB device compliant with Revision 2.0 or 3.1 of the USB specification.

MA USB device address: An 8-bit number uniquely identifying an MA USB device within an MA USB Service Set.

MA USB host: An architectural element of the MA USB PAL that includes a physical link interface and USB host logic as defined in USB Specifications.

MA USB Service Set (MSS): The collection of an MA USB host and all MA USB devices it manages. Each MSS defines an independent addressing domain, i.e., an MA USB device address is unique within the scope of the MSS to which the MA USB device belongs.

MA link: The end-to-end connection between an MA USB host and an MA USB device or an MA USB hub. An MA link may consist of multiple network segments.

USB	Universal Serial Bus
MAUSB	Media Agnostic USB
UDC	USB Device Controller
MA	Media Agnostic
MS	Media Specific
SSID	Service Set Identifier
PHY	Physical layer
PAL	Protocol Adaptation Layer
OS	Operating System
MSS	MA USB Service Set
EP	Endpoint

3. USB Architecture

3.1 USB Protocol Architecture

The USB is based on a so called 'tiered star topology' in which there is a single host controller and up to 127 'slave' devices. The host controller is connected to a hub, integrated within the PC, which allows a number of attachment points (often loosely referred to as ports). A further hub may be plugged into each of these attachment points, and so on.
However there are limitations on this expansion.

As stated above a maximum of 127 devices (including hubs) may be connected. This is because the address field in a packet is 7 bits long, and the address 0 cannot be used as it has special significance. (In most systems the bus would be running out of bandwidth, or other resources, long before the 127 devices was reached.)

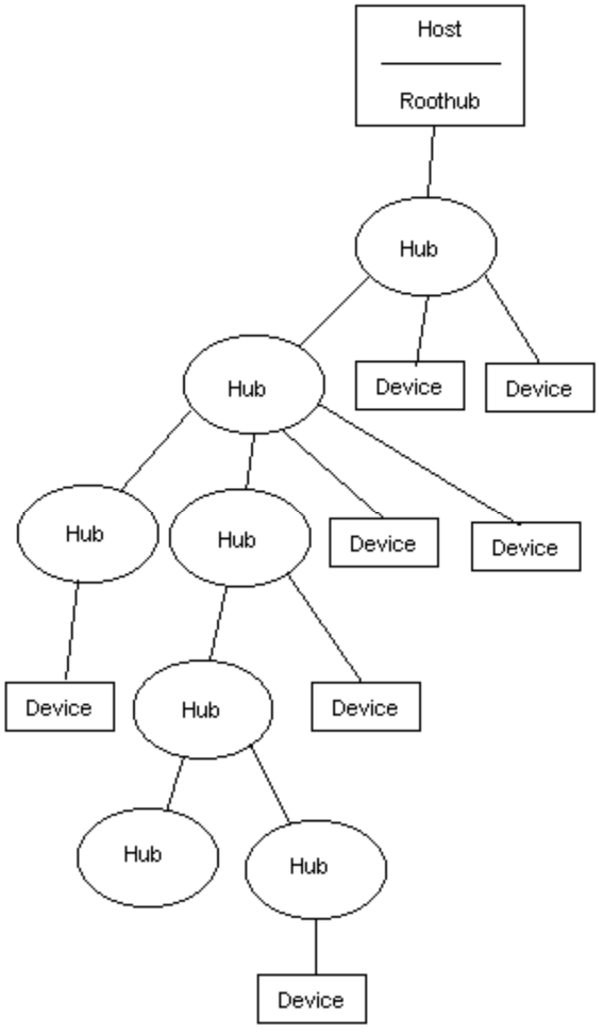

Figure 1: USB Topology

A device can be plugged into a hub, and that hub can be plugged into another hub and so on. However the maximum number of tiers permitted is six.

3.2 USB Packets

USB is made up of several layers of protocols. In fact most USB controller I.C.s will take care of the lower layer, thus making it almost invisible to the end designer. Each USB transaction consists of a

- ✓ Token Packet ,
- ✓ Optional Data Packet,
- ✓ Handshake Packet
- ✓ Start of frame Packet

Token packet used for SETUP, OUT and IN packets. They are always the first packet in a transaction, identifying the targeted endpoint, and the purpose of the transaction.

- ✓ **IN**-Informs the USB device that the host wishes to read information.
- ✓ **OUT** - Informs the USB device that the host wishes to send information.
- ✓ **SETUP** - Used to begin control transfers.

Data packets are capable of transmitting o to 1023 bytes of data. There are two types of data packets

- ✓ **DATA0**
- ✓ **DATA1**

Handshake Packets are three type of handshake packets which consist simply of the PID

- ✓ **ACK** – Acknowledgment that the packet has been successfully received.
- ✓ **NAK** – Reports that the device cannot send nor received data temporary. Also used during interrupt transaction to inform the host there is no data to send.
- ✓ **STALL** – The device finds it's in a state that it requires intervention from the host.

The Start of Frame packet is sent every 1 mill second on full speed links. The frame is used as a time frame in which to schedule the data transfers which are required. For example, an isochronous endpoint will be assigned one transfer per frame.

3.3 USB Endpoints

Endpoints can be described as sources or sinks of data. As the bus is host centric, endpoints occur at the end of the communications channel at the USB function. At the software layer, your device driver may send a packet to your devices EP1 for example. As the data is flowing out from the host, it will end up in the EP1 OUT buffer. Your firmware will then at its leisure read this data. If it wants to return data, the function cannot simply write to the bus as the bus is controlled by the host. Therefore it writes data to EP1 IN which sits

in the buffer until such time when the host sends a IN packet to that endpoint requesting the data. Endpoints can also be seen as the interface between the hardware of the function device and the firmware running on the function device.

All devices must support endpoint zero. This is the endpoint which receives all of the devices control and status requests during enumeration and throughout the duration while the device is operational on the bus.

3.4 USB Pipes

While the device sends and receives data on a series of endpoints, the client software transfers data through pipes. A pipe is a logical connection between the host and endpoint(s). Pipes will also have a set of parameters associated with them such as how much bandwidth is allocated to it, what transfer type (Control, Bulk, Iso or Interrupt) it uses, a direction of data flow and maximum packet/buffer sizes. For example the default pipe is a bi-directional pipe made up of endpoint zero in and endpoint zero out with a control transfer type. USB defines two types of pipes

- ✓ **Stream Pipes** have no defined USB format, that is you can send any type of data down a stream pipe and can retrieve the data out the other end. Data flows sequentially and has a pre-defined direction, either in or out. Stream pipes will support bulk, isochronous and interrupt transfer types. Stream pipes can either be controlled by the host or device.
- ✓ **Message Pipes** have a defined USB format. They are host controlled, which are initiated by a request sent from the host. Data is then transferred in the desired direction, dictated by the request. Therefore message pipes allow data to flow in both directions but will only support control transfers.

3.5 USB Transfer Types

The Universal Serial Bus specification defines four transfer/endpoint types,

- ✓ Control Transfers
- ✓ Interrupt Transfers
- ✓ Isochronous Transfers
- ✓ Bulk Transfers

3.5.1 Control Transfer

Control transfers are typically used for command and status operations. They are essential to set up a USB device with all enumeration functions being performed using control transfers. They are typically bursty, random packets which are initiated by the host and use best effort delivery. The packet length of control transfers in low speed devices must be 8 bytes, high speed devices allow a packet size of 8, 16, 32 or 64 bytes and full

speed devices must have a packet size of 64 bytes. A control transfer can have up to three stages.

1) The **Setup Stage** is where the request is sent. This consists of three packets. The setup token is sent first which contains the address and endpoint number. The data packet is sent next and always has a PID type of data0 and includes a setup packet which details the type of request. We detail the setup packet later. The last packet is a handshake used for acknowledging successful receipt or to indicate an error. If the function successfully receives the setup data (CRC and PID etc OK) it responds with ACK, otherwise it ignores the data and doesn't send a handshake packet. Functions cannot issue a STALL or NAK packet in response to a setup packet.

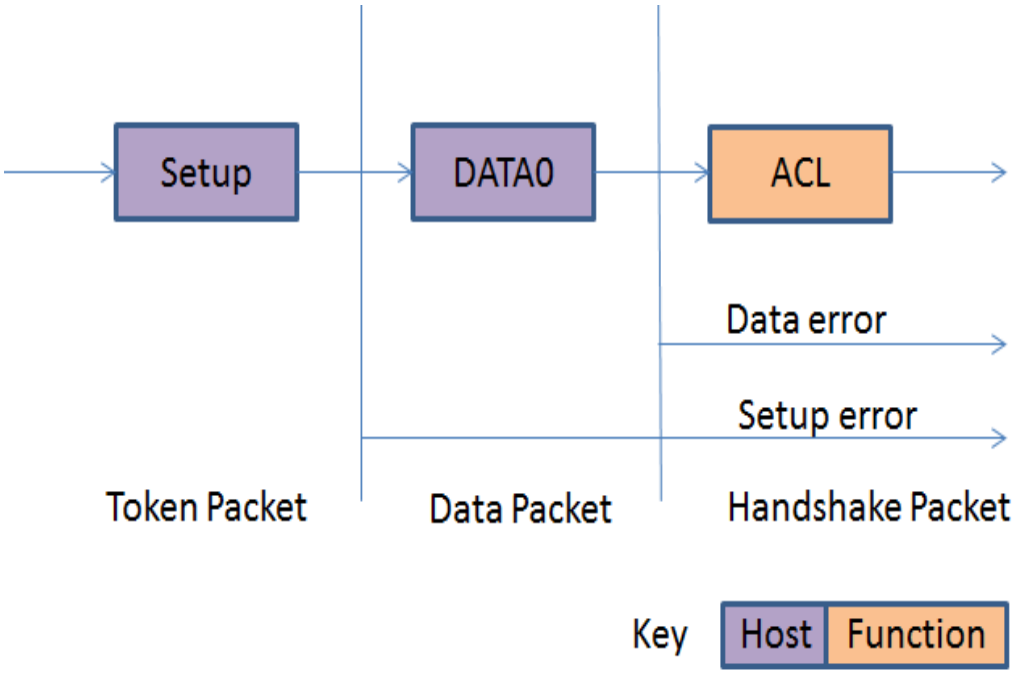

Figure 2: USB Control Transfer Setup Packet

2) The optional **Data Stage** consists of one or multiple IN or OUT transfers. The setup request indicates the amount of data to be transmitted in this stage. If it exceeds the maximum packet size, data will be sent in multiple transfers each being the maximum packet length except for the last packet.

   The data stage has two different scenarios depending upon the direction of data transfer.

a) **IN:** When the host is ready to receive control data it issues an IN Token. If the function receives the IN token with an error e.g. the PID doesn't match the inverted PID bits, then it ignores the packet. If the token was received correctly, the device can either reply with a DATA packet containing the control data to be sent, a stall packet indicating the endpoint has had a error or a NAK packet indicating to the host that the endpoint is working, but temporary has no data to send.

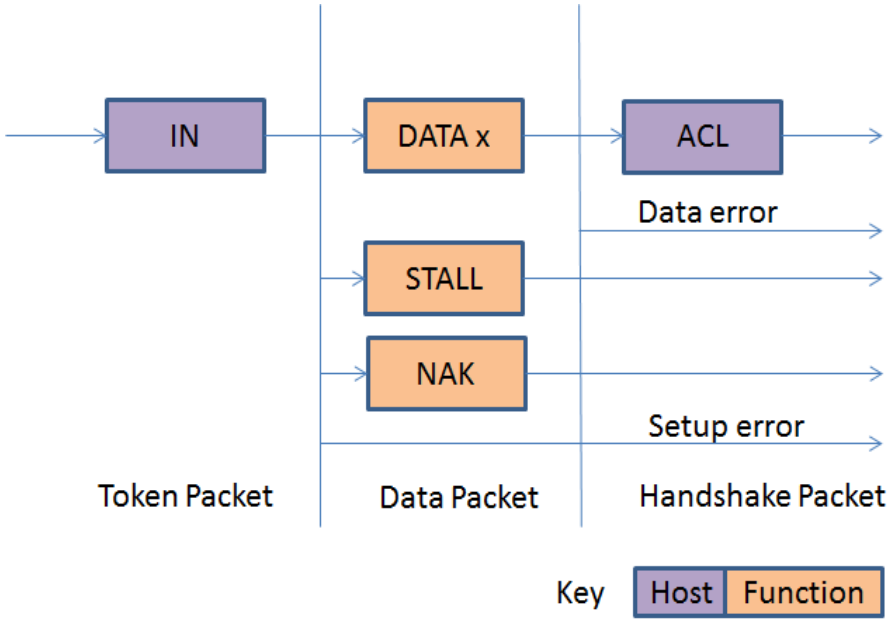

Figure 3: USB Control Transfer Data IN Packet

b) **OUT:** When the host needs to send the device a control data packet, it issues an OUT token followed by a data packet containing the control data as the payload. If any part of the OUT token or data packet is corrupt then the function ignores the packet. If the function's endpoint buffer was empty and it has clocked the data into the endpoint buffer it issues an ACK informing the host it has successfully received the data. If the endpoint buffer is not empty due to processing of theprevious packet, then the function returns a NAK. However if the endpoint has had a error and itshalt bit has been set, it returns a STALL.

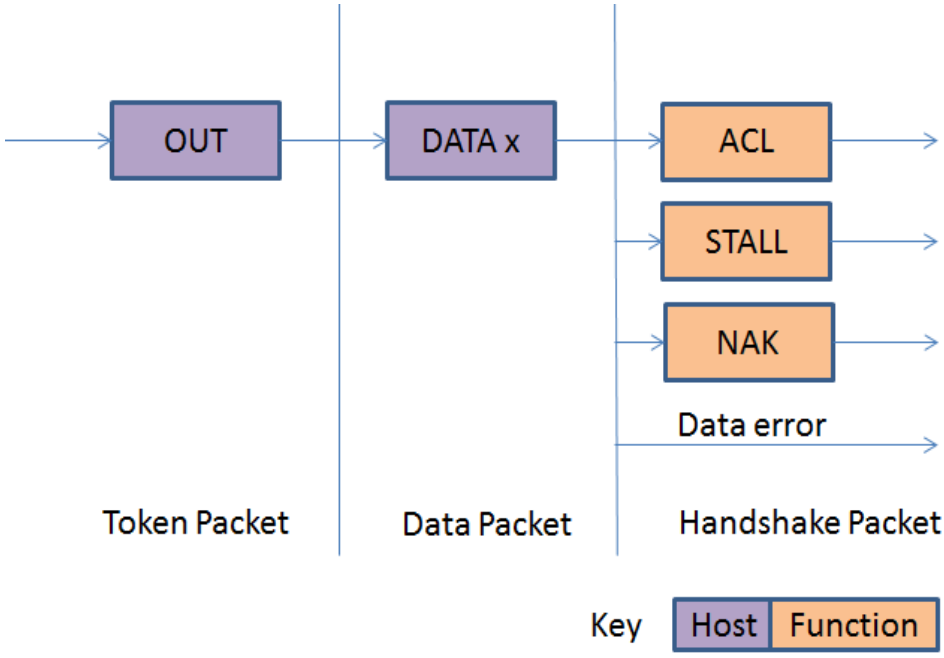

Figure 4: USB Control Transfer Data OUT Packet

3) **Status Stage** reports the status of the overall request and this once again varies due to direction of transfer. Status reporting is always performed by the function.

   a) **IN:** If the host sent IN token(s) during the data stage to receive data, then the host must acknowledge the successful receipt of this data. This is done by the host sending an OUT token followed by a zero length data packet. The function can now report its status in the handshaking stage. An ACK indicates the function has completed the command is now ready to accept another command. If an error occurred during the processing of this command, then the function will issue a STALL. However if the function is still processing, it returns a NAK indicating to the host to repeat the status stage later.

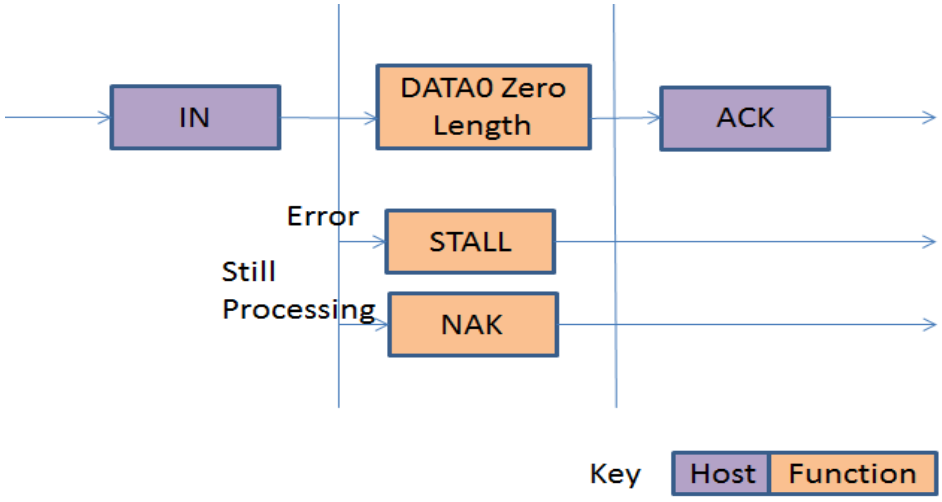

Figure 5: USB Control Transfer Status IN Packet

b) **OUT:** If the host sent OUT token(s) during the data stage to transmit data, the function will acknowledge the successful recept of data by sending a zero length packet in response to an IN token. However if an error occurred, it should issue a STALL or if it is still busy processing data, it should issue a NAK asking the host to retry the status phase later.

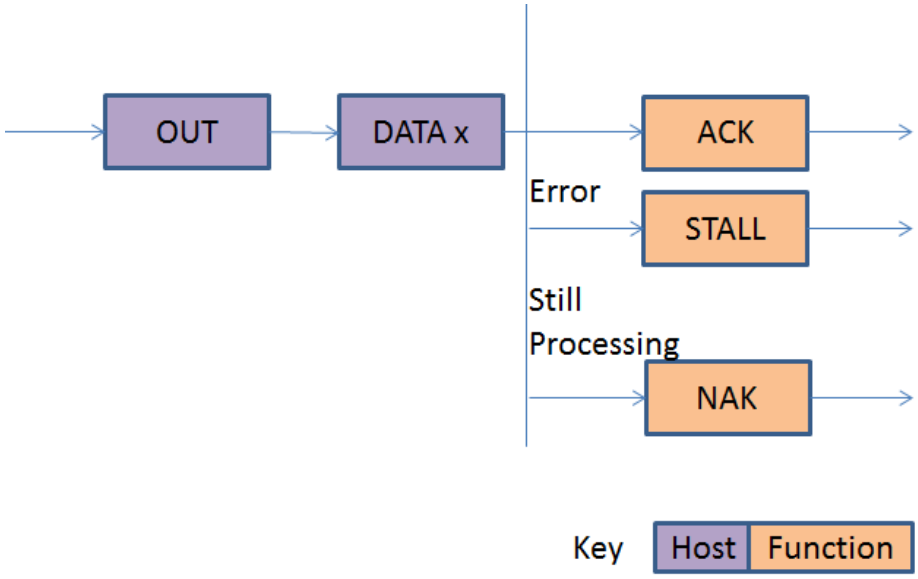

Figure 6: USB Control Transfer Status OUT Packet

3.5.2 Interrupt Transfer

Interrupt transfers have nothing to do with interrupts. The name is chosen because they are used for the sort of purpose where an interrupt would have been used in earlier connection types.

- ✓ Guaranteed Latency
- ✓ Stream Pipe - Unidirectional
- ✓ Error detection and next period retry.

Interrupt transfers are typically non-periodic, small device "initiated" communication requiring bounded latency. An Interrupt request is queued by the device until the host polls the USB device asking for data.

- ✓ The maximum data payload size for low-speed devices is 8 bytes.
- ✓ Maximum data payload size for full-speed devices is 64 bytes.
- ✓ Maximum data payload size for high-speed devices is 1024 bytes.

1) **IN:** The host will periodically poll the interrupt endpoint. This rate of polling is specified in the endpoint descriptor which is covered later. Each poll will involve the

host sending an IN Token. If the IN token is corrupt, the function ignores the packet and continues monitoring the bus for new tokens. If an interrupt has been queued by the device, the function will send a data packet containing data relevant to the interrupt when it receives the IN Token. Upon successful recept at the host, the host will return an ACK. However if the data is corrupted, the host will return no status. If on the other hand a interrupt condition was not present when the host polled the interrupt endpoint with an IN token, then the function signals this state by sending a NAK. If an error has occurred on this endpoint, a STALL is sent in reply to the IN token instead.

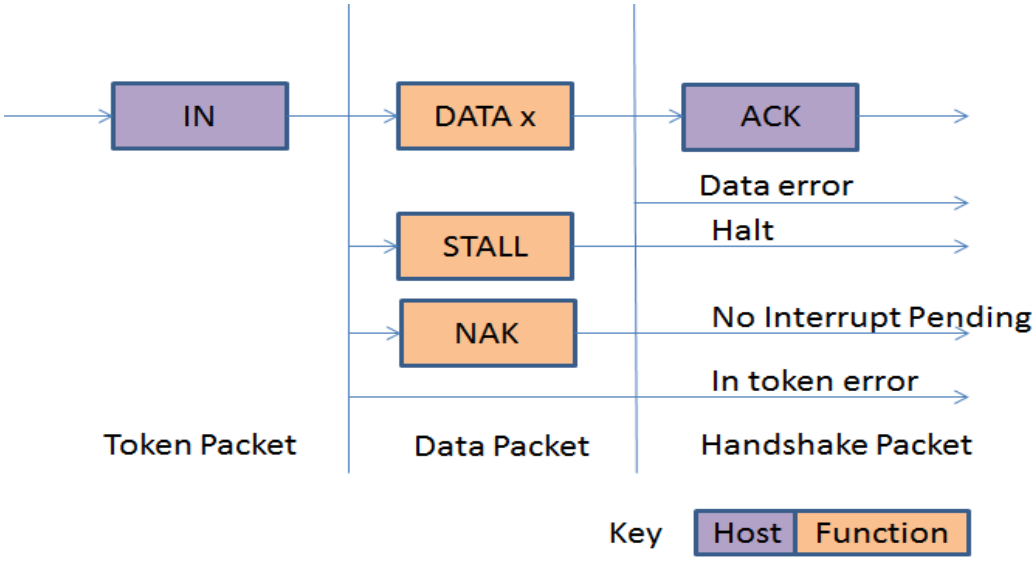

Figure 7: USB Interrupt Transfer IN Packet

2) **OUT:** When the host wants to send the device interrupt data, it issues an OUT token followed by a data packet containing the interrupt data. If any part of the OUT token or data packet is corrupt then the function ignores the packet. If the function's endpoint buffer was empty and it has clocked the data into the endpoint buffer it issues an ACK informing the host it has successfully received the data. If the endpoint buffer is not empty due to processing of a previous packet, then the function returns an NAK. However if an error occurred with the endpoint consequently and its halt bit has been set, it returns a STALL.

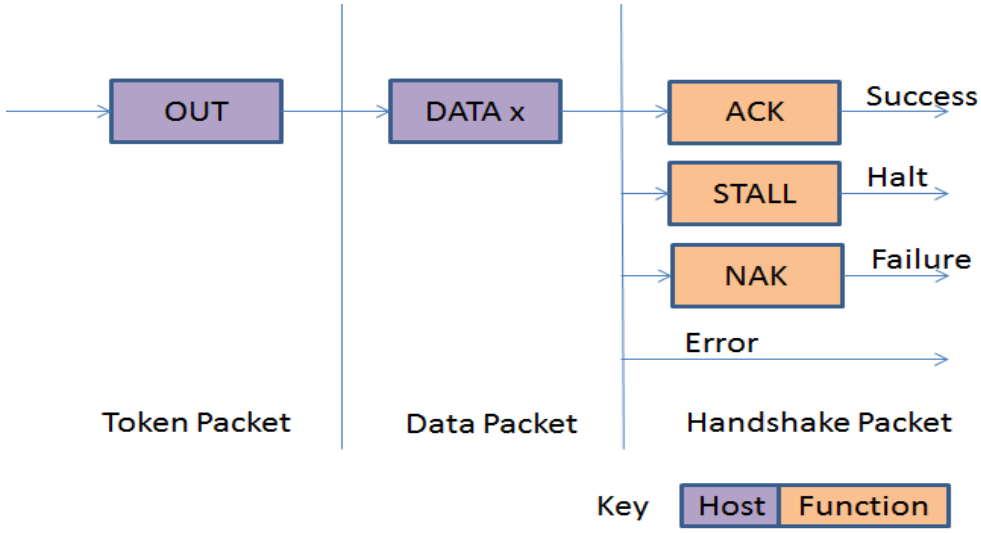

Figure 8: USB Interrupt Transfer OUT Packet

3.5.3 Bulk Transfer

Bulk transfers can be used for large bursty data. Such examples could include a print-job sent to a printer or an image generated from a scanner. Bulk transfers provide error correction in the form of a CRC16 field on the data payload and error detection/re-transmission mechanisms ensuring data is transmitted and received without error. Bulk transfers will use spare un-allocated bandwidth on the bus after all other transactions have been allocated. If the bus is busy with isochronous and/or interrupt then bulk data may slowly trickle over the bus. As a result Bulk transfers should only be used for time insensitive communication as there is no guarantee of latency.

- ✓ Used to transfer large bursty data.
- ✓ Error detection via CRC, with guarantee of delivery.
- ✓ No guarantee of bandwidth or minimum latency.
- ✓ Stream Pipe - Unidirectional
- ✓ Full & high speed modes only.

Bulk transfers are only supported by full and high speed devices. For full speed endpoints, the maximum bulk packet size is 8, 16, 32 or 64 bytes long. For high speed endpoints, the maximum packet size can be up to 512 bytes long. If the data payload falls short of the maximum packet size, it doesn't need to be padded with zeros. A bulk transfer is considered complete when it has transferred the exact amount of data requested, transferred a packet less than the maximum endpoint size of transferred a zero-length packet.

- ✓ IN token with an error, it ignores the packet. If the token was received correctly, the function can either reply with a DATA packet containing the bulk data to be sent, or a

stall packet indicating the endpoint has had a error or a NAK packet indication to the host that the endpoint is working, but temporary has no data to send.

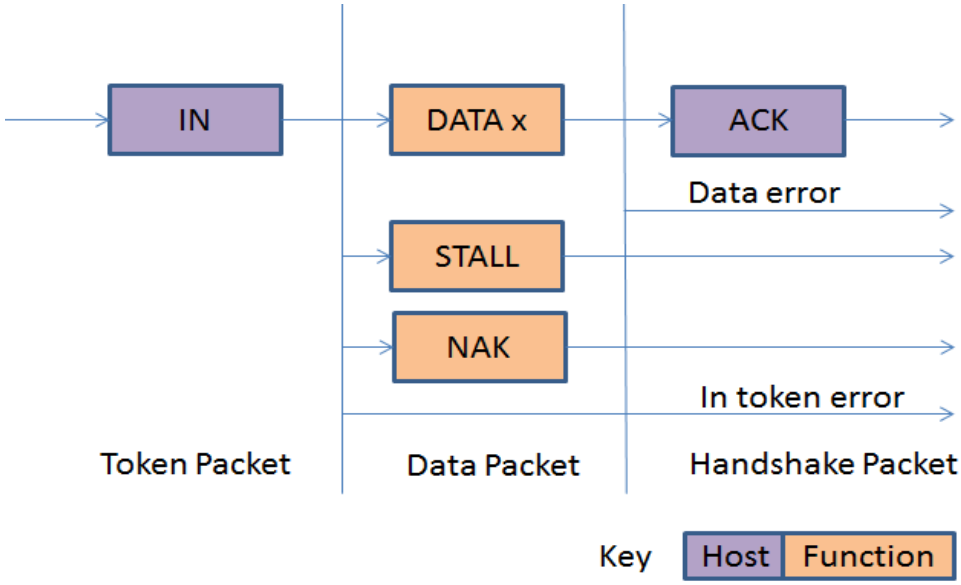

Figure 9: USB Bulk Transfer IN Packet

- ✓ **OUT**: When the host wants to send the function a bulk data packet it issues an OUT token followed by a data packet containing the bulk data. If any part of the OUT token or data packet is corrupt then the function ignores the packet. If the function's endpoint buffer was empty and it has clocked the data into the endpoint buffer it issues an AC informing the host it had successfully received the data. If the endpoint buffer is not empty due to processing a previous packet, then the function returns an NAK. However if the endpoint had had an error and it's halt bits has been set, it return a STALL.

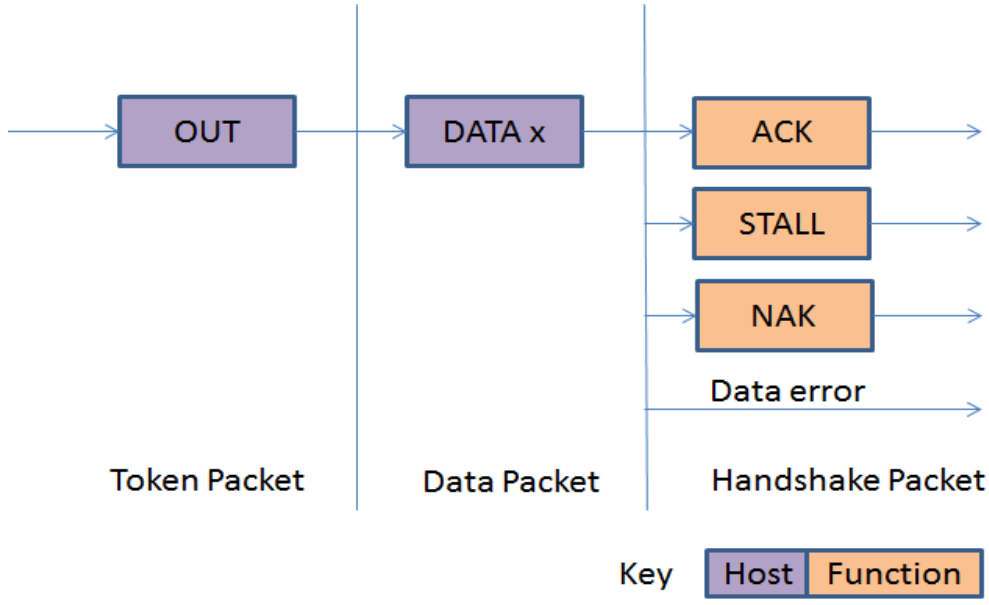

Figure 10: USB Bulk Transfer OUT Packet

3.5.4 Isochronous Transfer

Isochronous transfers occur continuously and periodically. They typically contain time sensitive information, such as an audio or video stream. If there were a delay or retry of data in an audio stream, then you would expect some erratic audio containing glitches. The beat may no longer be in sync. However if a packet or frame was dropped every now and again, it is less likely to be noticed by the listener.

- ✓ Guaranteed access to USB bandwidth.
- ✓ Bounded latency.
- ✓ Stream Pipe - Unidirectional
- ✓ Error detection via CRC, but no retry or guarantee of delivery.
- ✓ Full & high speed modes only.
- ✓ No data toggling.

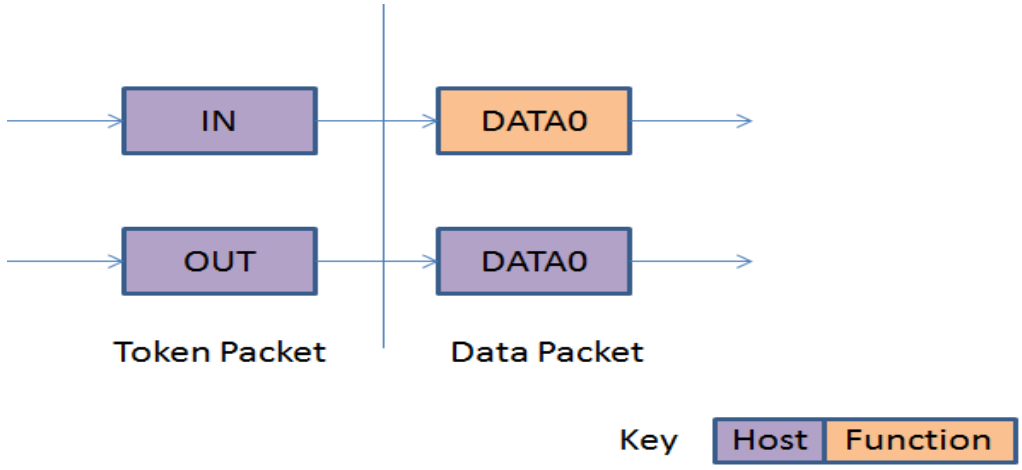

Figure 11: USB Isochronous Transfer IN/OUT Packet

The maximum size data payload is specified in the endpoint descriptor of an Isochronous Endpoint. This can be up to a maximum of 1023 bytes for a full speed device and 1024 bytes for a high speed device. As the maximum data payload size is going to effect the bandwidth requirements of the bus, it is wise to specify a conservative payload size. If you are using a large payload, it may also be to your advantage to specify a series of alternative interfaces with varying isochronous payload sizes. If during enumeration, the host cannot enable your preferred isochronous endpoint due to bandwidth restrictions, it has something to fall back on rather than just failing completely. Data being sent on an isochronous endpoint can be less than the pre-negotiated size and may vary in length from transaction to transaction.

3.6 USB Descriptors

All USB devices have a hierarchy of descriptors which describe to the host information such as what the device is, who makes it, what version of USB it supports, how many ways it can be configured, the number of endpoints and their types etc.

The more common USB descriptors are

- ✓ Device Descriptors
- ✓ Configuration Descriptors
- ✓ Interface Descriptors
- ✓ Endpoint Descriptors
- ✓ String Descriptors

USB devices can only have one device descriptor. The device descriptor includes information such as what USB revision the device complies to, the Product and Vendor IDs used to load the appropriate drivers and the number of possible configurations the device

can have. The number of configurations indicates how many configuration descriptors branches are to follow.

The configuration descriptor specifies values such as the amount of power this particular configuration uses, if the device is self or bus powered and the number of interfaces it has. When a device is enumerated, the host reads the device descriptors and can make a decision of which configuration to enable. It can only enable one configuration at a time.

For example, It is possible to have a high power bus powered configuration and a self-powered configuration. If the device is plugged into a host with a mains power supply, the device driver may choose to enable the high power bus powered configuration enabling the device to be powered without a connection to the mains, yet if it is connected to a laptop or personal organiser it could enable the 2nd configuration (self-powered) requiring the user to plug your device into the power point.

The configuration settings are not limited to power differences. Each configuration could be powered in the same way and draw the same current, yet have different interface or endpoint combinations. However it should be noted that changing the configuration requires all activity on each endpoint to stop. While USB offers this flexibility, very few devices have more than 1 configuration.

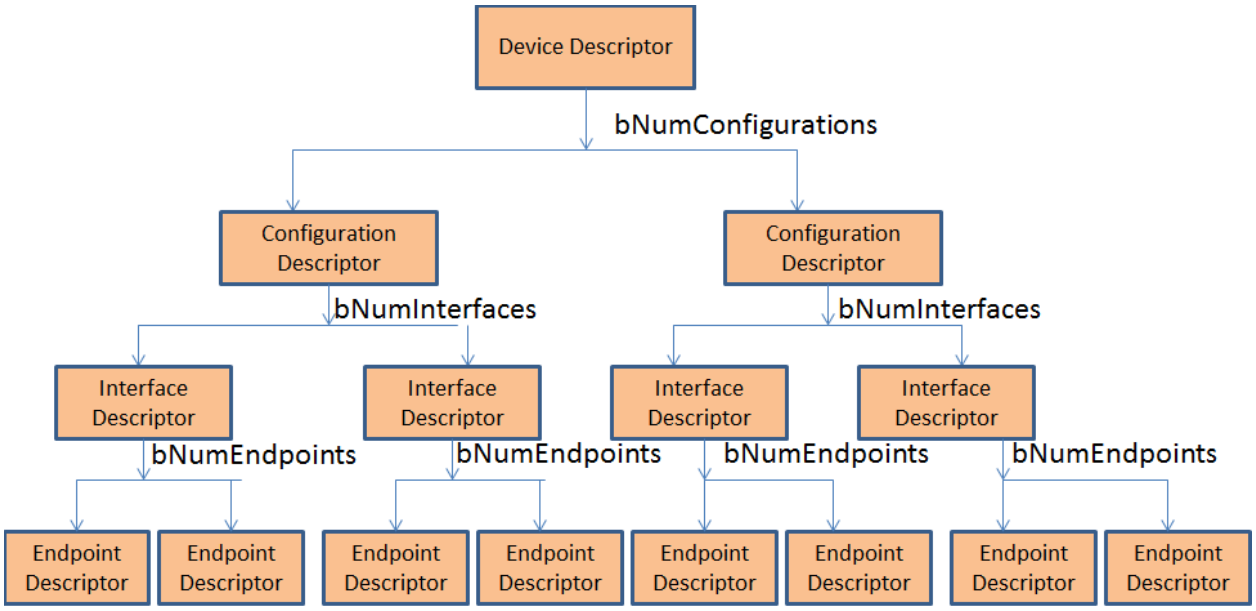

Figure 12: USB Descriptors

The interface descriptor could be seen as a header or grouping of the endpoints into a functional group performing a single feature of the device. For example you could have a multi-function fax/scanner/printer device. Interface descriptor one could describe the endpoints of the fax function, Interface descriptor two the scanner function and Interface

descriptor three the printer function. Unlike the configuration descriptor, there is no limitation as to having only one interface enabled at a time. A device could have 1 or many interface descriptors enabled at once.

Interface descriptors have a **bInterfaceNumber** field specifying the Interface number and a **bAlternateSetting** which allows an interface to change settings on the fly. For example we could have a device with two interfaces, interface one and interface two. Interface one has **bInterfaceNumber** set to zero indicating it is the first interface descriptor and a **bAlternativeSetting** of zero.

Interface two would have a **bInterfaceNumber** set to one indicating it is the second interface and a **bAlternativeSetting** of zero (default). We could then throw in another descriptor, also with a **bInterfaceNumber** set to one indicating it is the second interface, but this time setting the **bAlternativeSetting** to one, indicating this interface descriptor can be an alternative setting to that of the other interface descriptor two.

When this configuration is enabled, the first two interface descriptors with **bAlternativeSettings** equal to zero is used. However during operation the host can send a SetInterface request directed to that of Interface one with an alternative setting of one to enable the other interface descriptor.

This gives an advantage over having two configurations, in that we can be transmitting data over interface zero while we change the endpoint settings associated with interface one without effecting interface zero.

Each endpoint descriptor is used to specify the type of transfer, direction, polling interval and maximum packet size for each endpoint. Endpoint zero, the default control endpoint is always assumed to be a control endpoint and as such never has a descriptor.

4. MAUSB Architecture

The below figure provides an overall architecture diagram of Media agnostic driver.

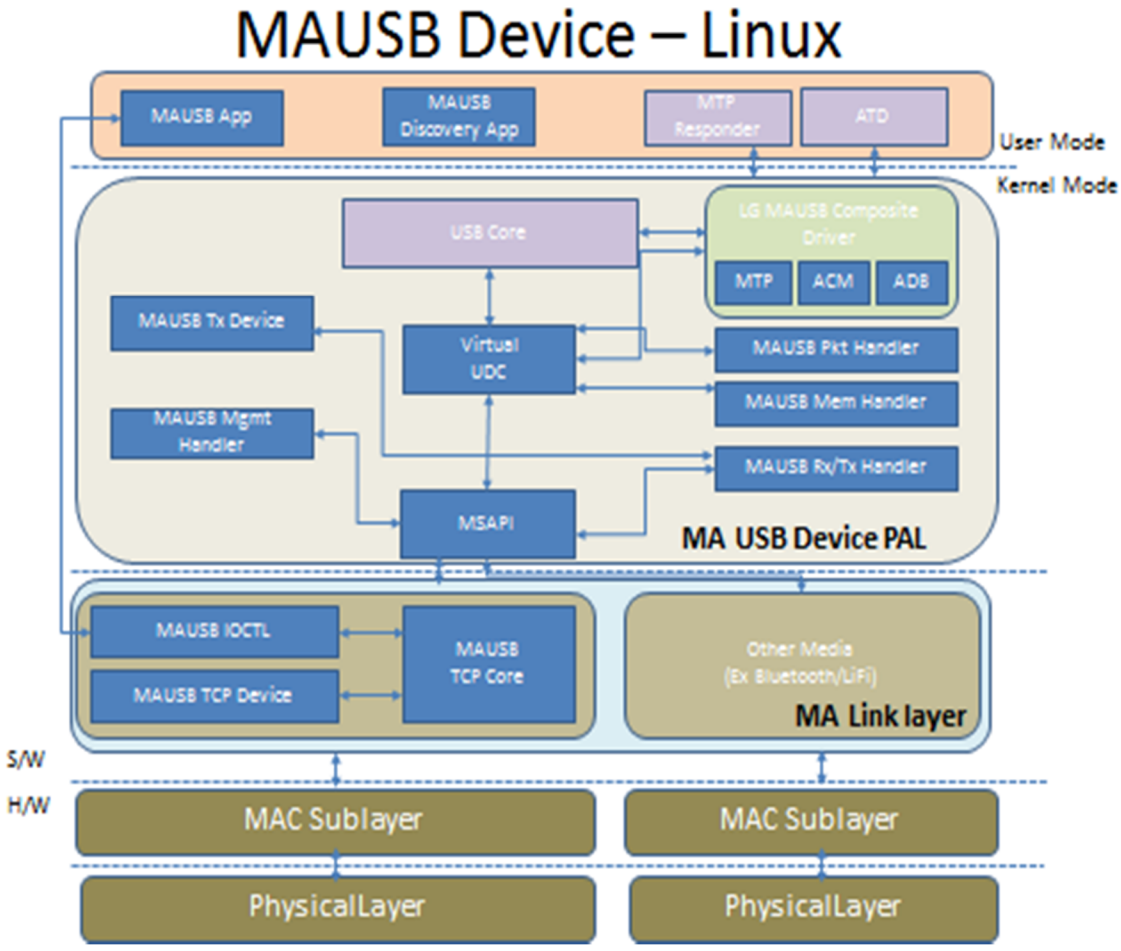

Figure 13: MAUSB Architecture

4.1 MAUSB Module Design

The Media Agnostic USB Driver contains the following modules in it

- MAUDC – Virtual USB Device Controller.
- MGMT – All Management packets are handled in this module.
- TRANSPORT – MAUSB RX thread is used to receive mausb packets from media.
- TRANSPORT – MAUSB TX thread is used to send mausb packets to media.
- MAUSB Packet handling
  - Management Packet handling
  - Control Packet Handling
  - Data Packet Handling
- MS Packet Handler

4.2 MAUSB Transfer types

MA USB transfers are broadly classified into Non-Isochronous and Isochronous transfers. Each of which in turn has OUT (Host->Device) and IN (Host<-Device) transfers.
IN transfers do not need flow control, as a receive buffer greater than the entire transfer size is assumed to be available on the host before the transfer is initiated. OUT transfers generally require flow control, as the receive buffer on the target MA USB device is not required to be as large as the transfer size.

The protocol-managed (p-managed) model uses protocol-level MA USB packets for flow control. Support of the p-managed transfer model is mandatory for all MA USB hosts and devices. The link-managed (l-managed) model offloads the flow control function to the underlying link protocol by carrying the transfer payload (targeting a single OUT endpoint) over a dedicated flow-controlled link-level connection.

4.3 MAUSB Protocol Structures

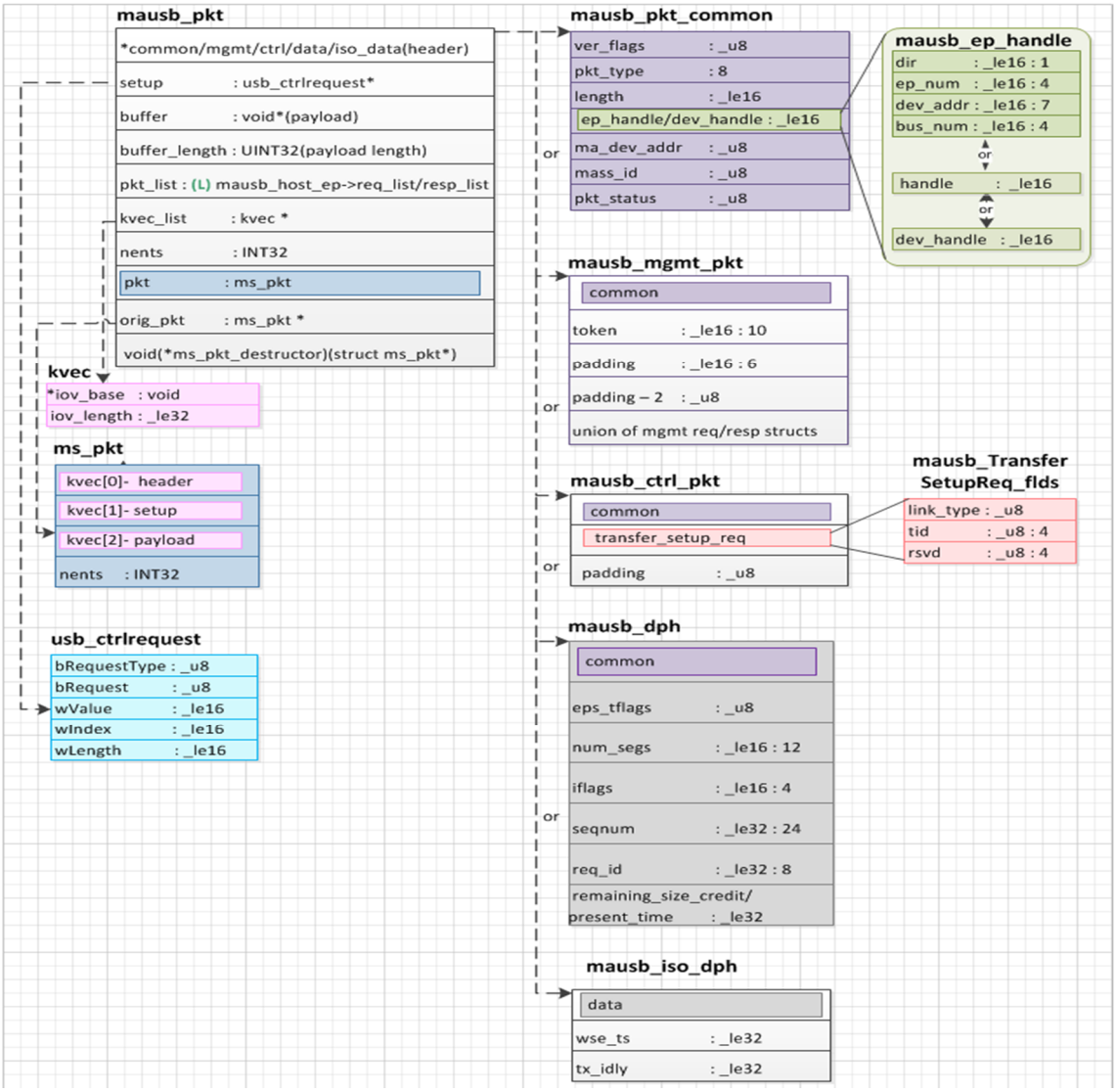

Figure 14: MAUSB Protocol Structures

4.4 MAUSB Device Enumeration

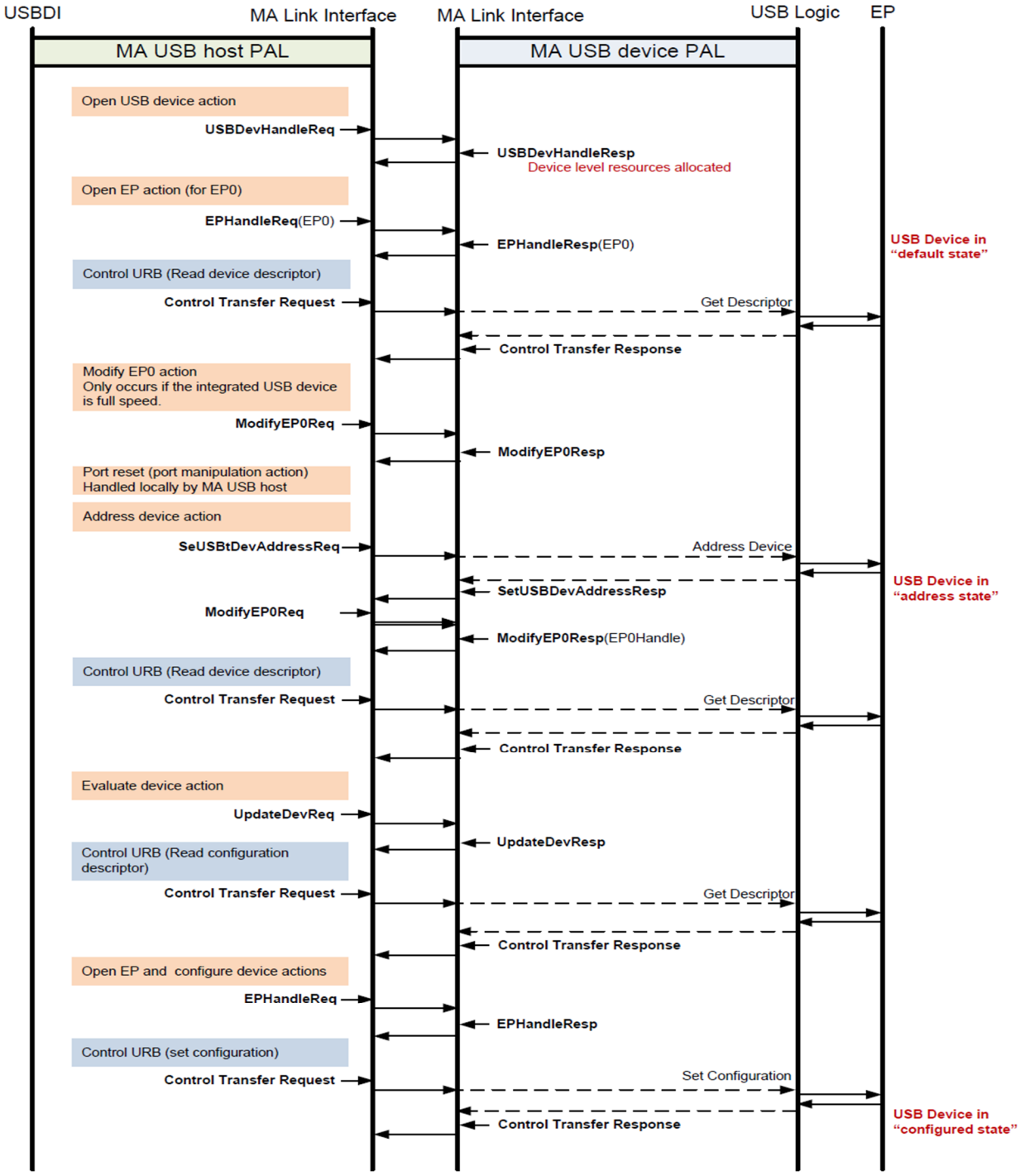

Figure 15: Enumeration of an integrated MAUSB device

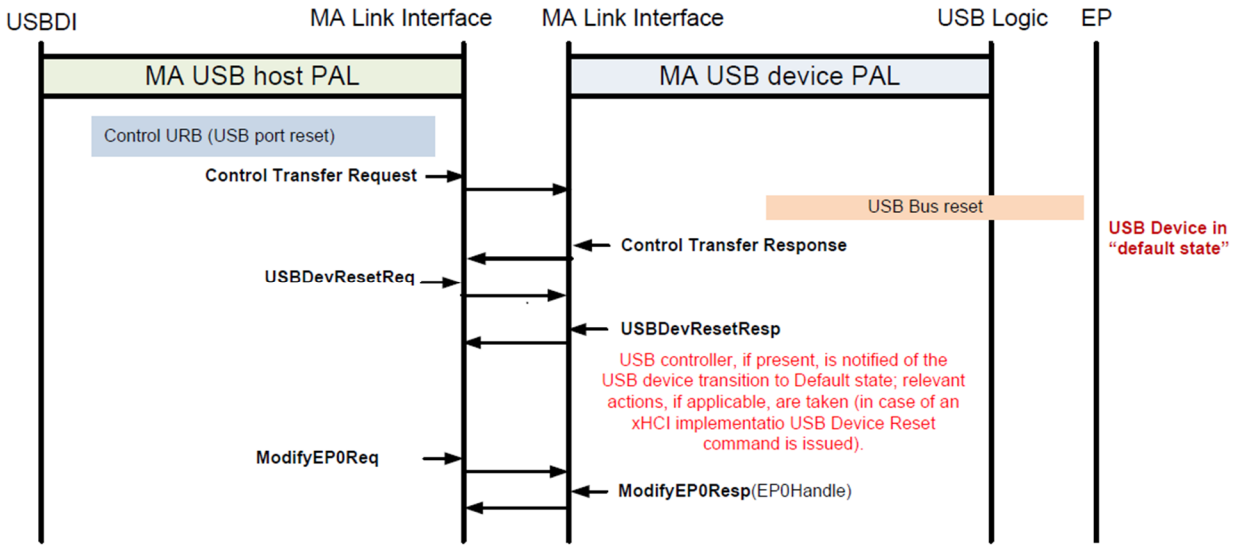

Figure 16: MAUSB device reset

4.5 MAUSB Control Transfer

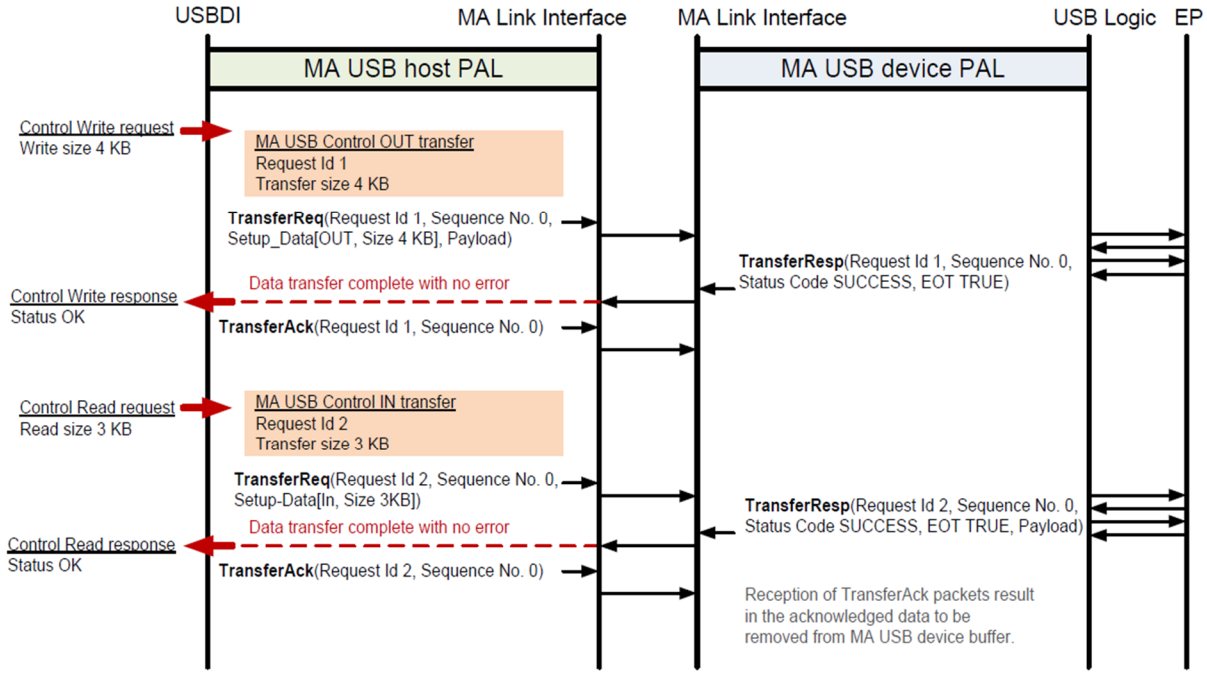

Figure 17: MAUSB Control Transfer

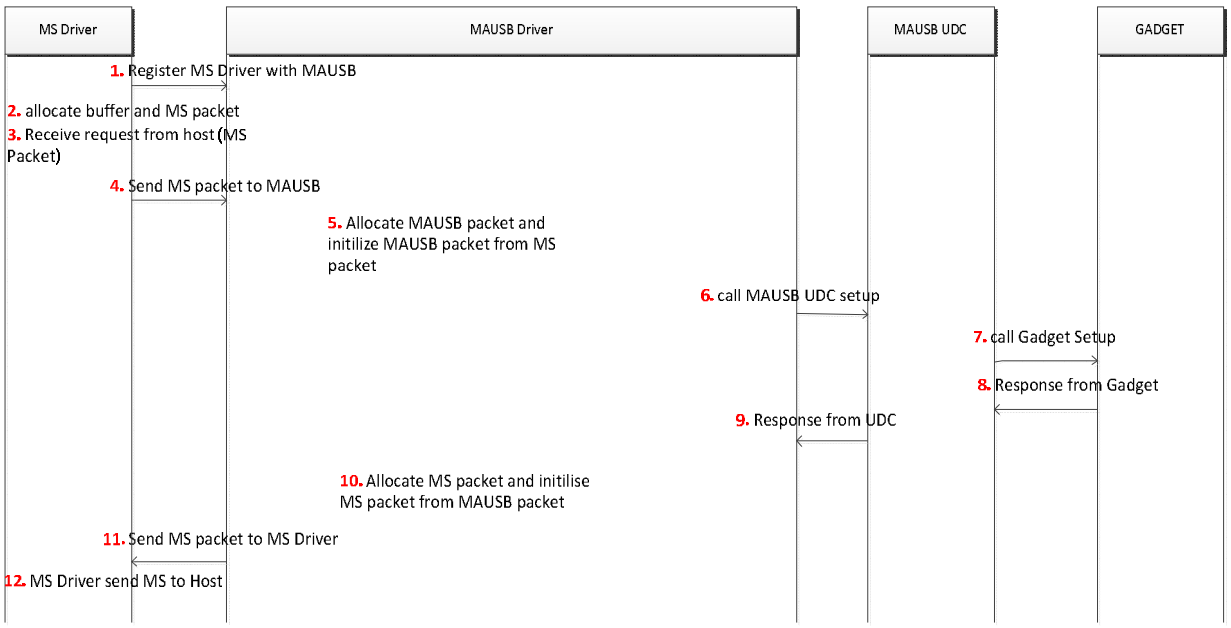

Figure 18: MAUSB Control Transfer Flow Diagram

4.6 MAUSB Management Transfer

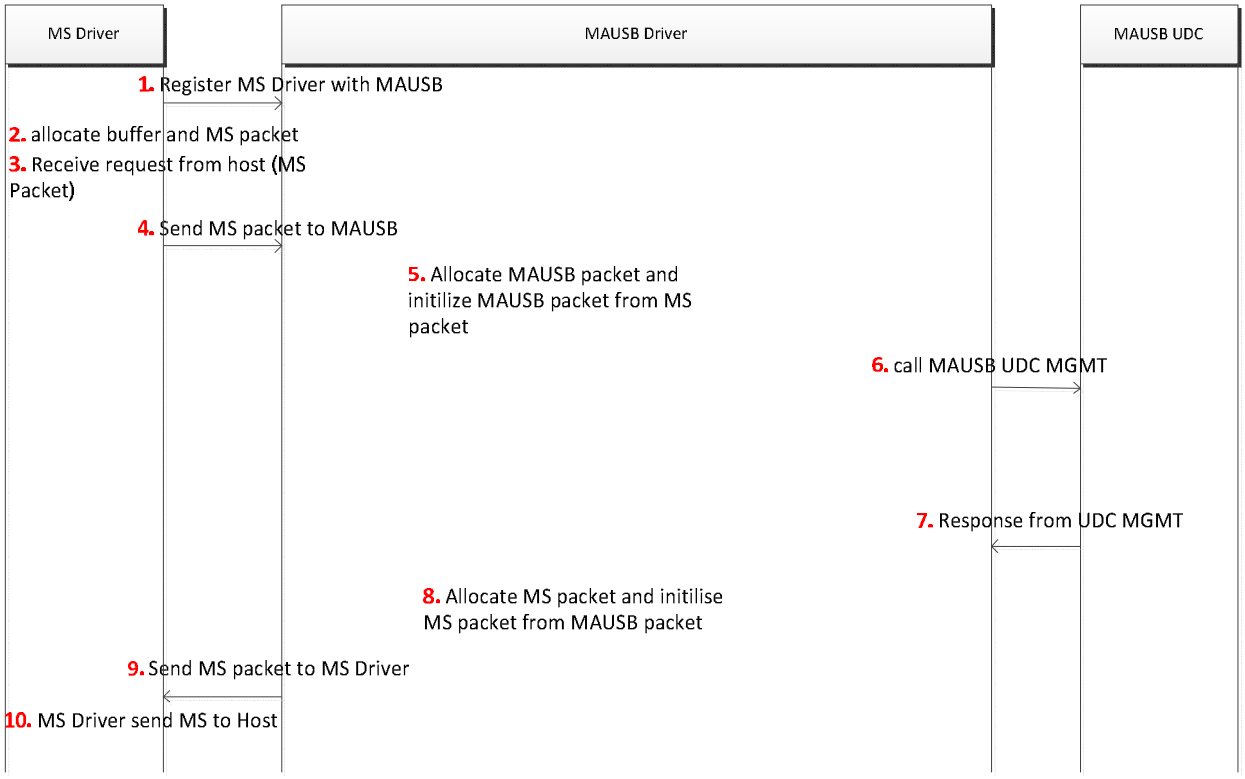

Figure 19: MAUSB Management Flow Diagram

4.7 MAUSB Out Transfer

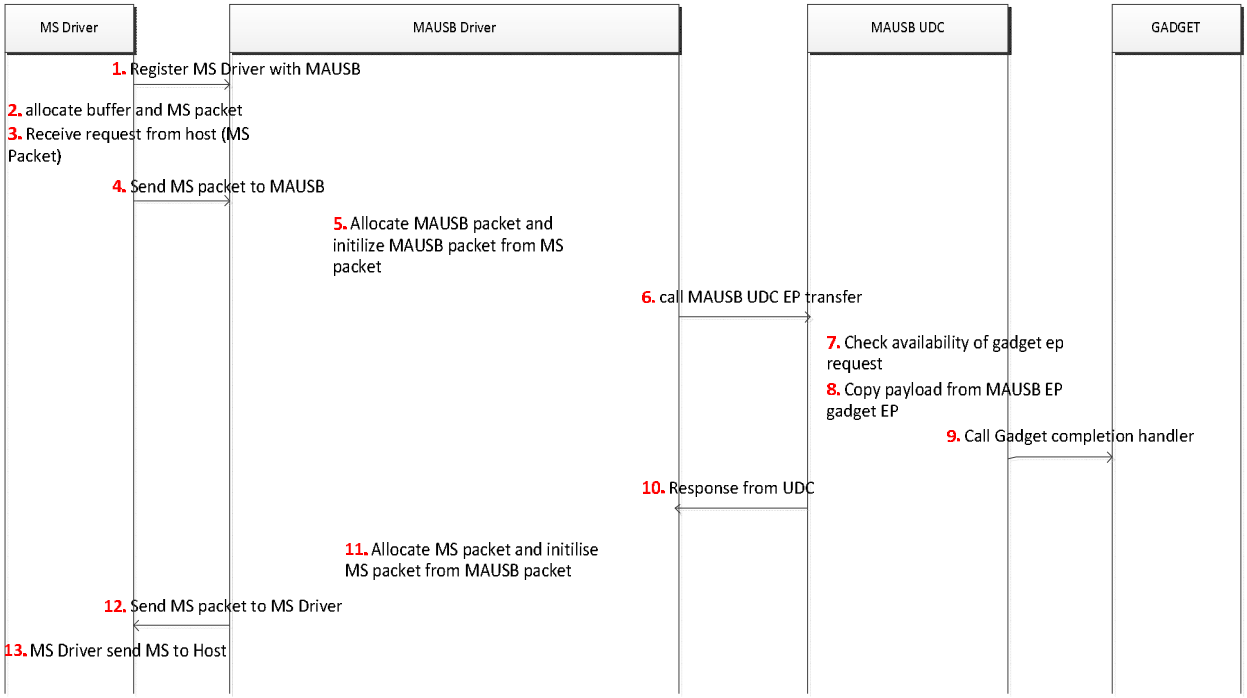

Figure 20: MAUSB OUT Transfer Flow Diagram

4.8 MAUSB In Transfer

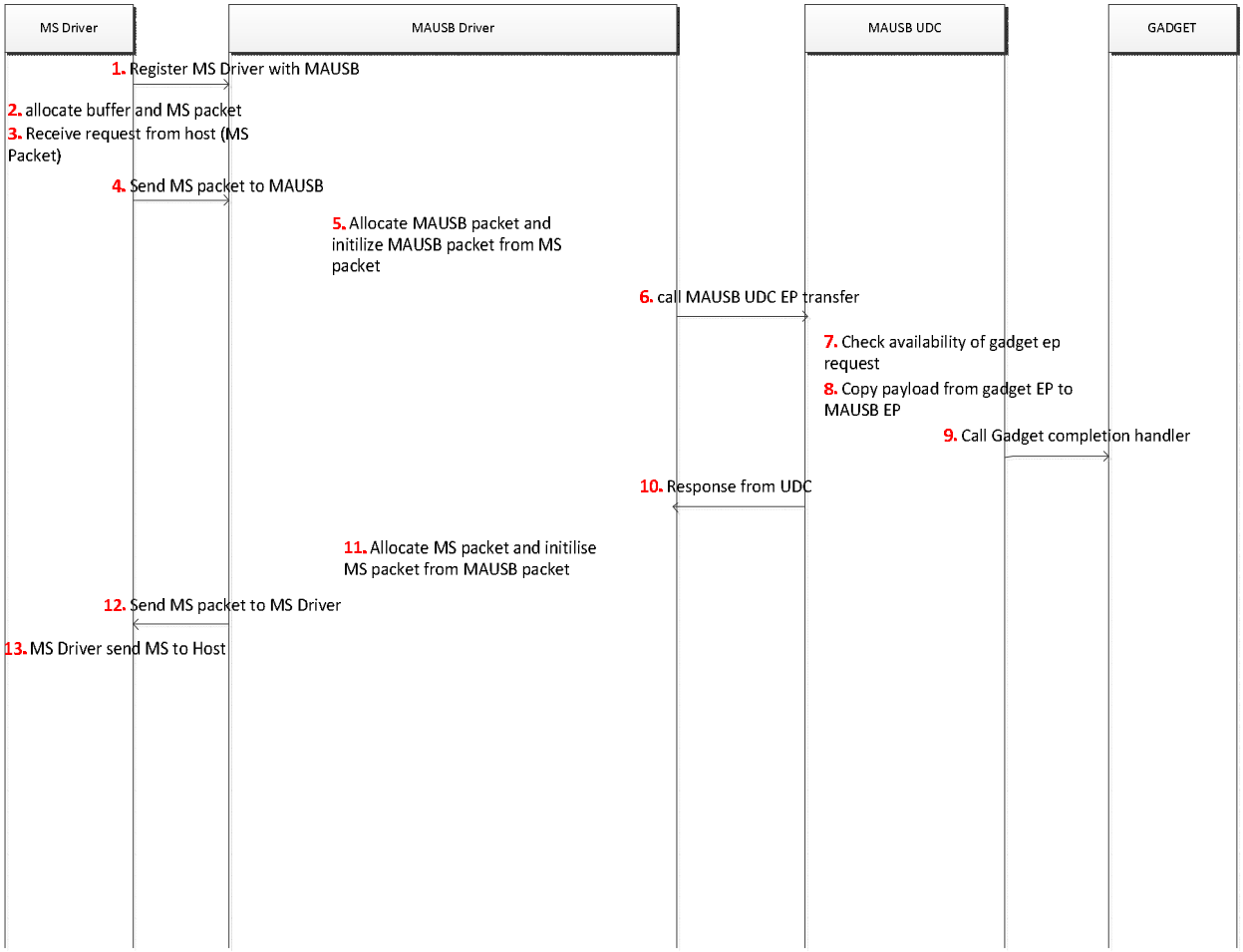

Figure 21: MAUSB IN Transfer Flow Diagram

5. MAUSB Details of Work done

Deep understanding of MAUSB specification to design MAUSB architecture. MAUSB architecture is divided into small modules.

a) Virtual USB Device Controller: Virtual UDC is core component in MAUSB architecture. Virtual UDC receives URB packets from USB core and submits URB packets to core.

b) MAUSB Management Packet Handler: All management commands are handled in this module.

c) MAUSB Transport: MAUSB Tx/Rx threads are created to receive and send data from media.

d) MAUSB control packet handling: In this routine we will handle USB control packets.

e) Data Packet handling

- MAUSB IN data packet handling: IN data packet means, data is transferred from device to host. All USB in packets are handled in this routine
- MAUSB OUT data packet handling: OUT data packet means, data is transferred from Host to Device. All USB out packets are handled in this routine.

6. Conclusion & Future of Work

This project forms a proof of concept to demonstrate the Media Agnostic Universal Serial Bus protocol. I have integrated MAUSB in wearable device. MAUSB can be integrated with other device like handset and Television. I have integrated ACM, DIAG and ADB gadget drivers with MAUSB. We can add support for other gadget drivers MTP.

Changes and improvements are possible with this MAUSB. In this PoC, I have used medium as IP. We can add other mediums like SNAP and ZigBee protocols. We can port this MAUSB in different devices like handsets and televisions.

7. Bibliography and References

[1] Daniel P. Bovet, Marco Cesati, Understanding the Linux Kernel, 3rd Edition, O'Reilly, 2005

[2] Wolfgang Maurer, Profession Linux Kernel Architecture, Wiley Publishing, 2008

[3] USB Forum, Media Agnostic Universal Serial Bus Specification, 2015.

[4] USB Forum, Universal Serial Bus 2.0 Specification, 2014.

[5] GregKroah-Hartman, How to Write a Linux USB Device Driver, Linux Journal, 2001.

Checklist of items for the Final Dissertation Report

This checklist is to be attached as the last page of the report.

This checklist is to be duly completed, verified and signed by the student.

1.	**Is the final report neatly formatted with all the elements required for a technical Report?**	Yes / No
2.	Is the Cover page in proper format as given in Annexure A?	Yes / No
3.	Is the Title page (Inner cover page) in proper format?	Yes / No
4.	(a) Is the Certificate from the Supervisor in proper format? (b) Has it been signed by the Supervisor?	Yes / No Yes / No
5.	Is the Abstract included in the report properly written within one page? Have the technical keywords been specified properly?	Yes / No Yes / No
6.	Is the title of your report appropriate? **The title should be adequately descriptive, precise and must reflect scope of the actual work done.** Uncommon abbreviations / Acronyms should not be used in the title	Yes / No
7.	Have you included the List of abbreviations / Acronyms?	Yes / No
8.	Does the Report contain a summary of the literature survey?	Yes / No
9.	Does the Table of Contents include page numbers? (i). Are the Pages numbered properly? (Ch. 1 should start on Page # 1) (ii). Are the Figures numbered properly? (Figure Numbers and Figure Titles should be at the bottom of the figures) (iii). Are the Tables numbered properly? (Table Numbers and Table Titles should be at the top of the tables) (iv). Are the Captions for the Figures and Tables proper? (v). Are the Appendices numbered properly? Are their titles appropriate	Yes / No Yes / No Yes / No Yes / No Yes / No Yes / No
10.	Is the conclusion of the Report based on discussion of the work?	Yes / No
11.	Are References or Bibliography given at the end of the Report? Have the References been cited properly inside the text of the Report? Are all the references cited in the body of the report	Yes / No Yes / No Yes / No
12.	Is the report format and content according to the guidelines? The report should not be a mere printout of a Power Point Presentation, or a user manual. Source code of software need not be included in the report.	Yes / No

Declaration by Student:

I certify that I have properly verified all the items in this checklist and ensure that the report is in proper format as specified in the course handout.

Place: Bangalore **Signature of the Student**

Date: 17/10/2016 **Name: Nagaraju Kadiri**

ID No.: 2014HT13190

	Supervisor	**Additional Examiner**

Name	Kishore PV	Suresh Kumar J
Qualification	B- Tech	MCA
Designation	Project Manager	Project Manager
Employing Organization & Location	LG Soft India Pvt Ltd, Bangalore.	LG Soft India Pvt Ltd, Bangalore.
Phone Number	+918066155211	+918066155081
Mobile Number	+919900578871	+918904088119
Email Address	Kishore.pv@lge.com	Suresh.Jabisetty@lge.com
Signature		
Place & Date	Bangalore & 17/10/2016	Bangalore & 17/10/2016