IrDA Control Specification (Formerly IrBus)

IrDA CIR (Control IR) Standard

Final Specification

Final Revision 1.0
June 30, 1998

INFRARED DATA ASSOCIATION (IrDA) - NOTICE TO THE TRADE -

SUMMARY:

Following is the notice of conditions and understandings upon which this document is made available to members and non-members of the Infrared Data Association.

- Availability of Publications, Updates and Notices
- Full Copyright Claims Must be Honored
- Disclaimer of Warranty
- Controlled Distribution Privileges for IrDA Members Only
- Trademarks of IrDA - Prohibitions and Authorized Use
- No Representation of Third Party Rights
- Limitation of Liability
- Product Testing for IrDA Specification Conformance

IrDA PUBLICATIONS and UPDATES:

IrDA publications, including notifications, updates, and revisions, are accessed electronically by IrDA members in good standing during the course of each year as a benefit of annual IrDA membership. Electronic copies are available to the public on the IrDA web site located at irda.org. Requests for publications, membership applications or more information should be addressed to: Infrared Data Association, P.O. Box 3883, Walnut Creek, California, U.S.A. 94598; or e-mail address: info@irda.org; or by calling (510) 943-6546 or faxing requests to (510) 934-5600.

COPYRIGHT:

1. Prohibitions: IrDA claims copyright in all IrDA publications. Any unauthorized reproduction, distribution, display or modification, in whole or in part, is strictly prohibited.

2. Authorized Use: Any authorized use of IrDA publications (in whole or in part) is under NONEXCLUSIVE USE LICENSE ONLY. No rights to sublicense, assign or transfer the license are granted and any attempt to do so is void.

DISCLAIMER of WARRANTY:

All IrDA publications are provided "AS IS" and without warranty of any kind. IrDA (and each of its members, wholly and collectively, hereinafter "IrDA") EXPRESSLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

IrDA DOES NOT WARRANT THAT ITS PUBLICATIONS WILL MEET YOUR REQUIREMENTS NOR THAT ANY USE OF A PUBLICATION WILL BE UN-INTERRUPTED OR ERROR FREE, OR THAT DEFECTS WILL BE CORRECTED. FURTHERMORE, IrDA DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING USE OR THE RESULTS OR THE USE OF IrDA PUBLICATIONS IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN PUBLICATION OR ADVICE OF A REPRESENTATIVE (OR MEMBER) OF IrDA SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY.

NOTICE: IrDA HAS BEEN NOTIFIED OF A THIRD PARTY CLAIM RELATED TO IMPLEMENTATION OF THE IRDA-CONTROL SPECIFICATIONS. IrDA IS PROVIDING THIS INFORMATION AS A NOTICE TO POTENTIAL IMPLEMENTERS OF THE SPECIFICATION AND MAKES NO REPRESENTATION AS TO THE VALIDITY OF THE CLAIM.

TRADEMARKS:

1. Prohibitions: IrDA claims exclusive rights in its trade names, trademarks, service marks, collective membership marks and feature trademark marks (hereinafter collectively "trademarks"), including but not limited to the following trademarks: INFRARED DATA ASSOCIATION (wordmark alone and with IR logo), IrDA (acronym mark alone and with IR logo), IR logo and MEMBER IrDA (wordmark alone and with IR logo). Any unauthorized use of IrDA trademarks is strictly prohibited.

2. Authorized Use: Any authorized use of a IrDA collective membership mark or feature trademark is by NONEXCLUSIVE USE LICENSE ONLY. No rights to sublicense, assign or transfer the license are granted and any attempt to do so is void.

NO REPRESENTATION of THIRD PARTY RIGHTS:

IrDA makes no representation or warranty whatsoever with regard to IrDA member or third party ownership, licensing or infringement/non-infringement of intellectual property rights. Each recipient of IrDA publications, whether or not an IrDA member, should seek the independent advice of legal counsel with regard to any possible violation of third party rights arising out of the use, attempted use, reproduction, distribution or public display of IrDA publications.

IrDA assumes no obligation or responsibility whatsoever to advise its members or non-members who receive or are about to receive IrDA publications of the chance of infringement or violation of any right of an IrDA member or third party arising out of the use, attempted use, reproduction, distribution or display of IrDA publications.

LIMITATION of LIABILITY:

BY ANY ACTUAL OR ATTEMPTED USE, REPRODUCTION, DISTRIBUTION OR PUBLIC DISPLAY OF ANY IrDA PUBLICATION, ANY PARTICIPANT IN SUCH REAL OR ATTEMPTED ACTS, WHETHER OR NOT A MEMBER OF IrDA, AGREES TO ASSUME ANY AND ALL RISK ASSOCIATED WITH SUCH ACTS, INCLUDING BUT NOT LIMITED TO LOST PROFITS, LOST SAVINGS, OR OTHER CONSEQUENTIAL, SPECIAL, INCIDENTAL OR PUNITIVE DAMAGES. IrDA SHALL HAVE NO LIABILITY WHATSOEVER FOR SUCH ACTS NOR FOR THE CONTENT, ACCURACY OR LEVEL OF ISSUE OF AN IrDA PUBLICATION.

LIMITED MEDIA WARRANTY:

IrDA warrants ONLY the media upon which any publication is recorded to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of distribution as evidenced by the distribution records of IrDA. IrDA's entire liability and recipient's exclusive remedy will be replacement of the media not meeting this limited warranty and which is returned to IrDA. IrDA shall have no responsibility to replace media damaged by accident, abuse or misapplication. ANY IMPLIED WARRANTIES ON THE MEDIA, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF DELIVERY. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM PLACE TO PLACE.

COMPLIANCE and GENERAL:

Membership in IrDA or use of IrDA publications does NOT constitute IrDA compliance. It is the sole responsibility of each manufacturer, whether or not an IrDA member, to obtain product compliance in accordance with IrDA Specifications.
IrDA is a trademark of the Infrared Data Association. All other product names are trademarks, registered trademarks, or servicemarks of their respective owners. HP is a registered trademark of Hewlett-Packard Corporation.

AUTHORS and CONTRIBUTORS:

Ray Chock, John Petrilla, Dick Crawford, Jason Hartlove (Hewlett-Packard Company)
Mohan Kumar, Jim Lansford, Steve Pawlowski (Intel Corporation)
Glade Bacon, Terry Lipscomb, Lord Nigel Featherston (Microsoft Corporation)
Hiroshi Uno, Masahiro Esashi, Fujiko Iai, Yoshihiro Ohtani, Katsuya Nakagawa,
Koji Sato, Yutaka Ikeda, Osamu Tsumori (SHARP Corporation)

Table of Contents

1 Introduction

1.1 Scope

IrDA Control technology is a command and control architecture for infrared communication that allows cordless peripherals, such as keyboards, mice, game pads, joysticks, and remote control units, to interact with many types of intelligent Host devices. This specification was developed under the project name "IrBus." Some software field codes or reference items may still retain references as part of the original "IrBus" project, and are included as part of the IrDA Control specification.

IrDA Control is applicable to a wide range of applications, including:

- Control of PC and HDTV systems
- Control of home appliances
- Communication between PC class devices and consumer appliances

This document describes the IrDA Control protocol specification for the following communication layers:

- PHY (Physical)
- MAC (Media Access Control)
- LLC (Logical Link Control)

This document also describes how to use IrDA Control with the communication scheme defined in the IrDA1.1 specification.

1.2 Definitions

The following definitions apply to this document:

Duty of Subcarrier Duty of Subcarrier is the proportion of the continuously IR emitting time in one subcarrier cycle to the whole time of one cycle of subcarrier.

Host IrDA Control compatible device that can poll IrDA Control peripherals.

Intensity Power per unit solid angle (milliwatts per steradian).

Irradiance Irradiance is power per unit area (microwatts per square centimeter).

Packet Time period from the start of the AGC field to the end of the STO field in an IrDA Control transmission.

Peak Wavelength Peak Wavelength (micrometers). Wavelength at which the optical output source intensity is a maximum.

Peripheral IrDA Control compatible device that is polled by IrDA Control Host

Link Length Link Length is the distance between the optical port surfaces of a host and peripheral.

Signaling Rate Signaling Rate (also called Bit Rate), (kilobits per second) is the rate at which information (data and protocol information) is sent or received.

Subcarrier Subcarrier is the alternating IR pulses whose modulation is used as the signal in data transfer.

1.3 Acronyms

The following acronyms apply to this document:

AGC Automatic Gain Control

BER Bit Error Rate or Bit Error Ratio

bps Bits per second

BIOS Basic Input/Output System

CL Critical Latency

CRC Cyclic Redundancy Check

CRC-8 8 bit CRC based on the polynomial $x^8 + x^7 + x^2 + 1$

CRC-16 16 bit CRC based on the polynomial $x^{16} + x^{15} + x^2 + 1$

DBS	Data Bit Set
HA	Home Appliance
HADD	Host Address
HDTV	High Definition TeleVision
HID	Human Interface Device
IR	Infrared
IRB-TM	Infrared Bus Transceiver Module
IRED	Infrared Emitting Diode
IrDA	Infrared Data Association
IrLAP	Infrared Link Access Protocol
Kbps	Kilo bits per second
LLC	Logical Link Control
LSB	Least Significant Bit
MAC	Media Access Control layer
MSB	Most Significant Bit
NCL	Non-Critical Latency
NRZ	Non Return to Zero code
PADD	Peripheral Address
PDA	Personal Digital Assistant
PFID	Peripheral physical IDentifier
PHY	Physical Layer
PRE	IrDA Control Preamble field
Pin-PD	Pin PhotoDiode
PSM	Pulse Sequence Modulation
SEPC	Subcarrier Emission Pulse Chip
SEPD	Subcarrier Emission Pulse Duration
SIR	IrDA Serial Infrared standard, 115.2kbit/s (1.0)
STA	IrDA Control Start flag
STL	IrDA Control Start flag (Long Packet)
STO	IrDA Control Stop flag
STS	IrDA Control Start flag (Short Packet)
USB	Universal Serial Bus

1.4 References

The following references are recommended for additional reading:

1. **Transmission of audio and/or video and related signals using infra-red radiation**, IEC 1603-1, International Electrotechnical Commission.
2. **Infrared Data Association Serial Infrared Physical layer Link Specification Version 1.1.**
3. **Infrared Data Association Serial Infrared Link Access Protocol (IrLAP) Version 1.1.**
4. **Universal Serial Bus Specification Version 1.0 January 19, 1996**
5. **USB Device Class Definition for Human Interface Devices (HID) Version 1.0 Draft #4**

2 Overview

The overview of the IrDA Control protocol stack is shown in Figure. 2.1.

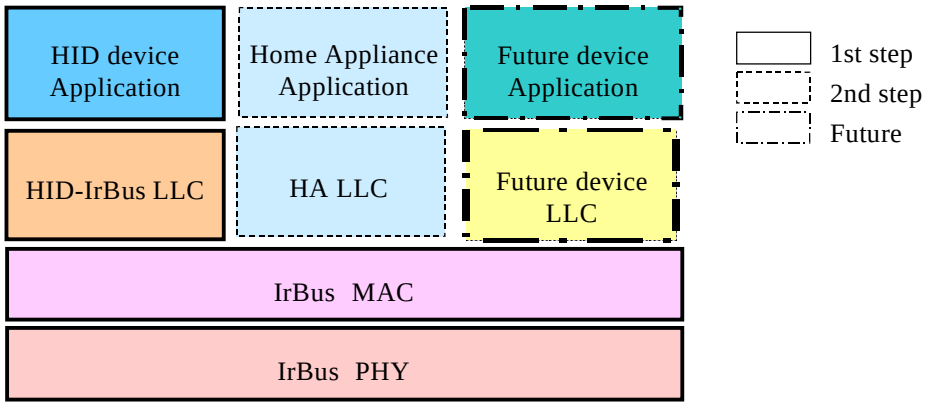


Figure 2.1 Protocol stack of IrBus

The PHY layer defines the physical characteristics and modulation scheme of the infrared signals. The MAC layer defines the media access scheme for infrared wireless data communications. The IrDA Control LLC provides the Link Layer resources for reliable communication of data to and from the MAC layer.

There are four possible application layer protocols encoded in 2-bits of the HostID field of the LLC frames. The two protocols that are currently defined are the HA protocol and the USB HID protocol. The HA protocol is intended for operating home appliances. The USB HID protocol is intended for computer input devices.

The IrDA Control protocol includes the following features:

PHY layer:

- Distance and range equivalent to that of current uni-directional infrared remote control units.
- Bi-directional communication.
- Data transmission rate of 75.0 kbps.
- Optimized for low power usage.
- Can be implemented with low-cost hardware.

MAC layer:

- Enables a device to communicate with multiple devices (1: n communications).
- Ensures fast response time (e.g.13.8 ms basic polling cycle)

LLC layer:

- Provides the basic facility to improve the reception and transmission of data to a higher level than the basic BER of the IR link.
- HID-IrDA Control Bridge enables the link control function of USB-HID. The USB-HID bridge is described in the appendix.

3 Physical Layer (PHY)

3.1 Overview

A block diagram of the hardware implementation for the IrDA Control system is given in Figure 3.1.

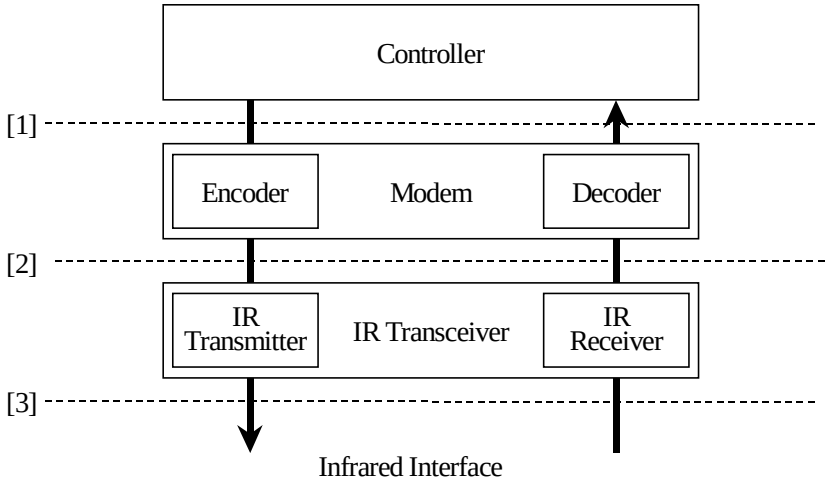
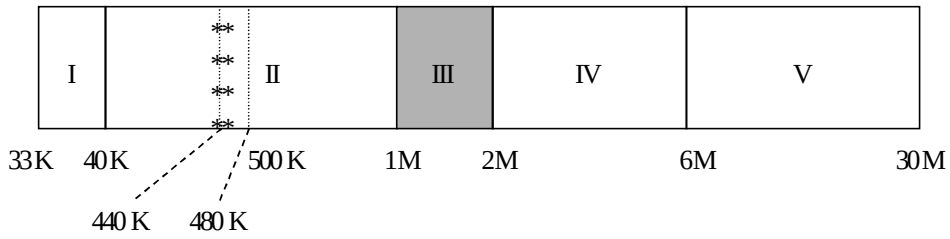


Figure 3.1 : Hardware implementation example of IRBus

The signals at [1] are normally a serial bit stream. Those in [2] are the modulated electrical signals of the above bit stream and those in [3] are the corresponding optical signals. Various schemes can be used to implement these parts of the IrDA Control system. This specification defines the transmission speeds, modulation schemes, infrared wavelengths etc. of the optical signals emitted by the infrared transmitter and of those coming into the infrared receiver in [3]. It does not mandate the signals in [1] and [2], or the signals in the infrared transmitter/receiver or the modulator/demodulator. More specifically, this specification does not define the voltage waveform of the driver circuit that drives the LED of the infrared transmitter or the current waveform after the photoelectric conversion done by the Pin-PD of the infrared receiver.

The IrDA Control system operates at a transmission speed of 75.0 kbps. The data to be transmitted is coded by the 16-Pulse Sequence Modulation (16PSM) scheme, as detailed in Section 3.2.1, multiplied by a subcarrier and then output by the infrared transmitter. A frequency of 1.5 MHz is used for the subcarrier. As shown in Figure 3.2, this frequency conforms to the frequency allocation specified in IEC1603-1 (Reference 1). The 16PSM scheme is able to reduce the interference between an IrDA Control system and a Remote Control System that uses frequencies in the 33kHz - 40kHz band (See Appendix A).



Band I	(33kHz - 40kHz)	:	Low-speed remote control
Band II	(45kHz - 1MHz)	:	Audio wideband and related signals transmission for conference and similar systems Audio transmission for conference and similar systems Low-speed remote control systems
Band III	(1MHz - 2MHz)	:	High-speed remote control and related data systems
Band IV	(2MHz - 6MHz)	:	Audio wideband and related signals transmission systems
Band V	(6MHz - 30MHz)	:	Video and audiovisual signal transmission systems of high quality

Note The * in II band indicates that channel C11(440kHz to 480kHz) has to be kept free for the i.f. in AM broadcast radio and some IR receivers.

Figure 3.2 : IEC1603-1 subcarrier frequency allocation

3.2 Coding

3.2.1 16PSM Data Coding Definition

The IrDA Control system uses the 16-Pulse Sequence Modulation (16PSM) scheme for data coding. In this scheme, a time defined as “symbol time (Dt) ” is equally divided into eight slots defined as “chips”, and a pulse is allowed only during two or four of those chip periods. Information is transmitted according to the pulse pattern of the sequence. The waveforms that have legal pulse sequences are defined as 16PSM Data Symbols, or simply as Symbols.

Each chip time (Ct) is given by the following equation:

$$Ct = Dt / 8$$

In the 16PSM scheme, four bits of information can be transmitted within a single symbol time. Accordingly, there are 16 waveforms defined as 16PSM Data Symbols. Each unique set of four bits corresponds to one of 16 symbol values, and is defined as a Data Bit Set (DBS). The DBSs associated with the 16 kinds of symbols are shown in Table 3.1.

Data Value (Hex)	Data Bit Set (DBS)	16PSM Data Symbol
0x0	0 0 0 0	1 0 1 0 0 0 0 0
0x1	0 0 0 1	0 1 0 1 0 0 0 0
0x2	0 0 1 0	0 0 1 0 1 0 0 0
0x3	0 0 1 1	0 0 0 1 0 1 0 0
0x4	0 1 0 0	0 0 0 0 1 0 1 0
0x5	0 1 0 1	0 0 0 0 0 1 0 1
0x6	0 1 1 0	1 0 0 0 0 0 1 0
0x7	0 1 1 1	0 1 0 0 0 0 0 1
0x8	1 0 0 0	1 1 1 1 0 0 0 0
0x9	1 0 0 1	0 1 1 1 1 0 0 0
0xA	1 0 1 0	0 0 1 1 1 1 0 0
0xB	1 0 1 1	0 0 0 1 1 1 1 0
0xC	1 1 0 0	0 0 0 0 1 1 1 1
0xD	1 1 0 1	1 0 0 0 0 1 1 1
0xE	1 1 1 0	1 0 1 0 0 1 0 1
0xF	1 1 1 1	1 1 1 0 0 0 0 1

Table 3.1: 16PSM Data Symbol Representation

Since the transmission speed in the IrDA Control system is 75.0 kbps, symbol time Dt and chip time Ct are given with the following equations:

Dt = 4 * (1 / 75000)
= 53.33 [μs]
Ct = 53.33E-6 / 8
= 6.67 [μs]

The 16PSM scheme, using this specific symbols set, has low energy in the frequency band of around 33kHz to 40kHz, which is the frequency band used for Remote Control Systems (see Figure 3.2). The spectrum characteristics of the IrDA Control signal based on the 16PSM scheme are shown in Appendix A.

3.2.2 PHY Packet Structure

The IrDA Control packet format is shown in Figure 3.3. There are two packet formats in the IrDA Control system; short packets and long packets. Each packet consists of six fields: the Automatic Gain Control (AGC); Preamble (PRE); Start Flag (STA or STL); MAC frame; Cyclic Redundancy Check (CRC which can be CRC-8 or CRC-16); and Stop Flag (STO) fields.

AGC	PRE	STA = STS	MAC frame	CRC = CRC – 8	STO
(2 bit times)	(5 bit times)	(5 bit times)		(8 bits)	(4 bit times)

(a) Short Packet

AGC	PRE	STA = STL	MAC frame	CRC = CRC–16	STO
(2 bit times)	(5 bit times)	(5 bit times)		(16 bits)	(4 bit times)

(b) Long Packet

Figure 3.3: IrDA Control Packet Format

The pulse sequence for each of AGC, PRE, STS, STL, and STO fields is shown in Table3.2. The chips are transmitted starting with the leftmost bit in each field; time moves from left to right.

Field Name	Pulse Sequence
AGC	1 1 1 1
PRE	0 1 0 1 0 1 0 1 0 1
STS	0 1 1 0 1 1 0 1 0 0
STL	0 1 0 0 1 0 1 1 0 1
STO	0 1 0 0 1 0 1 1

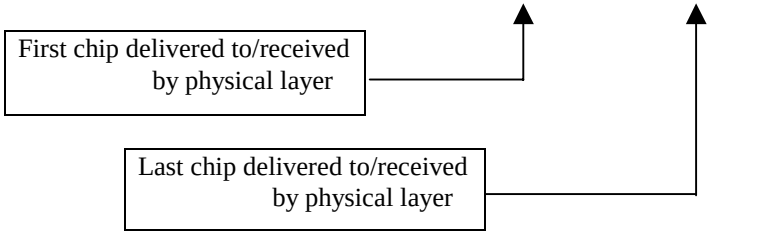


Table 3.2: Pulse Sequence Representation for AGC, PRE, STS, STL, and STO

For a short packet, STS/CRC-8 is used as STA/CRC. For a long packet, STL/CRC-16 is used as STA/CRC. The details of the CRC field will be described in Section 3.2.3.

The AGC field is used to control the gain and/or threshold level of the infrared receiver. The PRE field is used to get chip (clock) synchronization. The STA field is used to get symbol synchronization, and to indicate a short (STS) or long (STL) packet. The STO field is used to indicate the end of the packet.

As shown in Section 3.2.1, 16PSM has the regularity that a chip of logical "1" appears only twice or four times during the eight possible chip slots and that the combination of logical "0" and "1" in a symbol is restricted to 16 patterns shown in Table.3.1. Since the AGC, PRE, STA, and STO fields use pulse sequences that violate this regularity, there is no need for a bit stuffing mechanism in the 16 PSM coding scheme. This modulation scheme allows the time for a packet to be transmitted to be determined without dependence on the data contents. This is an advantage of 16 PSM coding in comparison with some other coding schemes. It can also be an advantage in implementing critical latency applications.

3.2.3 CRC Field

The CRC field is 8 or 16 bits in length for a short packet or a long packet, respectively. It contains a value computed over the entire MAC frame field.

A CRC-8 consists of the 16 PSM encoded data resulting from the CRC-8 algorithm for cyclic redundancy check as applied to the data contained in the packet. The CRC-8 polynomial is $x^8 + x^7 + x^2 + 1$. The data bytes are input to this

calculation in LSB first format. The CRC-8 calculated result for each packet is treated as one data byte, and the byte is encoded in the same manner as the data bytes in MAC frame.

A CRC-16 consists of the 16PSM encoded data resulting from the CRC-16 algorithm for cyclic redundancy check as applied to the data contained in the packet. The CRC-16 polynomial is $x^{16} + x^{15} + x^2 + 1$. The data bytes are input to this calculation in LSB first format. The CRC16 calculated result for each packet is treated as two data bytes, and each byte is encoded in the same manner as the data bytes in MAC frame.

The CRC register is preset to all "1's" prior to calculation of the CRC on the transmit data stream. When data has ended and the CRC is being shifted for transmission at the end of the packet, a "0" should be shifted in so that the CRC register becomes a virtual shift register. Note: the inverse of the CRC register is what is shifted as defined in the polynomial.

3.2.4 Procedure for Transmitting Data Packet

The procedure for creating a data transmission packet is described below (Figure 3.4):

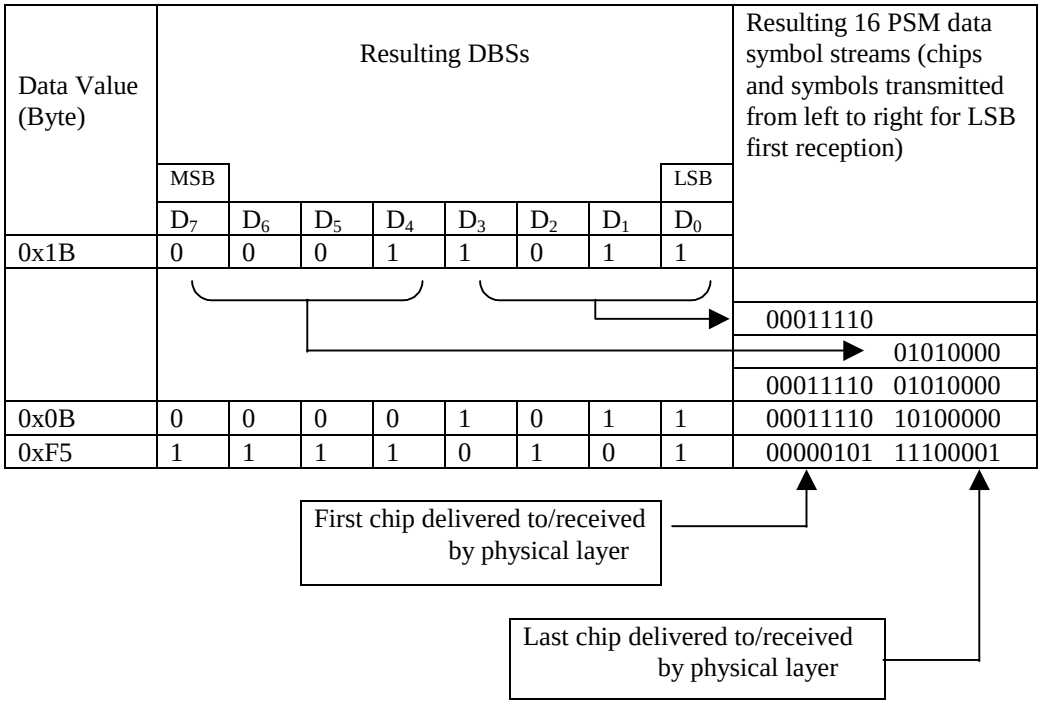


Figure 3.4: Example of packet generation procedure

(1) Each data byte [$D_7 – D_0$] in the original data which is included in the MAC frame and CRC fields is divided into two Data Bit Sets (DBSs), [$D_7 – D_4$] and [$D_3 – D_0$]. The DBSs are converted into the corresponding 16PSM Data Symbols in the order of [$D_3 – D_0$] and [$D_7 – D_4$] in Table 3.1.

(2) The AGC, PRE and STA fields are added to the head of the resulting stream of 16PSM data symbols in the above step and the STO to the tail of that stream.

3.3 Modulation

Corresponding to logic “0” or “1” in the packet generated by the procedure in Section 3.2.4, the infrared transmitter outputs the subcarrier signal for a chip of logic “1”, and outputs no signal for a chip of logic “0”. The relation between the data to be transmitted and the waveform output by the infrared transmitter is shown in Figure 3.5.

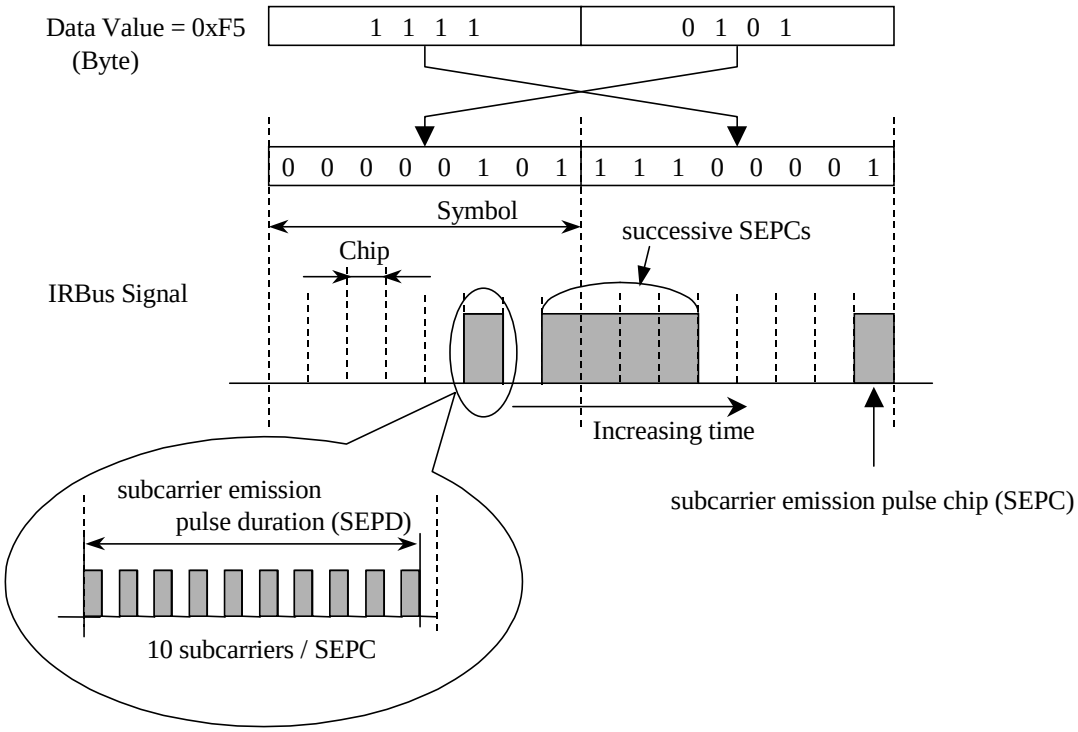


Figure 3.5 : Waveform Example of IRBus Signal

The chip for which the infrared transmitter outputs the subcarrier signal is defined as a subcarrier emission pulse chip (SEPC). For example, when the original data is “0xF5”, four successive SEPCs appear over two adjacent symbols, and the infrared transmitter continuously outputs the subcarrier for the duration of the four chips (fourfold chips), as shown in Figure 3.5. The duration between the rising edge of the initial subcarrier and the falling edge of the final subcarrier in the successive SEPCs is defined as subcarrier emission pulse duration (SEPD). Time is measured at the 50% point of the subcarrier amplitude.

The number of subcarriers included in each SEPD of Single, Double, Triple, Fourfold, Fivefold, Sixfold, Sevenfold and Eightfold Chips is 10, 20, 30, 40, 50, 60, 70 and 80, respectively. The infrared transmitter must synchronize the subcarrier with the SEPCs such that the number of subcarrier pulses is entirely included in each SEPD.

3.4 Overall Link Specifications

The specifications of the overall link are shown in Table 3.3.

Specification	Minimum	Typical	Maximum	Unit
Maximum Link Length				
Between Host and Peripheral Type1 Devices Host Devices (θ_H<+/-30deg and ϕ_H=0) Peripheral Type 1 Devices (θ_P<+/-15deg and ϕ_P=0) *	5.00			m
Between Host and Peripheral Type1 Devices Host Devices (θ_H<+/-50deg and ϕ_H=0) Peripheral Type 1 Devices (θ_P<+/-15deg and ϕ_P=0) *	3.00			m
Between Host and Peripheral Type1 Devices Host Devices (θ_H=0 and ϕ_H<+/-15deg) Peripheral Type 1 Devices (θ_P=0 and ϕ_P<+/-15deg) *	5.00			m
Between Host and Peripheral Type2 Devices Host Devices (θ_H<+/-40deg and ϕ_H=0) Peripheral Type2 Devices (θ_P<+/-40deg and ϕ_p=0) *	1.50			m
Between Host and Peripheral Type2 Devices Host Devices (θ_H=0 and ϕ_H<+/-25deg) Peripheral Type 2 Devices (θ_P=0 and ϕ_P<+/-25deg) *	1.50			m
Minimum Link Length				
Between Host and Peripheral Type1 Devices Host Devices (θ_H<+/-50deg and ϕ_H=0) Peripheral Type 1 Devices (θ_P<+/-15deg and ϕ_P=0) *			0.20	m
Between Host and Peripheral Type1 Devices Host Devices (θ_H=0 and ϕ_H<+/-15deg) Peripheral Type 1 Devices (θ_P=0 and ϕ_P<+/-15deg) *			0.20	m
Between Host and Peripheral Type2 Devices Host Devices (θ_H<+/-40deg and ϕ_H=0) Peripheral Type2 Devices (θ_P<+/-40deg and ϕ_P=0) *			0.20	m
Between Host and Peripheral Type2 Devices Host Devices (θ_H=0 and ϕ_H<+/-25deg) Peripheral Type 2 Devices (θ_P=0 and ϕ_P<+/-25deg) *			0.20	m
Peak Wavelength (All Devices)	850		900	nm
Duty of Subcarrier (All Devices)	47.5	50.0	52.5	%
Subcarrier Emission Pulse Duration (All Devices)				
(for Single Chip)	6.23	6.33	6.44	µs
(for Double Chip)	12.82	13.00	13.18	µs
(for Triple Chip)	19.42	19.67	19.92	µs
(for Fourfold Chip)	26.01	26.33	26.66	µs
(for Fivefold Chip)	32.60	33.00	33.40	µs
(for Sixfold Chip)	39.20	39.67	40.14	µs
(for Sevenfold Chip)	45.79	46.33	46.89	µs
(for Eightfold Chip)	52.39	53.00	53.63	µs
Rise Time Tr,10-90%,Fall Time Tf,90-10% (All Devices)			80	ns
Optical Over Shoot (All Devices)			25	%
Edge Jitter (Peak to Peak) (All Devices)			20	ns

*: See Figure 3.6

Table 3.3: Overall link Specifications

The required link length and directivity angle can vary depending on applications. For example, the link length that is required for a remote control unit is different from that for a desktop mouse. In the IrDA Control specification two types of peripheral devices for long and short distances are defined.

An IrDA Control system must satisfy a communication quality of BER=10^{-4} within the range of the following link length under a 20% IrDA sunlight ambient (=100μW/cm^2).

Each angle in Table 3.3 is illustrated in Figure 3.6. θ_H is the horizontal angle between the host's transceiver's plane normal and the line that includes both of the transceivers of the host and the peripheral. θ_P is the horizontal angle between the peripheral's transceiver's plane normal and the line that includes both of the transceivers of the host and the peripheral. ϕ_H is the vertical angle between the host's transceiver's plane normal and the line that includes both of the transceivers of the host and the peripheral. ϕ_P is the vertical angle between the peripheral's transceiver's plane normal and the line that includes both of the transceivers of the host and the peripheral.

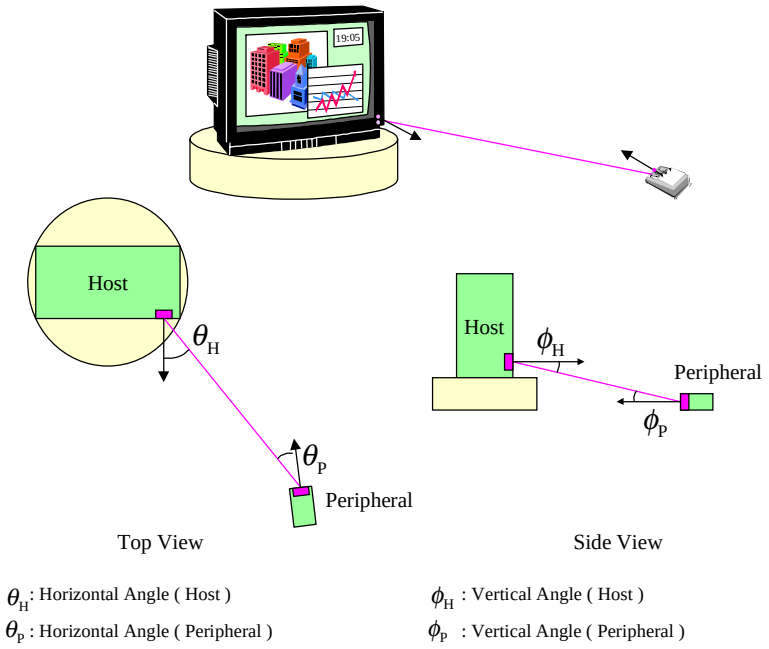


Figure 3.6: Definitions of Angles

The peak wavelength specification of the infrared radiation shall be the same as that in the IrDA 1.1 system (Reference 2).

As shown in Figure 3.7, Leading Edge Jitter is the deviation between time positions of the actual leading edge and the expected leading edge of the first subcarrier in the successive SEPCs. The expected leading edge time can be calculated from the average chip period within the preceding 16 bit times. Peak to peak Leading Edge Jitter is the difference between the maximum and minimum occurrences. The leading edge time position is measured at the 50% point of the amplitude of the first subcarrier in the successive SEPCs.

Trailing Edge Jitter is the deviation between time positions of the actual trailing edge and expected trailing edge of the last subcarrier in the successive SEPCs. The expected trailing edge time can be calculated from the average chip period within the preceding 16 bit times. Peak to peak Trailing Edge Jitter is the difference between the maximum and minimum occurrences. The trailing edge time position is measured at the 50% point of the amplitude of the last subcarrier in the successive SEPCs.

Edge Jitter is defined as the larger one of Leading Edge Jitter and Trailing Edge Jitter.

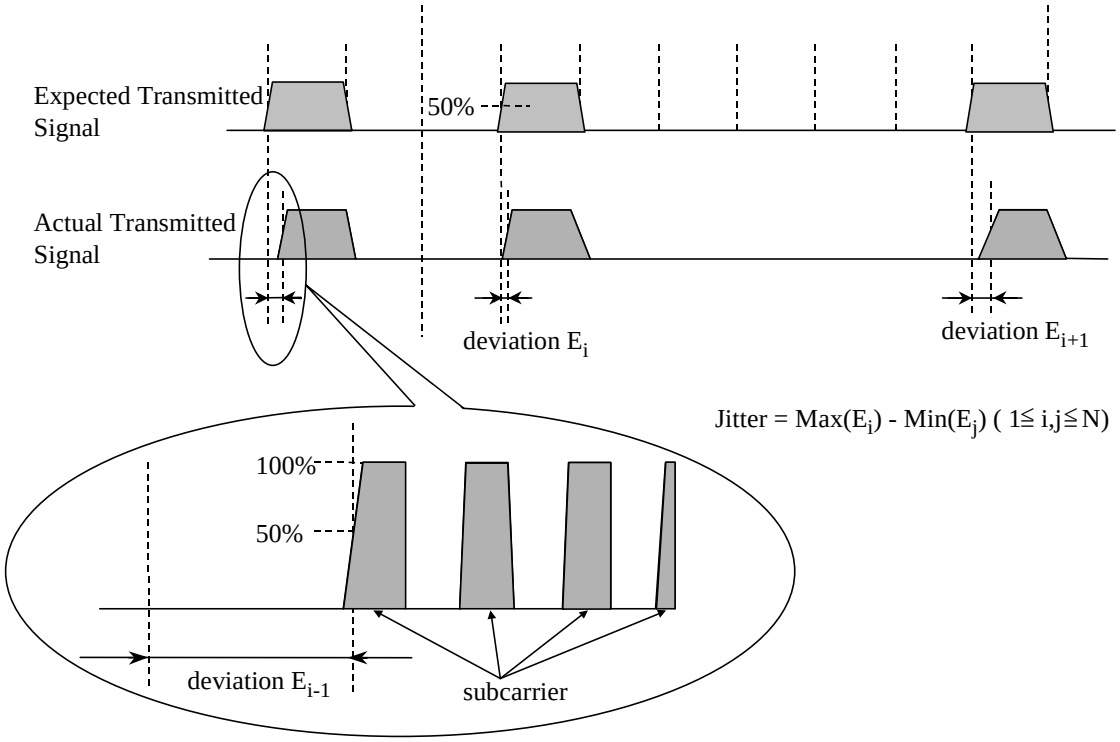


Figure 3.7 : Definition of Jitter

The subcarrier emission pulse duration (SEPD) is defined in Section 3.3. The minimum value of SEPD is given by the following equation, using the maximum subcarrier frequency specified in Section 3.5 and 3.6, the number of subcarriers included in each SEPD, the minimum duty of subcarrier, and the maximum edge jitter:

$$SEPD_{\min.} = \frac{Ns - (1 - Ds_{\min.})}{fs_{\max.}} - Je_{\max.}$$

Ns : number of subcarriers included in each SEPD
Ds : duty of subcarrier
fs : subcarrier frequency
Je : edge jitter

The maximum value of SEPD is given by the following equation, using the minimum subcarrier frequency specified in Section 3.5 and 3.6, the number of subcarriers included in each SEPD, the maximum duty of subcarrier and the maximum edge jitter:

$$SEPD_{\max.} = \frac{Ns - (1 - Ds_{\max.})}{fs_{\min.}} + Je_{\max.}$$

Ns : number of subcarriers included in each SEPD
Ds : duty of subcarrier
fs : subcarrier frequency
Je : edge jitter

Rise Time T_r, Fall Time T_f and Overshoot can be defined for a single subcarrier pulse. Since overshoot is referenced to the pulse amplitude at the end of the subcarrier pulse. For Rise Time T_r, Fall Time T_f and Overshoot is given in Figure 3.8.

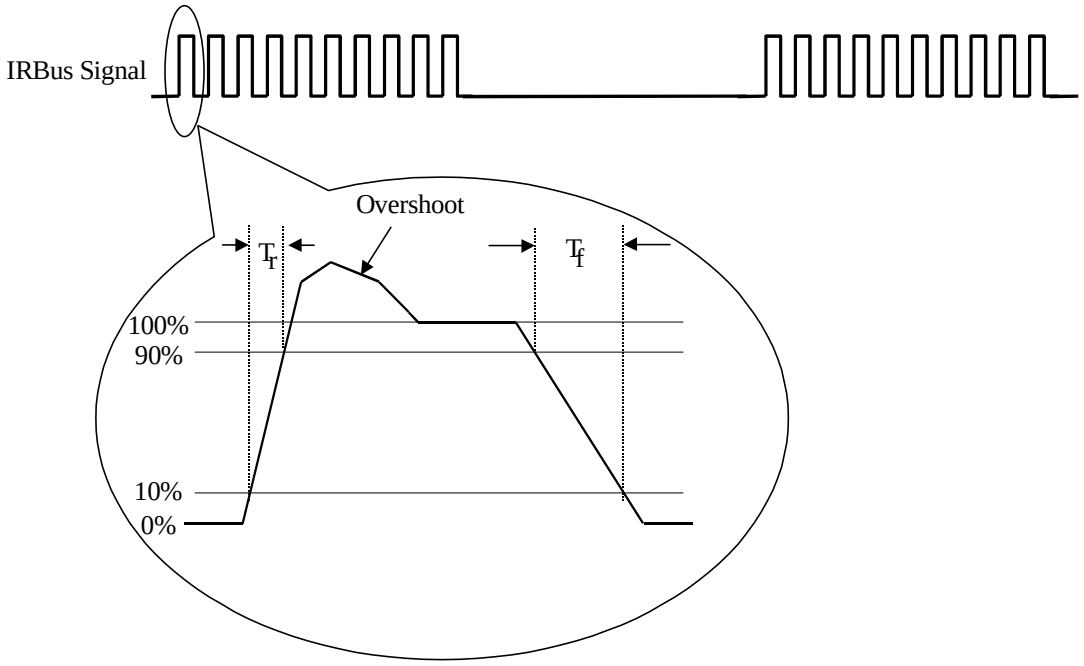


Figure 3.8: Definitions of T_r ,T_f and Overshoot

3.5 IR Output Interface Specifications

The IrDA Control infrared transmitter must satisfy the IR output interface specification shown in Table 3.4.

Specification	Minimum	Typical	Maximum	Unit
Intensity				
Host Devices (θ_H<+/-30deg and ϕ_H=0) *	100		500	mW/sr
Host Devices (θ_H<+/-40deg and ϕ_H=0) *	68		500	mW/sr
Host Devices (θ_H<+/-50deg and ϕ_H=0) *	36		500	mW/sr
Host Devices (θ_H=0 and ϕ_H<+/-15deg) *	100		500	mW/sr
Host Devices (θ_H=0 and ϕ_H<+/-25deg) *	20		500	mW/sr
Peripheral Type 1 Devices (θ_P<+/-15deg and ϕ_p=0) *	100		500	mW/sr
Peripheral Type 1 Devices (θ_P=0 and ϕ_p=<+/-15deg) *	100		500	mW/sr
Peripheral Type 2 Devices (θ_P<+/-40deg and ϕ_p=0) *	9		500	mW/sr
Peripheral Type 2 Devices (θ_P=0 and ϕ_p=<+/-25deg) *	9		500	mW/sr
Signaling Rate				
Host Devices	74.925	75.0	75.075	kbps
Peripheral Type 1 Devices	74.175	75.0	75.825	kbps
Peripheral Type 2 Devices	74.175	75.0	75.825	kbps
Subcarrier Frequency				
Host Devices	1.4985	1.50	1.5015	MHz
Peripheral Type 1 Devices	1.4835	1.50	1.5165	MHz
Peripheral Type 2 Devices	1.4835	1.50	1.5165	MHz

*: See Figure 3.6.

Table 3.4: IR Output Interface Specifications

The tolerance of the signaling rate and subcarrier frequency for host devices, peripheral type 1 devices and peripheral type 2 devices are +/- 0.1%, +/- 1.1% and +/- 1.1% of their typical value, respectively.

Figure 3.9 shows the Intensity in angular range in horizontal plane for respective devices. For angles of any degree, the intensity should be equal to or less than the maximum intensity. In addition, within the defined angular range, the intensity should be equal to or more than the minimum intensity.

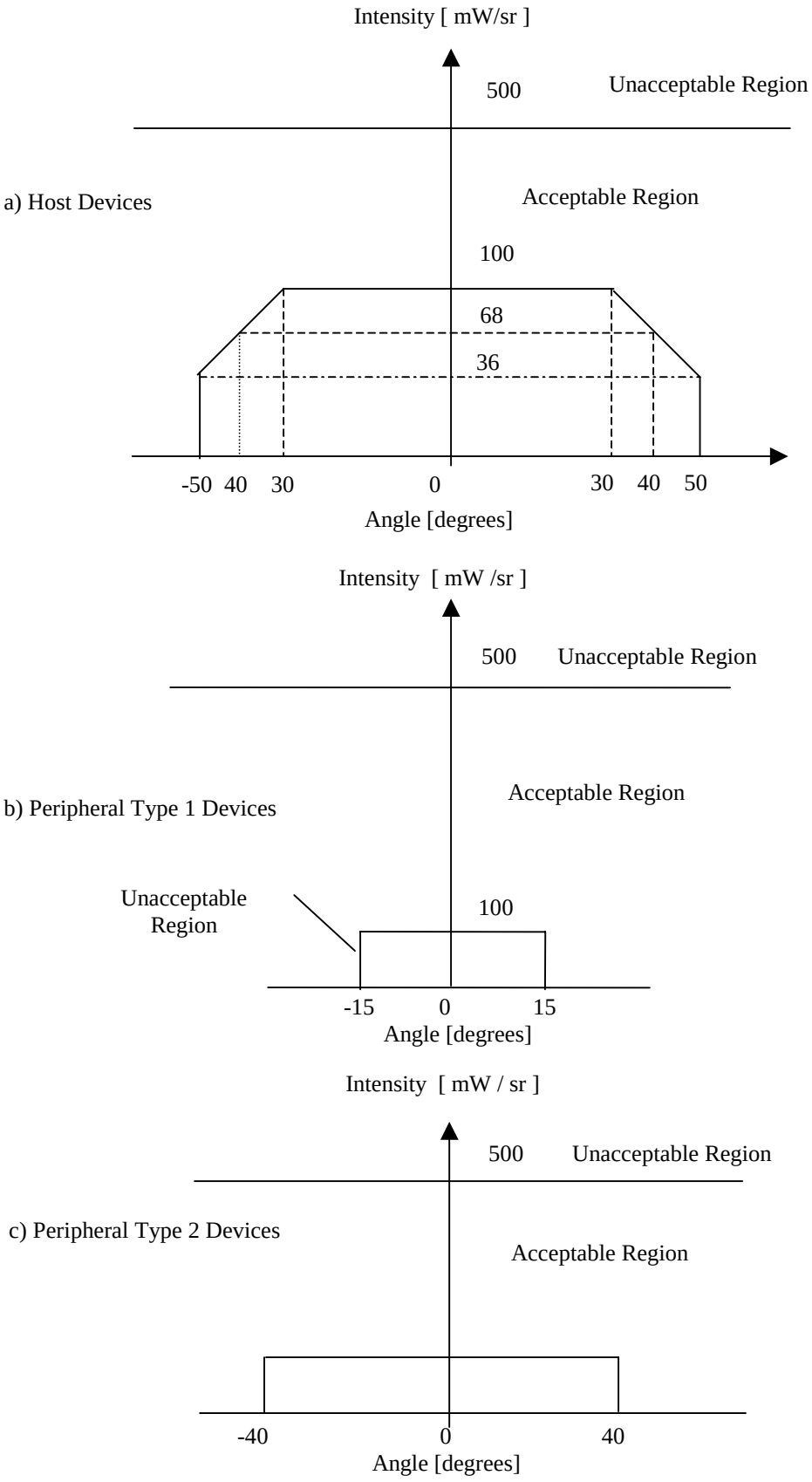


Figure 3.9: Acceptable Intensity Region in horizontal angular range

3.6 IR Input Interface Specifications

The IrDA Control infrared receiver must satisfy the IR input interface specification shown in Table 3.5.

Specification	Minimum	Typical	Maximum	Unit
Irradiance				
Host Devices (θ_H<+/-40deg and ϕ_H=0) *	0.4		1250	$\mu W/cm^2$
Host Devices (θ_H<+/-50deg and ϕ_H=0) *	1.111		1250	$\mu W/cm^2$
Host Devices (θ_H=0 and ϕ_H<+/-25deg) *	0.4		1250	$\mu W/cm^2$
Peripheral Type 1 Devices (θ_P<+/-15deg and ϕ_P=0) *	0.4		1250	$\mu W/cm^2$
Peripheral Type 1 Devices (θ_p=0 and ϕ_P=<+/-15deg) *	0.4		1250	$\mu W/cm^2$
Peripheral Type 2 Devices (θ_P<+/-40deg and ϕ_p=0) *	3.0		1250	$\mu W/cm^2$
Peripheral Type 2 Devices (θ_P=0 and ϕ_P=<+/-25deg) *	0.889		1250	$\mu W/cm^2$
Signaling Rate				
Host Devices	74.175	75.0	75.825	kbps
Peripheral Type 1 Devices	74.925	75.0	75.075	kbps
Peripheral Type 2 Devices	74.925	75.0	75.075	kbps
Subcarrier Frequency				
Host Devices	1.4835	1.50	1.5165	MHz
Peripheral Type 1 Devices	1.4985	1.50	1.5015	MHz
Peripheral Type 2 Devices	1.4985	1.50	1.5015	MHz
Receiver Latency Allowance (All Devices)			133	µs

*: See Figure 3.6.

Table 3.5: IR Input Interface Specifications

Because an infrared receiver might saturate when it has received a signal from the adjacent transmit IRED(s) in a transceiver package, time is required for the infrared receiver to return to the maximum sensitivity state immediately after a transmission. This time is defined as the Receiver Latency Allowance. In implementing the infrared receiver, a sensitivity-reset circuit may be needed to meet this requirement.

Figure 3.10 shows the Irradiance in angular range in horizontal plane for respective devices.
Within the thus defined angular range, the irradiance should be in the range of the minimum irradiance to the maximum irradiance. For optical inputs within the range shown in Figure 3.9, the infrared receiver must operate with communication quality of BER $=10^{-4}$.

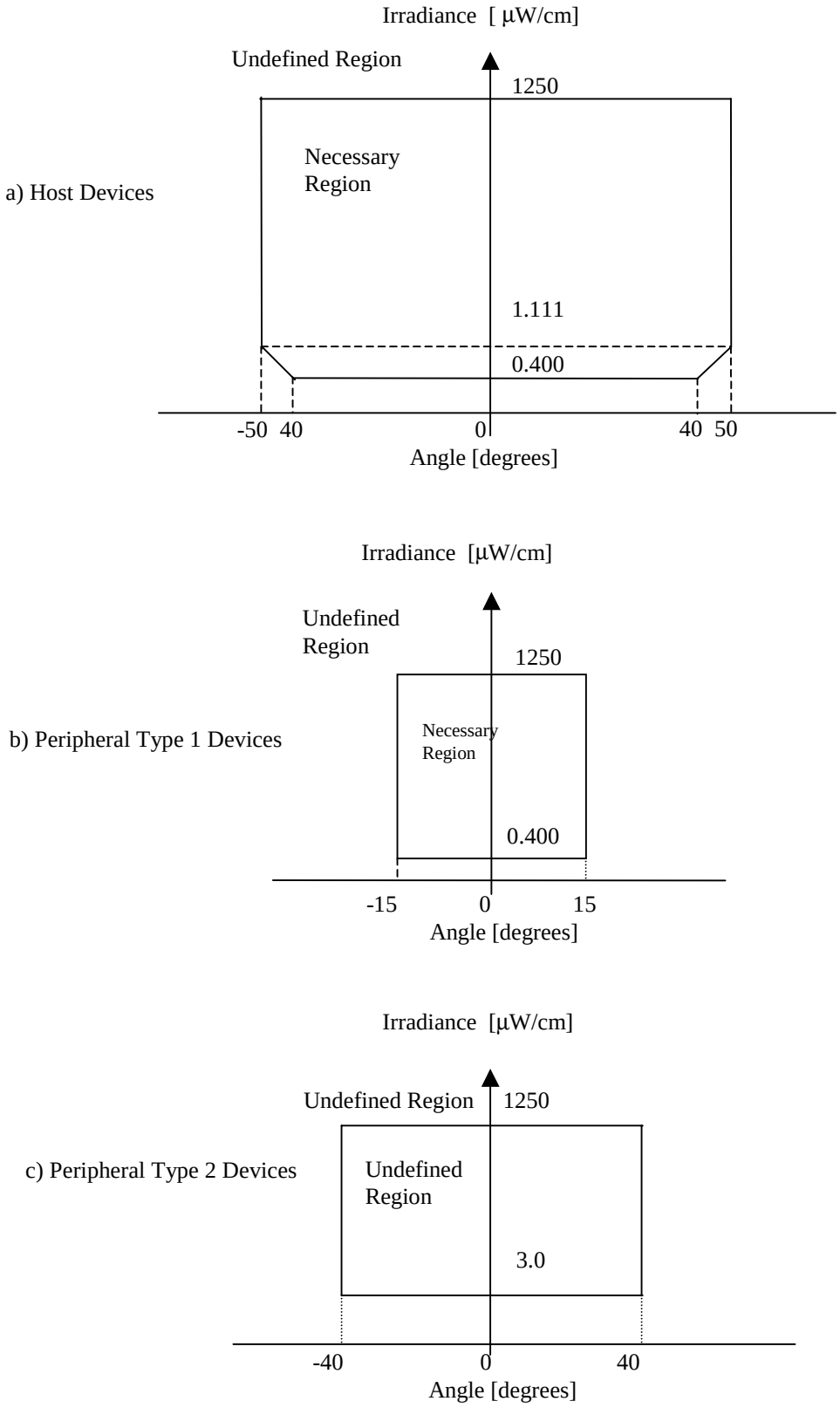


Figure 3.10: Necessary Irradiance Region in horizontal angular range

4 Media Access Control Layer (MAC)

4.1 Overview

The IrDA Control system consists of hosts and peripherals between which infrared communication takes place. In the IrDA Control system, a host manages its communications with multiple peripherals on a time division basis (poll-response), so that those devices can communicate with time-shared simultaneity. Communications only occur between the host and the peripherals. In general, a peripheral cannot transmit until it receives response permission from a host. Hosts do not communicate with each other as peers. However, a host may listen to another host in a multi-host environment. A host might act as a peripheral when it needs to communicate with another host. Multiple hosts in the same space cannot communicate simultaneously. However, multiple hosts could share the IR medium by time division multiplexing, under the constraints described in Appendix G.

Polling is the process in which a host issues a "response permission" for each peripheral and receives data from each peripheral. However, if a peripheral detects that a host is asleep (Mode-0), it is allowed to transmit a frame to wake up the sleeping host.

If there has been no input from any peripheral for given time, a host enters the sleep state (Mode-0), and stops all transmission.

Their respective addresses and identifiers identify hosts and peripherals. An 8-bit host address (HADD) and a 16-bit host ID (HostID) identify a host. A host address (HADD) may be set at the factory, or determined while the host is set up. A peripheral is identified by a 32 bit physical ID (PFID). A host and a peripheral have to exchange address/ID information (HADD/HostID and PFID) as part of a process called enumeration. A logical 4-bit peripheral address (PADD) is uniquely assigned to each peripheral by the host to establish “active” communication. This procedure is a part of a process called binding, which is performed when an enumerated peripheral attempts to establish communication with the host. The ID numbers (HostID/PFID) are used only in the beginning of a communication to identify the devices, and after the identification, hosts/peripherals are identified only by their address (HADD/PADD).

The requirements for infrared data communication vary depending on the application. In order to comply with various application requirements, the IrDA Control offers three operational modes for a host.

Mode-0 - Sleep Mode

This is a “Low resource usage” mode to minimize power consumption when a host and its peripherals do not need to communicate. This is also the default mode for each host.

Mode-1 - Normal Mode

This is the normal operational mode of the host. This mode supports peripherals that may have different bandwidth requirements. Peripherals supported include devices that must be handled within a certain time limits (Critical Latency peripheral, or, CL peripheral), like joysticks and game pads. Peripherals that normally do not have critical latency requirements (Non-critical Latency peripheral, or, NCL peripherals), like Remote Control units are also supported. Keyboards and Mice could be handled as NCL or CL peripherals under this mode. A CL peripheral is able to support CL polling rate. On the other hand, an NCL peripheral is not able to support CL polling rate and is always polled at the NCL polling rate. A host must guarantee a peripheral at the CL polling rate be polled every 13.8msec.

Mode-2 - IrDA-coexistence mode

This operating mode is available to allow coexistence of IrDA SIR version 1.1 data communication and IrDA Control communication.

A host mode change is based on the state transition diagram shown in Figure 4.1. It is not necessary for every host to support all three modes. For example, a host is allowed to implement Mode-0 and Mode-1 only. However, all transitions among supported modes must be implemented.

The time when a peripheral may transmit a frame to a host is determined by the information sent from the host to each peripheral. The host mode determines how each frame is configured. The concept of mode is related to the host. Peripherals are non-modal in that they do not directly determine the mode, and they are not aware of the mode the host is running in. The definitions and operations are described in section 4.3.

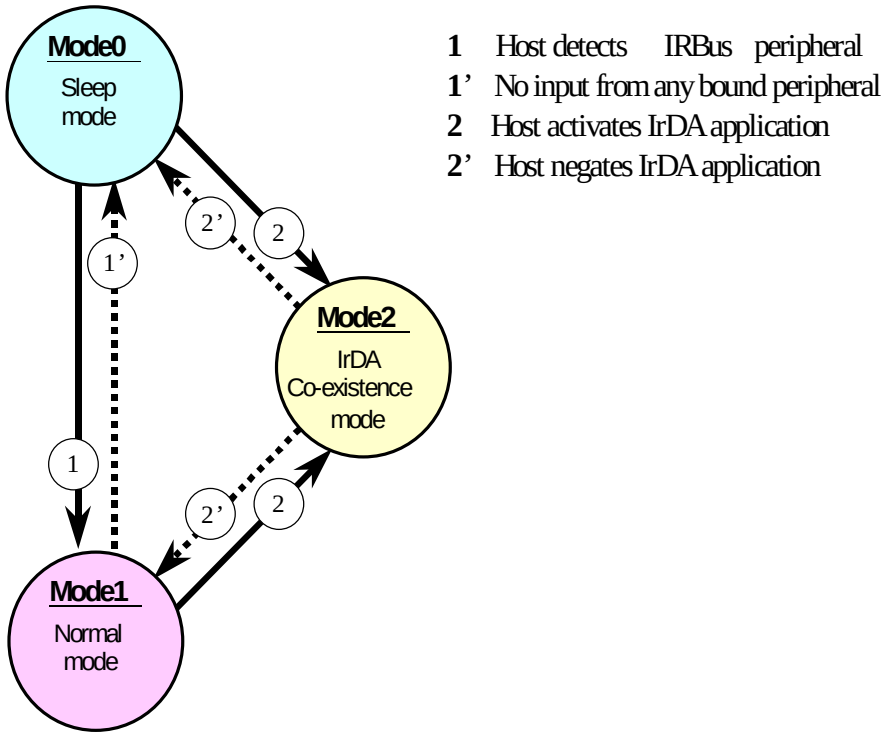


Figure 4.1 Mode state transition diagram of IrDA Control Host device

Two kinds of MAC frames are defined based on the maximum MAC payload data length that can be transmitted by a host or a peripheral. One is a short frame and the other is a long frame. A short frame can accommodate up to 9 bytes of MAC payload data and must be transmitted with the STS flag, STO flag and CRC-8. A long frame can accommodate up to 97 bytes of MAC payload data and must be transmitted with the STL flag, STO flag and CRC-16. Long frames are suitable for larger data exchanges.

Host devices and peripheral devices may always use short frames. Host devices may use long frames in Mode-1 only. Peripheral device may use long frames, only when responding to a polling packet from a host device whose long frame enable bit is set to 1, which occurs when the host is in Mode-1. It is prohibited that both a host device and a peripheral device use long frame in the same polling procedure (in the polling frame from a host as well as the responding frame from a peripheral). The possible short/long frame combinations used in hosts and peripherals are summarized in Table 4.1.

Host Status		From Host to Peripheral (MAC payload length)	From Peripheral to Host (MAC payload length)
To unenumerated peripherals		Short (0 - 9 bytes)	Short (0 - 9 bytes)
To unbound peripherals in Mode-1/2		Short (0 - 9 bytes)	Short (0 - 9 bytes)
To bound peripherals	Mode-0	-	Short (0 - 9 bytes)
	Mode-1	Short (0 - 9 bytes)	Short (0 - 9 bytes)
		Short (0 - 9 bytes)	Long (0 - 97 bytes)
		Long (0 - 97 bytes)	Short (0 - 9 bytes)
	Mode-2	Short (0 - 9 bytes)	Short (0 - 9 bytes)

Table 4.1 Possible Short/Long frame combination

The basic polling cycle time is defined as 13.8ms. Up to 4 CL peripherals at the CL polling rate can be polled with short frames during this cycle time. The basic polling cycle time (13.8ms) is based on the minimum interval between inputs from a human input device. Therefore, a host device that finishes polling all its bound peripherals within 13.8ms must wait for the expiration of the current 13.8ms period before starting its next polling cycle.

Each CL peripheral at the CL polling rate is polled within this basic polling time (13.8ms) in Mode-1. However, a peripheral at the Non-critical Latency polling rate is not guaranteed a poll within this basic polling time (13.8ms). The entire polling cycle time is defined as the time period in which all bound peripherals can be polled by a host. The host has to manage peripherals so that the entire polling cycle time is shorter than 69ms. The following Table 4.2 shows the maximum number of each device that can be accommodated in each host mode.

Host mode	IrDA 1.1 device	Peripherals at the CL polling rate	Peripherals at the NCL polling rate		Entire Polling Cycle Time
		Short	Long	Short	
Mode-1	0	0	0	8	$8\ T_{SS}$
			1	7	$1T_{SL} + 7\ T_{SS}$
			2	6	$2T_{SL} + 6\ T_{SS}$
			3	5	$3T_{SL} + 5\ T_{SS}$
			4	4	$4T_{SL} + 4\ T_{SS}$
			5	1	$5T_{SL} + 1\ T_{SS}$
		1	0	7	$10\ T_{SS}$
		2		6	$12\ T_{SS}$
		3		5	$20\ T_{SS}$
		4		1	$17\ T_{SS}$
Mode-2	1	0	0	2	$T_{IrDA} + 2\ T_{SS}$

Table 4.2 Maximum number of peripherals that can be accommodated in a Host mode

T_{SS} means the time which is required for a host to poll one peripheral by Short frame polling packet / Short frame responding packet. T_{SL} means the time which is required for a host to poll a peripheral by Short frame polling packet / Long frame responding packet, or, Long frame polling packet / Short frame responding packet. T_{IrDA} means the time, which is required for a host to communicate with one IrDA SIR 1.1 device in Mode-2. The maximum time of T_{SS}, T_{SL}, T_{IrDA} is shown in Table 4.3.

	Max. (The last GAP time = 16 bit time)	Actual Max. time (including clock tolerance)
T_{SS}	256 bit time	3.45ms
T_{SL}	968 bit time	13.05ms
T_{IrDA}	50 ms	50.00ms

Table 4.3 Max time of $T_{SS,}$ $T_{SL,}$ T_{IrDA}

4.2 MAC Frame Structure

A MAC frame is composed of a Host Address field (1Byte), Peripheral Address field (4-bits), Mac Control field (4-bits), and MAC payload (0 – 97 Bytes). The MAC frame format is shown in Figure 4.2.

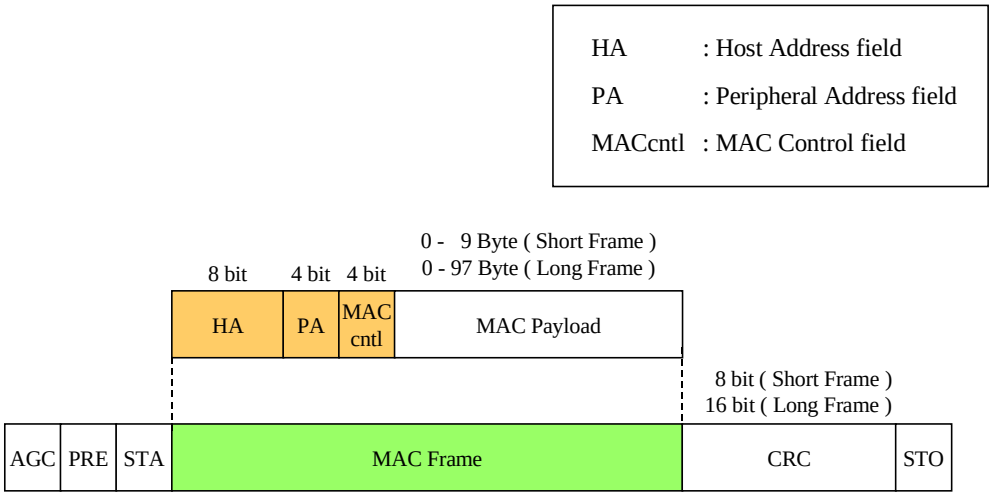


Figure 4.2 MAC Frame Format

4.2.1 Host Address Field

An 8-bit host address (HADD) is assigned to a host. It may be preset at the factory or determined when the host is set up. The host may also provide a mechanism to change its host address. HADD “0x0” is a special host address used for broadcast wake-up for the enumeration procedure. Any host device in Mode-0 that has no dedicated switch for enumeration has to wakeup and start the enumeration procedure on receiving this broadcast host address.

4.2.2 Peripheral Address Field

A peripheral is identified by its own 32 bit physical ID (PFID). A peripheral is assigned a 4-bit peripheral address (PADD) from the host in the process of binding to a host. The Peripheral address (PADD) assigned to a peripheral is not permanent, and may change during every binding procedure.

The PADD is necessary to establish “active” communication with a peripheral. A host must (dynamically) maintain the mapping between the 4-bit PADDs and the 32 bit physical IDs (PFIDs) of the peripherals that it has currently bound.

PADDs “0x0” and “0xF” are reserved peripheral addresses, used for binding and enumeration, respectively.

Peripheral address “0x0” is used for enumerated, but unbound, peripheral devices to respond to their host. The host periodically polls the peripheral address “0x0” (Device hailing for binding). Unbound peripherals are allowed to respond to host polls with a PADD “0x0” only.

Peripheral address “0xF” is used in the process of enumeration. During enumeration, the host polls using a peripheral address “0xF”. Unenumerated peripherals are allowed to respond to host polls with PADD of “0xF” only.

4.2.3 MAC Control Field

The MAC layer uses this control field. The meanings of each bit in the MAC control field are dependent on whether the frame is sent from a host to a peripheral or from a peripheral to a host. The meaning of each bit in the control field is shown in Table 4.4.

		Frame from Host device		
		Meaning	1	0
MAC Control field	**D7**	Packet direction	1 (Host is sending)	
	D6	Bind timer restarted	Restarted	Not Restarted
	D5	Long frame enable	Enable	Disable
	D4	Device hailing	Hailing	Not hailing
Peripheral Address field	**D3**	(Peripheral Address)	-	-
	D2		-	-
	D1		-	-
	D0		-	-

		Frame from Peripheral device		
		Meaning	1	0
MAC Control field	**D7**	Packet direction	0 (Peripheral is sending)	
	D6	Polling Request	Request	No Request
	D5	(Reserved)	(Reserved)	
	D4	(Reserved)	(Reserved)	
Peripheral Address field	**D3**	(Peripheral Address)	-	-
	D2		-	-
	D1		-	-
	D0		-	-

Table 4.4 MAC Control field

MAC Control field (Frame from Host device)

D7 (Packet Direction)
The '1' indicates that this packet is sent from a host to a peripheral.

D6 (Bind timer restarted)
This bit is set to '1' if a host has received a frame from a peripheral in response to a previous poll to the peripheral, and the host has restarted its bind timer from the initial value.

D5 (Long Frame Enable)
This bit is set to '1' when a host allows the polled peripheral to use a long frame in its response, and is set to '0' when a host does not allow the polled peripheral to use a long frame. If long frames are disabled, only 9 bytes (or less) of data can be contained in the MAC payload field. If long frames are enabled, up to 97 bytes (or less) of data can be contained in the MAC payload field.

D4 (Device Hailing)
This bit is set to '1' when the host initiates a device hail to PADD="0x0" or "0xF". If the host detects a response to this device hail, the host will reset this bit and start binding or enumeration procedure, respectively.

MAC Control field (Frame from Peripheral device)

D7 (Packet Direction)
The '0' indicates that this packet is sent from a peripheral to a host.

D6 (Polling Request)
This bit is set to '1' when a peripheral requests to be polled, and is set to '0' when a peripheral requests that it not be polled anymore.

D5 (Reserved)
This bit is reserved. Set to '0'.

D4 (Reserved)
This bit is reserved. Set to '0'.

4.3 Host Operating Mode

4.3.1 Enumeration

Enumeration is the procedure in which a host and a peripheral recognize (discover) each other to enable communication between them. The host identifies the peripheral using the peripheral physical identifier (PFID) and the peripheral identifies the host using a host address (HADD). The PFID and the HADD are exchanged during the enumeration procedure. Figure 4.3 illustrates the enumeration procedure.

An IrDA Control peripheral must be enumerated (and bound) with a host before it can exchange data with the host application layer. A peripheral that has not been enumerated must not perform any communication other than the enumeration procedure. The host ignores a hailing response received from any peripheral to which it has not enumerated.
Special mechanisms may be required on IrDA Control devices to initiate the enumeration procedure. One such mechanism is described later in this section.
The enumeration procedure uses short frames only and is carried out in the following steps.

Enumeration procedure

IrDA Control Enumeration (Active Host)

Peripheral address “0xF” is used in the process of enumeration. During enumeration, the host polls using a peripheral address “0xF”. Unenumerated peripherals are allowed to respond to host polls with PADD of “0xF” only.

1. The host issues an enumeration hail with the ‘hailing’ bit set to 1, and a peripheral address 0xF. This host poll frame includes information about the host (Host ID and Host Info). The format of the Host ID and Host Info field is defined in section 4.5.1.

Source	HADD	MACC	PADD	DATA
Host	Host Addr	0x9	0xF	Host ID + Host Info

2. After storing the HADD, HostID and Host Info data, a peripheral that desires enumeration responds to the frame in (1) with a frame including its PFID and information about itself (Peripheral Info). The Peripheral Info tells the host whether the peripheral is a critical latency peripheral (i.e., the peripheral supports the CL polling rate) or not, as well as whether the peripheral has the ability to send or receive long frames. The format of the PFID and Peripheral Info field is defined in section 4.5.2.

Source	HADD	MACC	PADD	DATA
Peripheral	Host Addr	0x04	0xF	PFID + Peripheral Info + Host ID

3. The host, which has received the frame in (2), stores the PFID and Peripheral Info. Then in the next polling cycle, it responds to the frame in (2) with a frame including the received PFID.

Source	HADD	MACC	PADD	DATA
Host	Host Addr	0x8	0xF	PFID

The enumeration procedure may fail due to multiple peripherals responding to the same hail. After responding to the enumeration hail, the peripheral should receive a response from the host with PFID (see step 3 above). If a peripheral does not receive the above packet from the host within 69 ms after a request, the peripheral recognizes the failure and goes back to (2) with a random back-off value between 0 to 7. If the random back-off value is 0, this peripheral can send a frame described in (2) in the next hailing cycle. If the random back-off value is 7, this peripheral will ignore 7 hailing frames and can send a frame described in (2) in the 8th hailing cycle.

IrDA Control Enumeration (Inactive Host)

1. A sleeping host (Mode-0) does not hail. On user activity, an unenumerated peripheral waits for up to 1 second for the host hail. If the host hail is not received in 1 second, the peripheral infers that the host is in sleep mode (Mode-0) and transmits a frame with the polling request bit set to 1, the HADD=”0x0” and PADD=”0xF”.

Source	HADD	MACC	PADD	DATA
Peripheral	0x0	0x4	0xF	None

2. The host which has the ability to receive the broadcast HADD frame has to respond with a device hail for enumeration (i.e. step 1). The enumeration procedure continues as described above with active host. The sequence is shown below for completeness.

Source	HADD	MACC	PADD	DATA
Host	Host Addr	0x9	0xF	Host ID + Host Info

Source	HADD	MACC	PADD	DATA
Peripheral	Host Addr	0x4	0xF	PFID + Peripheral Info + Host ID

Source	HADD	MACC	PADD	DATA
Host	Host Addr	0x8	0xF	PFID

Host Re-enumeration

The host will consider all peripherals unenumerated under the following circumstances:

- The host/dongle has been power-cycled, re-booted, re-connected or reset. The host/dongle has lost track of all peripheral IDs and PADDs, as well as being re-configured on USB.

Peripheral Re-enumeration

The peripheral will consider itself unenumerated under the following circumstances:

- The peripheral has been power-cycled or reset and has lost track of the Host ID and HADD.
- The peripheral is unbound and during binding with its host/dongle, the HostID does not match with the HADD issued during the previous bind with that HADD.

The enumeration is complete when all procedures mentioned above are completed. On completion of enumeration, the peripheral is allowed to enter the host's polling cycle through the binding procedure below.

The enumeration procedure discussed here only establishes a peripheral association between the host and the peripheral. Peripheral specific information is exchanged, if needed, in the upper layers, which is specific to the type of upper layer (for example, HID or HA).

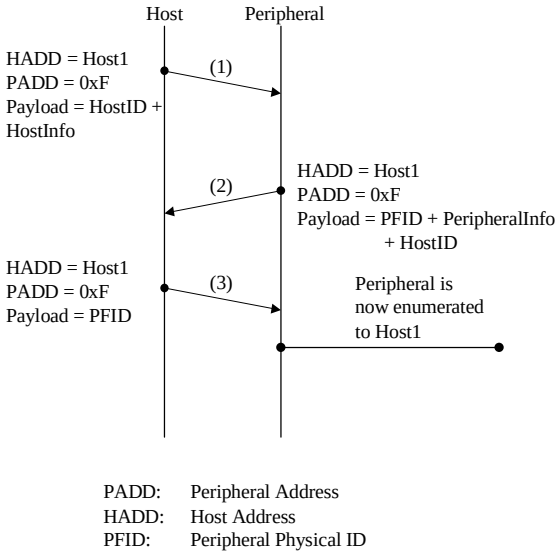


Figure 4.3 Enumeration Procedure

The procedures for initiating upper layer enumeration are not fully specified in this specification.

A certain class of hosts may initiate enumeration by a dedicated switch or some other mechanism.
An IrDA Control peripheral is not required to respond to the enumeration polls until a special trigger activates the enumeration procedure for that peripheral. This trigger could be a dedicated switch or some other mechanism.

Periodic enumeration

An IrDA Control host that implements periodic enumeration scheme discovers new devices by hailing. Such a host hails periodically for new devices to enumerate by polling to peripheral address '0xF'. The enumeration polls are issued only when the host is in Mode-1. The duration of the host polling cycle determines the frequency of enumeration polls.

Scan mechanism

An IrDA Control peripheral that implements a scan mechanism allows enumeration to a chosen host. In the presence of multiple IrDA Control hosts, a peripheral is enumerated to the host that sent the first enumeration poll after the peripheral activated its enumeration procedure. To allow a peripheral to enumerate to a chosen host, a scan procedure is implemented by the IrDA Control peripherals. This procedure allows a peripheral to cycle through all the hosts in the range, enumerating with each one in turn, eventually enumerating with the chosen host. The peripheral always picks the host with the next highest host address to the current host address for enumeration. For example, a peripheral currently enumerated to host address 20 cycles enumerating with hosts with addresses 30 and 234 before eventually enumerating with the chosen host with host address 2, as shown in Table 4.5.

The host must indicate a successful enumeration to the user. The mechanisms for implementing the user indications (e. g., flash an LED) are not specifically specified.

Current host binding (HADD) for peripheral	Hosts sending enumeration polls (Host addresses)	Next host binding (next higher host number MODULO 256)
20	2, 30, 234	30
30	2, 30, 234	234
234	2, 30, 234	2 (Chosen host!)

Table 4.5 Enumeration in presence of multiple hosts

4.3.2 Binding

The process in which a host dynamically recognizes that an enumerated peripheral needs to be added to the active device polling loop is called "Binding". Figure 4.4 illustrates the binding procedure.

When a bound peripheral does not respond to polling for a certain number of times or a certain time period, the host recognizes that the peripheral does not need further communication and stops polling. This process is called "Unbinding."

IrDA Control Binding (Active Host)

The binding procedure is carried out in the following steps:

1. An active host periodically sends a frame with PADD="0x0" and with the 'hailing' bit set to 1. This process is called "Hailing", and any unbound, but enumerated (by this host) peripheral may respond to this frame. This host poll frame includes the identification of the host (Host ID).

Source	**HADD**	**MACC**	**PADD**	**DATA**
Host	Host Addr	0x9	0x0	Host ID

2. A peripheral which has received a user input and is about to start communication responds to this hailing with a frame where the PADD="0x0". The payload data includes the peripheral's 32-bit physical ID (PFID).

Source	HADD	MACC	PADD	DATA
Peripheral	Host Addr	0x4	0x0	PFID

3. The host that receives the peripheral frame (in step 2) correctly will poll in the next cycle to PADD="0x0" with a frame whose "hailing" bit is reset to 0 and "bind timer restarted" bit is set to 1. The payload includes the peripherals 32-bit physical ID (PFID) and a new allocated peripheral address (PADD = 0xN).

Source	HADD	MACC	PADD	DATA
Host	Host Addr	0xC	0x0	PFID + PADD (0xN)

On completing step 3, the host will include the peripheral into its polling loop and reset its "Bind Timer" to the initial value.

The binding procedure may fail due to multiple peripherals responding to the same hail, or multiple peripherals trying to wake a sleeping host simultaneously. If a peripheral does not receive a PADD assignment packet within 69ms after a request (i.e., after step 2), the peripheral recognizes the failure and goes back to (2) with a random back-off value between 0 to 7. If the random back-off value is 0, this peripheral is allowed to send a frame described in (2), in the next hailing cycle. If the random back-off value is 7, this peripheral must ignore the next 7 hailing frames and send a frame described in (2) in the 8th hailing cycle.

4. The peripheral that receives the host frame in step 3 correctly may respond to the future host polls to its assigned address, namely, PADD="0xN".

Source	HADD	MACC	PADD	DATA
Peripheral	Host Addr	0x4	0xN	Payload Data

5. The peripheral will be polled with PADD="0xN" during the next polling cycle.

Source	HADD	MACC	PADD	DATA
Host	Host Addr	0xX	0xN	(Payload Data)

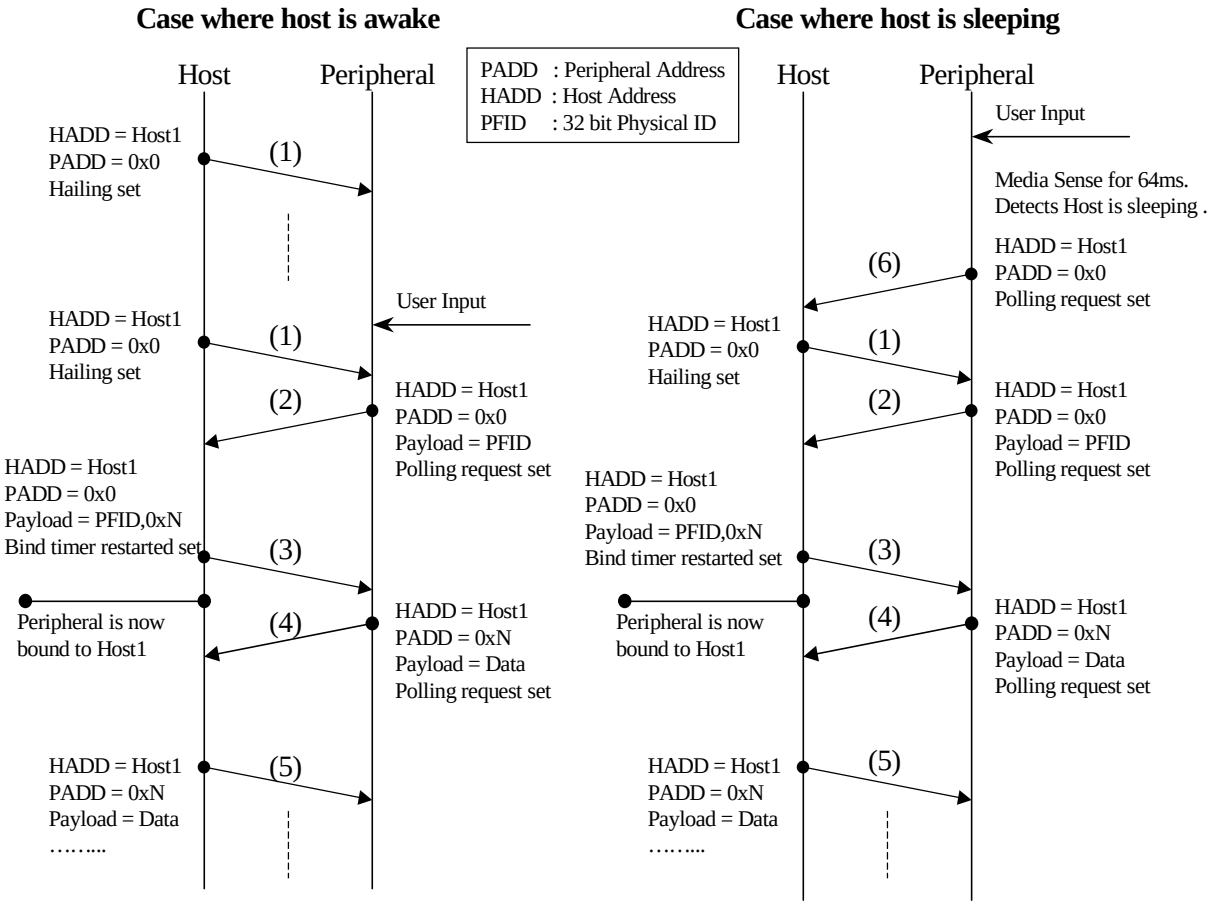


Figure 4.4 Binding a previously Enumerated Peripheral

Binding of Enumerated Peripheral (Inactive Host)

6. A sleeping host (Mode-0) does not hail. On user activity, an unbound peripheral waits for up to 69ms for the host hail. If the host hail is not received in 69ms, the peripheral infers that the host is in sleep mode (Mode-0) and transmits a frame with the polling request bit set to 1 and PADD="0x0".

Source	HADD	MACC	PADD	DATA
Peripheral	Host Addr	0x4	0x0	HostID

The host has to respond with a peripheral hail after it receives a polling request from a peripheral with peripheral address "0x0". The binding procedure continues as described above with active host. The sequence is shown below for completeness.

Source	HADD	MACC	PADD	DATA
Host	Host Addr	0x9	0x0	HostID

Source	HADD	MACC	PADD	DATA
Peripheral	Host Addr	0x4	0x0	PFID

Source	HADD	MACC	PADD	DATA
Host	Host Addr	0xC	0x0	PFID + PADD (0xN)

Source	HADD	MACC	PADD	DATA
Peripheral	Host Addr	0x4	0xN	Payload Data

Source	HADD	MACC	PADD	DATA
Host	Host Addr	0xX	0xN	(Payload Data)

Once a peripheral is bound to a host device, the host will start to poll that peripheral. However, if a specific peripheral does not respond to the host's polling for a certain time, that peripheral will be unbound, and the host will stop polling that peripheral. An NCL peripheral will be unbound if it does not respond to polling for 5 seconds, and a CL peripheral will be unbound if it does not respond to polling for 30 seconds.

Both NCL and CL peripheral have to count the time so that they can judge whether they are bound to the host or not. If a peripheral wants to send a frame while it is bound to the host, it can respond to the host polling frame whose PADD is the same as the last-allocated PADD="0xN" of this peripheral. If it wants to send a frame after it is unbound from the host, it must not respond to a host's polling frame even if the (host sent) PADD is the same as the last PADD allocated to this peripheral.

4.3.3 Unbinding

A peripheral must count the time that has elapsed since its last communication with the host and thereby check whether it has been unbound from the host. A host keeps a peripheral bound and continues to poll that peripheral for 69ms longer than the time which the peripheral regards itself as bound by the host, in order not to re-bind another peripheral with the same PADD while the original peripheral still regards itself as bound.

Unbind times	NCL Peripheral	CL Peripheral
Continued Host polling	5 seconds plus 69ms	30 seconds plus 69ms
Peripheral regards itself as bound	5 second	30 seconds

Peripherals that do not want further polls from the host can request unbinding by sending a frame whose 'Polling request' bit is 0. The host that receives a frame whose 'Polling request' bit is 0 should stop polling the peripheral immediately.

Host's binding/Unbinding rule

Once a peripheral is bound to a host, the host will start to poll that peripheral. However, if a specific peripheral does not respond to the host's polling for a certain time (5 seconds plus 69ms for NCL peripheral, 30 seconds plus 69ms for NCL peripheral), that peripheral will be unbound, and the host will stop polling that peripheral. More detailed host binding/unbinding rules are listed below.

- A host has "Bind Timers" for each peripheral. Each "Bind Timer" will be set to its initial value (5 seconds plus 69ms for NCL peripheral, 30 seconds plus 69ms for NCL peripheral) upon receiving a peripheral response to a host polling. A host that receives a peripheral's response to the host's polling has to set the "bind timer restarted" bit in the next polling to that peripheral.
- The host will unbind a peripheral when the "Bind Timer" for that peripheral expires.
- A host unbinds any bound peripheral if it receives a poll response with "polling request" bit reset to 0.

Peripheral's binding/Unbinding rule

If a peripheral does not receive the host's polling with "bind timer restarted" bit set to 1 for a certain time (5 seconds for NCL peripheral, 30 seconds for NCL peripheral), that peripheral has to regard itself as unbound from the host. More detail peripheral binding/unbinding rules are listed below.

- A peripheral has a "Bind Timer". This "Bind Timer" will be set to its initial value (5 seconds for NCL peripheral, 30 seconds for CL peripheral) upon receiving a host polling with "bind timer restarted" bit set to 1.
- The peripheral regards itself as unbound from the host when the "Bind Timer" expires.
- A peripheral may request to the host to unbind itself by sending a poll response with the "polling request" bit reset to 0. In this case, the peripheral must not respond to the succeeding host's polling frames.
- The peripheral regards itself as unbound when it has been power-cycled or reset and has lost track of Host ID and HADD.

After a peripheral has been enumerated and bound at the IrDA Control level, it may expect that it will be enumerated at a higher level (for example, USB-HID). This may take long enough for the peripheral to unbind. To prevent unbinding, a peripheral may request to be kept in the polling loop by sending a dummy frame with the polling request bit set to 1.

Source	HADD	MACC	PADD	DATA
Peripheral	Host Addr	0x4	0xN	None

4.4 Modes

4.4.1 Sleep Mode (Mode-0)

This is the default mode for any host. When a host is in this mode, it does not poll any peripherals. If a host in this mode receives a frame from a peripheral requesting that the host start polling it (Wake up frame; rebind or new device), then the host changes its state to Mode-1 and starts polling for the peripherals. In this case, binding must be performed to associate a peripheral with the host before the peripheral can start communication with the host.

Only the host with the matching host address (HADD) wakes up in response to a wakeup frame from a peripheral. This allows only enumerated peripherals to wakeup a specific host. However, a wakeup frame with a host address (HADD) of 0 is treated as a broadcast request to such hosts that support a periodic enumeration scheme.

Such hosts that support a periodic enumeration scheme must wakeup and resume host activity including enumeration polls, when they receive a wakeup frame with a host address of 0. This scheme allows unenumerated peripherals to wakeup hosts for the purpose of enumeration.

The mode changes related to Mode-0 are listed below:

Change from Mode-0 to another mode:

- If a host has received a wake-up frame from a NCL or a CL peripheral (it enters Mode-1).
- If a host activates an IrDA SIR version 1.1 data communication application (it enters Mode-2).

Change to Mode-0 from another mode:

- If a host in Mode-1 has received no frames from any NCL peripherals for 5 seconds and no frames from any CL peripherals for 30 seconds.
- If a host in Mode-2 has finished an IrDA SIR ver1.1 data communication, and has received no frame from any NCL peripherals within 5 seconds.

4.4.2 Normal Mode (Mode-1)

This mode supports peripherals with varying latency requirements. Typically, the peripherals supported in this mode include critical latency peripherals such as joysticks and gamepads, as well as non-critical latency peripherals such as keyboards, mice and RC units.

A host in this mode implements a dynamic polling scheme. The host may poll each bound peripheral at one of the following distinct rates: NCL polling rate, CL polling rate. The polling rate for a peripheral is dependent on the peripheral activity number and the maximum polling rate supported by that peripheral.

The maximum polling rate supported by a peripheral is specified as part of its 'PeripheraInfo' and defines the maximum rate at which a peripheral is capable of generating data. Regardless of this maximum rate, a host always binds a new peripheral at the NCL polling rate.

The host keeps a running count of the number of polls to each of the bound peripherals and the number of responses with new data from the peripheral (NAK responses are not counted as new data). The host periodically computes the peripheral activity number for each bound peripheral. The peripheral activity number is the ratio of new data responses to the 100 polls to a peripheral. If the peripheral activity number exceeds the threshold for the current polling rate, the peripheral is polled at a higher rate (bounded by the maximum rate supported by the peripheral). Similarly, if the peripheral activity number is less than the threshold for the current polling rate, the peripheral is polled at a lower rate.

Under this scheme, a CL peripheral (such as a joystick) is not always polled at the CL polling rate. The polling rate is varied depending on the activity of the device. When the CL peripheral activity decreases, the polling rate is reduced to a NCL polling rate. Further NCL peripherals (such as a keyboard) could specify CL polling rate as its maximum rate. This allows such peripherals to be treated as CL peripherals when the device activity exceeds the threshold for the NCL polling rate.

This mode operates on the following procedure:

1. The host polls in turn for CL peripherals at the Critical Latency polling rate. If the host has data to transmit to the peripherals, the data being transmitted may be included in its response request frame. At least 4 CL peripherals at the Critical Latency polling rate are supported in this mode. There are cases where 4 CL devices are the maximum numbers that can be handled (dependent on the amount of data that the devices need).

2. After finishing the polling of all bound CL peripherals at the Critical Latency polling rate, the host polls all peripherals at the NCL polling rate and polls to PADD='0x0' to find new devices (for binding) in a round robin scheme. If the host cannot poll one of these peripherals during some cycle, it tries to poll those peripherals during subsequent cycles. A host in this mode is guaranteed to poll a bound NCL peripheral and poll to PADD='0x0'at least once every 69ms (or equivalently, at least once every 5 host polling periods). If any specific NCL device has not responded to the host polls for 5 seconds, or if any specific CL device has not responded to the host polls for 30 seconds, the host unbinds that peripheral.

3. A host which supports a periodic enumeration scheme polls to PADD='0xF' at least once in 69ms to find new devices (for enumeration).

A host in Normal Mode must never fail to transmit a frame to each of its bound CL peripherals at the CL polling rate every 13.8ms. The entire polling procedure in paragraphs (1) and (2) does not always fall within 13.8 ms. If the entire polling procedure in paragraphs (1) and (2) are finished in a shorter time than 13.8 ms, the host must wait until 13.8 ms elapse and then resume the procedure described in paragraph (1). If the entire polling procedure in paragraphs (1) and (2) take longer than 13.8 ms, the host can resume the procedure described in paragraph (1) immediately.

If a host has received no frame from bound NCL peripheral in 5seconds, it unbinds the peripheral.
A CL peripheral that does not respond to host polls drops down to an NCL polling rate before the host eventually unbinds the peripheral when it has received no frame from that CL peripheral for 30 seconds plus 69ms.

Long frames are not supported if the host is currently polling any peripheral at the CL polling rate. However, the host completes a long frame transfer currently in progress prior to transitioning a peripheral to the CL polling rate.

Further, the host rejects a request to change to the IrDA coexistence mode, if even one of the peripherals in its active polling loop is currently running at the CL polling rate.

The mode changes related to Normal Mode are listed below:

Change from Normal Mode to another mode:
- If a host in Mode-1 has received no frames from any NCL peripherals for 5 second plus 69ms and no frames from any CL peripherals for 30 seconds plus 69ms. (it enters Mode-0)
- If a host activates an IrDA SIR ver1.1 data communication application and the host is not currently polling any peripheral at the CL polling rate (it enters Mode-2).

Change to Normal Mode from another mode:
- If a host in Mode-0 has received a Wake-up frame from a peripheral.
- If a host in Mode-2 finishes an IrDA SIR ver1.1 data communication application, but it has received a frame from NCL peripherals within 5 seconds plus 69ms.

4.4.3 IrDA-Coexistence Mode (Mode-2)

This mode is available for supporting co-operative co-existence between IrDA SIR ver1.1 and IrDA Control.

IrDA SIR ver1.1 and the IrDA Control use the same wavelength of infrared, and their signal frequencies overlap each other. Therefore, it is impossible to operate both of them simultaneously and independently. In Mode-2, time is shared between the IrDA SIR ver1.1 and the IrDA Control to enable these two applications to coexist.

If the user starts an IrDA SIR ver1.1 data communication session, the host changes to Mode-2. CL devices cannot be supported at CL polling rate in this mode. Up to two peripherals at the NCL polling rate can be supported in this mode.

A host in Mode-2 repeats the polling procedure described in paragraphs (1) and (2) as one cycle:

1. The host carries out the IrDA SIR ver1.1 data communication. After the IrDA communication period (Max. 50ms) has been completed, the host proceeds to the operation in paragraph (2).

2. The host polls all of the bound NCL peripherals in turn. If the host has data to transmit to any peripheral, it may include the data in its response request frame. Subsequently, the host returns to the IrDA SIR ver1.1 data communication described in the above paragraph (1). If any specific device has not responded to host polls for 1 second, the host unbinds that peripheral.

The polling period of the host in Mode-2 must fall within 10ms. Even if the entire polling procedure in paragraph (2) is finished in less than 10ms, the host must wait until 10ms elapse and then resumes the procedure described in paragraph (1).

The mode changes related to Mode-2 are listed below. A host will reject an ‘IrDA start’ request if the host is currently in Mode-1 and is polling peripherals at the CL Polling rate.

Change to Mode-2 from another mode:
- If a host activates an IrDA SIR ver1.1 data communication application

Change from Mode-2 to another mode:
- If a host finished the IrDA SIR ver1.1 data communication application (it enters Mode-0 or Mode-1)

For the details of coexistence between the IrDA SIR ver1.1 and the IrDA Control, see Appendix E.

4.5 Peripheral Operation

Each peripheral operates with a single MAC algorithm, regardless of the mode of the host. An IrDA Control peripheral is only allowed to transmit frames to a host when:

- The host is engaged in device hailing for enumeration (when the peripheral is not enumerated to the host).
- The host is engaged in device hailing for binding (when the peripheral is not bound to the host).
- The peripheral has received a response permission signal from the host (when the peripheral is bound to the host).
- The peripheral has listened for host's frames for 69 ms and no IrDA Control frame has been received (the peripheral sends a wakeup frame to the host).

A peripheral is not required to respond to any of the host frames. The decision to transmit a frame is entirely up to the peripheral within the constraints of the media access rule.

A simplified MAC algorithm for peripherals is shown in Figure 4.5.

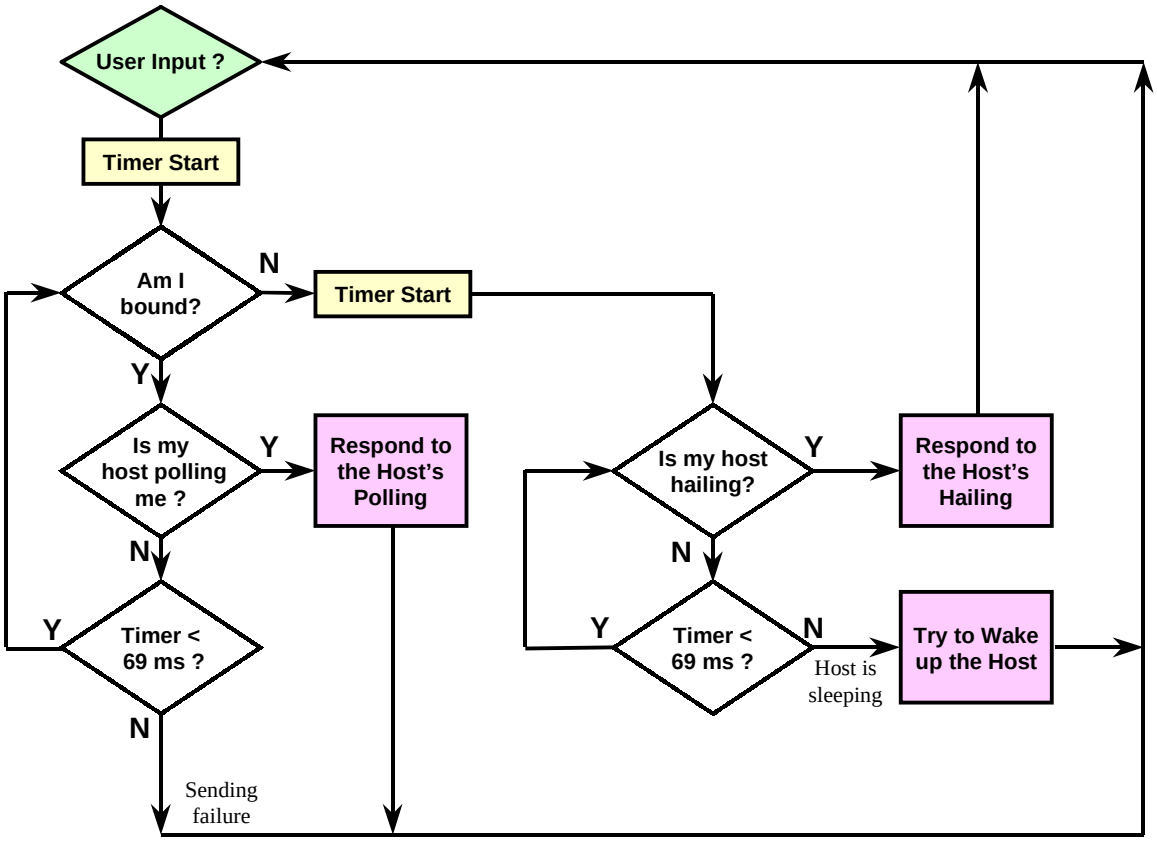


Figure 4.5 Algorithms of IrDA Control Peripheral devices

4.5.1 Host Requirements

IrDA Control hosts require the following capabilities and resources.
All fields are stored in little endian byte order (Most significant byte first) in the MAC payload.

HostID field

The HostID field is used to identify hosts that might have the same host address (HADD) field.
This field includes the information of Upper Layer Protocol with which the host can communicate. The value for D15-D4 bits can be pre-assigned, or, can be a pseudo random number which can be generated at the power-on (boot) or reset time of the host device.

The format of the HostID field is shown in Table 4.6. Reserved bits in this field must be set to 0.

Bits	Denotes	Meaning
D0	Upper Later Protocol	0 – Host does not support HID-IrDA Control Bridge 1 – Host supports HID-IrDA Control Bridge
D1		0 – Host does not support HA-LLC 1 – Host supports HA-LLC
D2		Reserved (for the 3rd protocol)
D3		Reserved (for the 4th protocol)
D4-D15	Pre-assigned number or a pseudo random number	

Table 4.6 HostID field Definitions

HostInfo field

HostInfo field indicates the capabilities of the host to the peripheral during enumeration. The format of the HostInfo field is shown in Table 4.7. Reserved bits in this field must be set to 0.

Bits	Denotes	Meaning
D0	Upper Later Protocol	0 – Host does not support HID-IrDA Control Bridge 1 – Host supports HID-IrDA Control Bridge
D1		0 – Host does not support HA-LLC 1 – Host supports HA-LLC
D2		Reserved (for the 3rd protocol)
D3		Reserved (for the 4th protocol)
D4	Support for Long frames from host	0 – Host does not support long frames from host 1 – Host supports long frames from host
D5	Support for Long frames to host	0 – Host does not support long frames to host 1 – Host supports long frames to host
D6-D15	Reserved	

Table 4.7 HostInfo field Definitions

4.5.2 Peripheral Requirements

IrDA Control peripherals require the following capabilities and resources.
All fields are stored in little endian byte order (Most significant byte first) in the MAC payload.

Peripheral ID field

They must be able to generate and store a 32 bit Physical ID (PFID). This ID is composed of a 30 bit pre-assigned or pseudo-random number and a 2-bit field to signify if it is capable of being used as a system boot keyboard or pointing device (see Table. 4.8).

Bits	Denotes	Meaning
D0-D29	Pre-assigned number or a pseudo random number	
D30	Mouse Device flag	0 – This peripheral is not a Mouse device 1 – This peripheral is a Mouse device
D31	Keyboard Device flag	0 – This peripheral is not a Keyboard device 1 – This peripheral is a Keyboard device

Table 4.8 Peripheral physical ID field Definitions

Example: PFID = b00xx xxxx xxxx xxxx xxxx xxxx xxxx xxxx for a non-boot peripheral
(e.g. gamepad or joystick).

Peripheral Info field

An IrDA Control Peripheral must have a 16-bit Peripheral Info field. This field indicates the capabilities of the peripheral to the host during enumeration. The format of the Peripheral Info field is shown in the Table 4.9 below. Reserved bits in this field must be set to 0.

Bits	Denotes	Meaning
D0	Upper Layer Protocol	0 – Peripheral does not support HID-IrDA Control Bridge 1 – Peripheral supports HID-IrDA Control Bridge
D1		0 – Peripheral does not support HA-LLC 1 – Peripheral supports HA-LLC
D2		Reserved (for the 3rd protocol)
D3		Reserved (for the 4th protocol)
D4	Support for Long frames from host	0 – Peripheral does not support long frames from host 1 – Peripheral supports long frames from host
D5	Support for Long frames to host	0 – Peripheral does not support long frames to host 1 – Peripheral supports long frames to host
D6	Max. Polling Rate	0 – NCL Polling Rate 1 – CL Polling Rate
D7-D15	Reserved	

Table 4.9 PeripheralInfo field Definitions

Example: Peripheral Info = b0000 0000 0000 0001 for game peripherals.

IrDA Control peripherals also need to be able to store a host generated 4 bit Peripheral Address (PADD) an 8 bit Host Address (HADD), and a 16-bit host ID (HostID).

Some peripherals can be enumerated to only one host: some can be enumerated to multiple hosts.

A peripheral, which can only be enumerated to one host, requires a reset of the peripheral to be enumerated to a new host. Battery removal and replacement, pushing a button, etc might accomplish the reset.
A peripheral, which can be enumerated to multiple hosts at the same time, needs a way to select hosts. A remote control unit might have a selector switch. A PDA might use its display to select a host.

4.6 Packet GAP Interval

The packet GAP interval is the time period between the end of one packet and the beginning of the next packet. The minimum specification of this value is required to allow an IR link manufactured with “Practical” components enough time to turn around. The maximum specification is required to meet the IrDA Control latency and throughput requirements. Maximum gap time is how long a host will listen for a response before it starts to poll another device. A peripheral needs to know that it has missed this time requirement, and must stay quiet until it is polled again. The following Table 4.10 shows the specifications.

Packet GAP Interval	Min.	Max.
Between the end of a host’s polling packet and the respond packet from a peripheral	10 bit times	16 bit times
Between the end of a packet from a peripheral and the next packet from a host	10 bit times	So that the resulting polling cycle will meet the requirements in each host mode
Between host packets (if there was no response to the first host packet)	20 bit times	So that the resulting polling cycle will meet the requirements in each host mode

Table 4.10 Packet GAP Interval

4.7 MAC Protocol Machine

This section describes general MAC protocol machines using the "C" programming language in order to clarify the procedures of the IrDA Control MAC protocol. The following programs do not restrict any implementation and give no assurance of actual execution. They do not cover all the specifications. For example, enumeration procedure is not mentioned here.

Common definition for MAC protocol machine definition

```
/* MAC frame */

#define MAXPAYLOAD    97

struct macframe {
        unsigned char       hostaddress;

        union {
                struct {
                        unsigned char       direction: 1;
                        unsigned char       bindtimerrestart: 1;
                        unsigned char       longframeenable: 1;
                        unsigned char       devicehailing: 1;
                        unsigned char       peripheraladdress: 4;
                } host;                     /* from Host */
                struct {
                        unsigned char       direction: 1;
                        unsigned char       pollingrequest: 1;
                        unsigned char       reserved: 2;
                        unsigned char       peripheraladdress: 4;
                } peripheral;               /* from Peripheral */
        } control;
```

```
        int         longframe;          /* TRUE if long frame */
        int         payloadlen;
        unsigned char       payload[MAXPAYLOAD];
};

#define PeripheralToHost 0
#define HostToPeripheral 1
```

4.7.3 Host MAC Protocol Machine Definition

State Definitions

MACSTATE_MODE0	Mode 0 (no peripheral bound)
MACSTATE_WAKINGUP	Going to wake up
MACSTATE_ACTIVE	Being awake
MACSTATE_POLLING	Polled(hailed) and waiting for a response
MACSTATE_IRDAWAIT	IrDA communication is carrying out

Peripheral bind states

MACPSTATE_UNBOUND	The peripheral is unbound
MACPSTATE_BINDING	The hailing response has received
MACPSTATE_BOUND	The peripheral is bound

Event Definitions

Upper layer (service primitive)

MACEVT_DATA_REQ	Data request from LLC layer
MACEVT_DATA_IND	Data indication to LLC layer
MACEVT_BIND_IND	Bind indication to LLC layer
MACEVT_UNBIND_IND	Unbind indication to LLC layer
MACEVT_NORSP_IND	No response received on polling

Lower layer (MAC frame including MAC PDU)

PHYEVT_DATA_REQ	Data request to Physical layer
PHYEVT_DATA_IND	Data indication from Physical layer

Timer module

TIMEVT_START_REQ	Timer start/re-start request to timer module
TIMEVT_EXPIRE_IND	Timer expiration indication from timer module
TIMEVT_MINGAPSTART_REQ	Minimum gap timer start request to timer module
TIMEVT_MINGAPEXPIRE_IND	Minimum gap timer expiration indication from timer module
TIMEVT_BINDSTART_REQ	Bind timer(per peripheral) start/re-start request to timer module
TIMEVT_BINDEXPIRE_IND	Bind timer(per peripheral) expiration indication from timer module

IrDA module

IRDAEVT_START_IND	IrDA communication cycle start indication
IRDAEVT_END_IND	IrDA communication cycle end indication

Variables

State	current host state
Pstate[i]	Peripheral(PADDR=i) bind state
Pdata[i]	Peripheral(PADDR=i) data request
Preceipt[i]	Receive response to a previous poll from Peripheral(PADDR=i)
HADDR	This host address
HOSTID	This host id
PADDR	current polling peripheral address

State Transitions

```
/*
```

```
        Event I/O functions
*/

/* Asynchronous event output function */
void    eventsend(int evtid, void *evtdata);

/* Asynchronous event input function */
int     eventreceive(int *evtid, void **evtdata);


/*
        Peripheral management table
*/
struct peripheraltable {
        unsigned long       pfid;
        int                 longframe;          /* TRUE if it uses a long frame */
        int                 cl;         /* TRUE if it's a CL-device */
        int                 pollingrate;
        int                 pollingcount;
        int                 responsecount;
} Peripheral[15];

#define NCL_POLLING_RATE        1
#define CL_POLLING_RATE                 2

#define MAX_POLLING_COUNT     100       /* count for 100*13.8ms = 1.38s */
#define NCLPR_TO_CLPR_THRESHOLD     90
#define CLPR_TO_NCLPR_THRESHOLD     70

int     clprperipheralnum;              /* CL polling rate peripheral */
#define MAX_CLPR_PERIPHERALS            4

void    AdjustPollingRate(unsigned char paddr);


/*
        Event data
*/

/* Service primitives */

/* MACEVT_DATA_REQ/IND */
struct macdata {
        unsigned char       paddr;
        int                 longframe;          /* TRUE if long frame */
        int                 payloadlen;
        unsigned char       payload[MAXPAYLOAD];
};


/* PDU */

/* Wakeup indication */
struct wakeupind {
        unsigned short      hostid;
};

/* Bind indication */
struct bindind {
        unsigned long       pfid;
};

/* Bind response */
struct bindrsp {
        unsigned long       pfid;
        unsigned char       paddr;
};


/* Timer */

/* Timer start */
```

```
struct timerreq {
	long	time_us;
};

/* Bind timer start */
struct bindtimerreq {
	unsigned char	paddr;
	long	time_ms;
};

#define		CLKEEPTIME	30069L
#define		NCLKEEPTIME	5069L


/* MAC main procedure for Host */

MAC_Host_main()
{
	int		evtid;
	void		*evtdata;
	int		i;

	State = MACSTATE_MODE0;
	for (i = 1; i < 9; i++)
		Pstate[i] = MACPSTATE_UNBOUND;

	clprperipheralnum = 0;

	while (1) {
		if (eventreceive(&evtid, &evtdata))
			MAC_Host_dispatch(evtid, evtdata);
	}
}


/* event dispatcher for MAC */

MAC_Host_dispatch(int evtid, void *evtdata)
{
	switch (State){
	case MACSTATE_MODE0:
		switch (evtid) {
		case PHYEVT_DATA_IND:
			State = Action11(evtdata);
			break;
		}
		break;
	case MACSTATE_WAKINGUP:
		switch (evtid) {
		case TIMEVT_MINGAPEXPIRE_IND:
			State = Action12(evtdata);
			break;
		}
		break;
	case MACSTATE_ACTIVE:
		switch (evtid) {
		case MACEVT_DATA_REQ:
			Action21(evtdata);
			/* Same State */
			break;
		case TIMEVT_BINDEXPIRE_IND:
			State = Action22(evtdata);
			break;
		case TIMEVT_EXPIRE_IND:
			State = Action23(evtdata);
			break;
		case IRDAEVT_START_IND:
			if (/* A peripheral in CL rate or
			    a peripheral using long frame is bound */) {
				RejectIrDAcommunication();
				State = State;
			} else
```

```
                                        State = MACSTATE_IRDAWAIT;
                                break;
                        }
                        break;
        case MACSTATE_POLLING:
                        switch (evtid) {
                        case  TIMEVT_EXPIRE_IND:
                                State = Action31(evtdata);
                                break;
                        case PHYEVT_DATA_IND:
                                State = Action32(evtdata);
                                break;
                        case MACEVT_DATA_REQ:
                                Action21(evtdata);
                                /* Same State */
                                break;
                        case TIMEVT_BINDEXPIRE_IND:
                                State = Action22(evtdata);
                                break;
                        }
                        break;
        case MACSTATE_IRDAWAIT:
                        switch (evtid) {
                        case MACEVT_DATA_REQ:
                                Action21(evtdata);
                                /* Same State */
                                break;
                        case TIMEVT_BINDEXPIRE_IND:
                                State = Action22(evtdata);
                                break;
                        case IRDAEVT_END_IND:
                                State = Action23(evtdata);
                                break;
                        }
                        break;
        }
}


/* MACSTATE_MODE0:PHYEVT_DATA_IND */
Action11(struct macframe *macfp)
{
        struct wakeupind     wakeupi;

        /* Check wake-up frame */
        if (macfp->hostaddress != HADDR ||
           macfp->control.peripheral.peripheraladdress != 0 ||
           macfp->control.peripheral.direction != PeripheralToHost ||
           macfp->control.peripheral.pollingrequest != TRUE ||
           macfp->payloadlen != 2)
                return State;

        pdudecode(macfp, &wakeupi);
        if (wakeupi.hostid != HOSTID)
                return State;


        /* Minimum gap timer */
        eventsend(TIMEVT_MINGAPSTART_REQ, NULL);

        return MACSTATE_WAKINGUP;
}

/* MACSTATE_WAKINGUP:TIMEVT_MINGAPEXPIRE_IND */
Action12(void *evtdata)
{
        struct timerreq         tim;
        struct macframe      macf;
        struct wakeupind     wakeupi;

        tim.time_us = 3450; /* Tss time */
        eventsend(TIMEVT_START_REQ, &tim);
```

```
	/* Send hail packet */
	macf.hostaddress = HADDR;
	macf.control.host.peripheraladdress = 0;
	macf.control.host.direction = HostToPeripheral;
	macf.control.host.bindtimerrestart = 0;
	macf.control.host.longframeenable = 0;
	macf.control.host.devicehailing = 1;
	macf.longframe = FALSE;
	macf.payloadlen = 2;
	wakeupi.hostid = HOSTID;
	pduencode(&wakeupi, &macf);
	eventsend(PHYEVT_DATA_REQ, &macf);

	PADDR = 0;

	return MACSTATE_POLLING;
}

/* MACSTATE_ACTIVE:MACEVT_DATA_REQ */
/* MACSTATE_POLLING:MACEVT_DATA_REQ */
/* MACSTATE_IRDAWAIT:MACEVT_DATA_REQ */
Action21(struct macdata *macd)
{
	/* Save data */
	Pdata[macd->paddr] = macd;
}

/* MACSTATE_ACTIVE:TIMEVT_BINDEXPIRE_IND */
/* MACSTATE_POLLING:TIMEVT_BINDEXPIRE_IND */
/* MACSTATE_IRDAWAIT:TIMEVT_BINDEXPIRE_IND */
Action22(unsigned char *paddrp)
{
	Pstate[*paddrp] = MACPSTATE_UNBOUND;

	eventsend(MACEVT_UNBIND_IND, &paddrp);

	if (/* No peripheral is bound */)
		return MACSTATE_MODE0;
	else
		return State;
}


/* MACSTATE_ACTIVE:TIMEVT_EXPIRE_IND */
/* MACSTATE_IRDAWAIT:IRDAEVT_END_IND */
Action23(void *evtdata)
{
	struct macframe	macf;
	struct bindrsp	bindr;
	struct wakeupind	wakeupi;
	struct timerreq	tim;

	if (/* No bound peripheral */)
		return MACSTATE_MODE0;

	PADDR = NextPADDRtoBePolled();

	switch (PADDR) {
	case HAILING:	/* 0 */
		tim.time_us = 3450; /* Tss time */
		eventsend(TIMEVT_START_REQ, &tim);

		macf.hostaddress = HADDR;
		macf.control.host.peripheraladdress = 0;
		macf.control.host.direction = HostToPeripheral;
		macf.control.host.bindtimerrestart = 0;
		macf.control.host.longframeenable = 0;
		macf.control.host.devicehailing = 1;
		macf.longframe = FALSE;
		macf.payloadlen = 2;
		wakeupi.hostid = HOSTID;
```

```
                pduencode(&wakeupi, &macf);
                eventsend(PHYEVT_DATA_REQ, &macf);

                return MACSTATE_POLLING;

        case NOTHING:
                tim.time_us = 3450; /* Tss time */
                eventsend(TIMEVT_START_REQ, &tim);

                return MACSTATE_ACTIVE;

        default:    /* 1 - 8 */
                switch (Pstate[PADDR]) {
                case MACPSTATE_BINDING:
                        tim.time_us = 3450; /* Tss time */
                        eventsend(TIMEVT_START_REQ, &tim);

                        macf.hostaddress = HADDR;
                        macf.control.host.peripheraladdress = 0;
                        macf.control.host.direction = HostToPeripheral;
                        macf.control.host.bindtimerrestart = 1;
                        macf.control.host.longframeenable = 0;
                        macf.control.host.devicehailing = 0;
                        macf.longframe = FALSE;
                        bindr.pfid = Peripheral[PADDR].pfid;
                        bindr.paddr = PADDR;
                        pduencode(&bindr, &macf);
                        eventsend(PHYEVT_DATA_REQ, &macf);

                        eventsend(MACEVT_BIND_IND, &PADDR);
                        Pstate[PADDR] = MACPSTATE_BOUND;

                        return MACSTATE_POLLING;

                case MACPSTATE_BOUND:
                        if (Peripheral[PADDR].longframe)
                                tim.time_us = 13050;            /* Tsl time */
                        else
                                tim.time_us = 3450; /* Tss time */
                        eventsend(TIMEVT_START_REQ, &tim);
                        macf.hostaddress = HADDR;
                        macf.control.host.peripheraladdress = PADDR;
                        macf.control.host.direction = HostToPeripheral;
                        macf.control.host.bintimerrestart = Preceipt[PADDR] ? 1 : 0;
                        macf.control.host.longframeenable = 1;
                        macf.control.host.devicehailing = 0;
                        pduencode(Pdata[PADDR], &macf);
                        eventsend(PHYEVT_DATA_REQ, &macf);

                        Peripheral[PADDR].pollingcount++;

                        return MACSTATE_POLLING;
                }
        }
}

/* MACSTATE_POLLING:TIMEVT_EXPIRE_IND */
Action31(void *evtdata)
{
        unsigned char       paddr;

        if (PADDR >= 1 && PADDR <= 8) {
                /* No response to polling */
                paddr = PADDR;
                Preceipt[PADDR] = 0;
                eventsend(MACEVT_NORSP_IND, &paddr);

                AdjustPollingRate(paddr);
        }

        /* Same action as Action23() */
        return Action23(evtdata);
```

```
}

/* MACSTATE_POLLING:PHYEVT_DATA_IND */
Action32(struct macframe *macfp)
{
        unsigned char         paddr;
        struct macdata        macd;
        struct bindind        bindi;
        struct bindtimerreq   bint;

        /* Check frame */
        if (macfp->hostaddress != HADDR ||
            macfp->control.peripheral.direction != PeripheralToHost)
                return State;

        paddr = macfp->control.peripheral.peripheraladdress;

        if (PADDR == 0) {    /* Hailing response */
                if (paddr != 0)         /* illegal */
                        return State;
                pdudecode(macfp, &bindi);
                if ((paddr = AssignPaddr(bindi.pfid)) == ERROR)
                        return State;           /* ignore ? */
                Peripheral[paddr].pfid = bindi.pfid;
                Peripheral[paddr].longframe = /* assigned by enumeration */
                Peripheral[paddr].cl = /* assigned by enumeration */
                Peripheral[paddr].pollingrate = NCL_POLLING_RATE;
                Peripheral[paddr].pollingcount = 0;
                Peripheral[paddr].responsecount = 0;
                Pstate[paddr] = MACPSTATE_BINDING;
        } else if (Pstate[paddr] == MACPSTATE_BOUND) {
                /* MAC Data indication */
                pdudecode(macfp, &macd);
                eventsend(MACEVT_DATA_IND, &macd);

                if (macfp->control.peripheral.pollingrequest == FALSE)
                        /* unbinding */
                        return Action22(&paddr);

                bint.paddr = paddr;
                if (Peripheral[paddr].cl)
                        bint.time_ms = CLKEEPTIME;
                else
                        bint.time_ms = NCLKEEPTIME;
                eventsend(TIMEVT_BINDSTART_REQ, &bint);

                Peripheral[paddr].responsecount++;
                AdjustPollingRate(paddr);
        }

        return MACSTATE_ACTIVE;
}


void   AdjustPollingRate(unsigned char paddr)
{
        struct peripheraltable              *p;

        p = &Peripheral[paddr];
        if (p->pollingcount < MAX_POLLING_COUNT)
                return;

        if (p->cl &&
            p->pollingrate == NCL_POLLING_RATE &&
            p->responsecount >= NCLPR_TO_CLPR_THRESHOLD &&
            clprperipheralnum < MAX_CLPR_PERIPHERALS) {
                p->pollingrate = CL_POLLING_RATE;
                clprperipheralnum++;
        }

        if (p->pollingrate == CL_POLLING_RATE &&
            p->responsecount < CLPR_TO_NCLPR_THRESHOLD) {
```

```
                p->pollingrate = NCL_POLLING_RATE;
                clprperipheralnum--;
        }

        p->pollingcount = 0;
        p->responsecount = 0;
}


int     NextPADDRtoBePolled()
{
        /* Mode 1
                give priority to peripherals on CL polling rate mode
                        return paddr;
                keep 13.8ms basic polling cycle
                        return NOTHING;
                periodically hailing
                        return HAILING;
        */

        /* Mode 2
                max 2 peripherals in a cycle
                        return paddr;
                periodically hailing
                        return HAILING;
        */
}
```

4.7.4 Peripheral Protocol Machine

State Definitions

MACSTATE_UNBOUND	Enumerated but not bound
MACSTATE_BOUND	Bound
MACSTATE_HAILWAIT	Waiting for host to hail
MACSTATE_WAKEUPWAIT	Waiting for host to wake up
MACSTATE_HAILRESPONDING	Going to respond to hailing
MACSTATE_BINDING	Respond to hail and waiting for host to assign PADDR
MACSTATE_BINDRESPONDING	Going to respond to binding
MACSTATE_POLLED	Polled by host

Events

Upper layer (service primitive)

MACEVT_DATA_REQ	Data request from LLC layer
MACEVT_DATA_IND	Data indication to LLC layer
MACEVT_BIND_REQ	Bind request from LLC layer
MACEVT_BIND_CNF	Bind confirmation to LLC layer
MACEVT_UNBIND_IND	Unbind indication to LLC layer

Lower layer (MAC frame including MAC PDU)

PHYEVT_DATA_REQ	Data request to Physical layer
PHYEVT_DATA_IND	Data indication from Physical layer

Timer module

TIMEVT_START_REQ	Timer start/re-start request to timer module
TIMEVT_EXPIRE_IND	Timer expiration indication from timer module
TIMEVT_MINGAPSTART_REQ	Minimum gap timer start request to timer module
TIMEVT_MINGAPEXPIRE_IND	Minimum gap timer expiration indication from timer module

Variables

State	current peripheral state

HADDR	Host address to be bound
HOSTID	Host id to be bound
PFID	32bits PFID
PADDR	Peripheral address given by host
Userdata	MAC user data to be sent
Devicetype	Indicates CL or NCL peripheral

State Transitions

```
/*
        Event I/O functions
*/

/* Asynchronous event output function */
void    eventsend(int evtid, void *evtdata);

/* Asynchronous event input function */
int     eventreceive(int *evtid, void **evtdata);


/*
        Event data
*/

/* Service primitives */

/* MACEVT_DATA_REQ/IND */
struct macdata {
        int             longframe;              /* TRUE if long frame */
        int             payloadlen;
        unsigned char           payload[MAXPAYLOAD];
};

/* MACEVT_BIND_REQ */
struct macbind {
        unsigned char           hostaddress;
        struct macdata          data;
};


/* PDU */

/* Wakeup request */
struct wakeupreq {
        unsigned short          hostid;
};

/* Bind request */
struct bindreq {
        unsigned long           pfid;
};

/* Bind confirmation */
struct bindcnf {
        unsigned long           pfid;
        unsigned char           paddr;
};


/* Timer */

/* Timer start */
struct timerreq {
        long            time_ms;
};

#define CLKEEPTIME      30000L
#define NCLKEEPTIME     5000L


/* MAC main procedure for Peripheral */
```

```
MAC_Peripheral_main()
{
        int         evtid;
        void        *evtdata;

        State = MACSTATE_UNBOUND;
        Userdata = NULL;
        Devicetype = CL or NCL;        /* depends on its characteristic */

        while (1) {
                if (eventreceive(&evtid, &evtdata))
                        MAC_Peripheral_dispatch(evtid, evtdata);
        }
}


/* event dispatcher for MAC */

MAC_Peripheral_dispatch(int evtid, void *evtdata)
{
        switch (State){
        case MACSTATE_UNBOUND:
                switch (evtid) {
                case MACEVT_BIND_REQ:
                        Action11(evtdata);
                        State = MACSTATE_HAILWAIT;
                        break;
                }
                break;
        case MACSTATE_BOUND:
                switch (evtid) {
                case PHYEVT_DATA_IND:
                        State = Action21(evtdata);
                        break;
                case TIMEVT_EXPIRE_IND:
                        Action22(evtdata);
                        State = MACSTATE_UNBOUND;
                        break;
                }
                break;
        case MACSTATE_POLLED:
                switch (evtid) {
                case MACEVT_DATA_REQ:
                        Action31(evtdata);
                        /* Same State */
                        break;
                case TIMEVT_MINGAPEXPIRE_IND:
                        Action32(evtdata);
                        State = MACSTATE_BOUND;
                        break;
                case TIMEVT_EXPIRE_IND:
                        Action22(evtdata);
                        State = MACSTATE_UNBOUND;
                        break;
                }
                break;
        case MACSTATE_HAILWAIT:
                switch (evtid) {
                case PHYEVT_DATA_IND:
                        State = Action41(evtdata);
                        break;
                case TIMEVT_EXPIRE_IND:
                        Action42(evtdata);
                        State = MACSTATE_WAKEUPWAIT;
                        break;
                }
                break;
        case MACSTATE_WAKEUPWAIT:
                switch (evtid) {
                case PHYEVT_DATA_IND:
                        State = Action41(evtdata);
```

```
                        break;
                case TIMEVT_EXPIRE_IND:
                        Action22(evtdata);
                        State = MACSTATE_UNBOUND;
                        break;
                }
                break;
        case MACSTATE_HAILRESPONDING:
                switch (evtid) {
                case TIMEVT_MINGAPEXPIRE_IND:
                        Action43(evtdata);
                        State = MACSTATE_BINDING;
                        break;
                }
                break;
        case MACSTATE_BINDING:
                switch (evtid) {
                case PHYEVT_DATA_IND:
                        State = Action61(evtdata);
                        break;
                case TIMEVT_EXPIRE_IND:
                        Action22(evtdata);
                        State = MACSTATE_UNBOUND;
                        break;
                }
                break;
        case MACSTATE_BINDRESPONDING:
                switch (evtid) {
                case TIMEVT_MINGAPEXPIRE_IND:
                        Action62(evtdata);
                        State = MACSTATE_BOUND;
                        break;
                }
                break;
        }
}


/* MACSTATE_UNBOUND:MACEVT_BIND_REQ */
Action11(struct macbind *macb)
{
        struct timerreq         tim;

        /* Save host address */
        HADDR = macb->hostaddress;

        /* Save data */
        Userdata = &macb->data;

        /* Start 69ms timer */
        tim.time_ms = 69;
        eventsend(TIMEVT_START_REQ, &tim);
}

/* MACSTATE_BOUND:PHYEVT_DATA_IND */
Action21(struct macframe *macfp)
{
        struct macdata          macd;
        struct timerreq         tim;

        /* Check data frame */
        if (macfp->hostaddress != HADDR ||
            macfp->control.host.peripheraladdress != PADDR ||
            macfp->control.host.direction != HostToPeripheral ||
            macfp->control.host.devicehailing != 0)
                return State;

        eventsend(TIMEVT_MINGAPSTART_REQ, NULL);

        /* MAC Data indication */
        pdudecode(macfp, &macd);
        eventsend(MACEVT_DATA_IND, &macd);
```

```
	if (macfp->control.host.bindtimerrestart == 1) {
		/* Re-start bound timer */
		if (Devicetype == NCL)
			tim.time_ms = NCLKEEPTIME;
		else
			tim.time_ms = CLKEEPTIME;
		eventsend(TIMEVT_START_REQ, &tim);
	}

	/* If (payloadlen == 0) it means a polling indication */

	return MACSTATE_POLLED;
}

/* MACSTATE_BOUND:TIMEVT_EXPIRE_IND */
/* MACSTATE_POLLED:TIMEVT_EXPIRE_IND */
/* MACSTATE_WAKEUPWAIT:TIMEVT_EXPIRE_IND */
/* MACSTATE_BINDING:TIMEVT_EXPIRE_IND */
Action22(void *evtdata)
{
	/* Unbound */
	eventsend(MACEVT_UNBIND_IND, NULL);
}

/* MACSTATE_POLLED:MACEVT_DATA_REQ */
Action31(struct macdata *macd)
{
	/* Save data */
	Userdata = macd;
}

/*  MACSTATE_POLLED:TIMEVT_MINGAPEXPIRE_IND */
Action32(void *evtdata)
{
	struct macframe	macf;

	/* If saved data exists, send it */
	/*
		need to handle a long frame!
	*/
	if (Userdata != NULL) {
		macf.hostaddress = HADDR;
		macf.control.peripheral.peripheraladdress = PADDR;
		macf.control.peripheral.direction = PeripheralToHost;
		macf.control.peripheral.pollingrequest = TRUE;
		pduencode(Userdata, &macf);
		eventsend(PHYEVT_DATA_REQ, &macf);
		Userdata = NULL;
	}
}

/* MACSTATE_HAILWAIT:PHYEVT_DATA_IND */
/* MACSTATE_WAKEUPWAIT:PHYEVT_DATA_IND */
Action41(struct macframe *macfp)
{
	struct wakeupreq	wakeupr;

	/* Check hailing frame */
	if (macfp->hostaddress != HADDR ||
	   macfp->control.host.peripheraladdress != 0 ||
	   macfp->control.host.direction != HostToPeripheral ||
	   macfp->control.host.bindtimerrestart != 0 ||
	   macfp->control.host.devicehailing != 1 ||
	   macfp->payloadlen != 2)
		return State;

	pdudecode(macfp, &wakeupr);
	if (wakeupr.hostid != HOSTID)
		return State;

	eventsend(TIMEVT_MINGAPSTART_REQ, NULL);
```

```
        return MACSTATE_HAILRESPONDING;
}

/* MACSTATE_HAILWAIT:TIMEVT_EXPIRE_IND */
Action42(void *evtdata)
{
        struct timerreq         tim;
        struct macframe         macf;
        struct wakeupreq        wakeupr;

        /* Start timer for host to wake up */
        tim.time_ms = 3000;             /* Time value depends on implementations
*/
        eventsend(TIMEVT_START_REQ, &tim);

        /* Wake up the host */
        macf.hostaddress = HADDR;
        macf.control.peripheral.peripheraladdress = 0;
        macf.control.peripheral.direction = PeripheralToHost;
        macf.control.peripheral.pollingrequest = TRUE;
        macf.payloadlen = 2;
        wakeupr.hostid = HOSTID;
        pduencode(&wakeupr, &macf);
        eventsend(PHYEVT_DATA_REQ, &macf);
}

/* MACSTATE_HAILRESPONDING:TIMEVT_MINGAPEXPIRE_IND */
Action43(void *evtdata)
{
        struct timerreq         tim;
        struct bindreq          bindr;
        struct macframe         macf;

        /* Start timer for bind response */
        tim.time_ms = 69;
        eventsend(TIMEVT_START_REQ, &tim);

        /* Send bind request */
        macf.hostaddress = HADDR;
        macf.control.peripheral.peripheraladdress = 0;
        macf.control.peripheral.direction = PeripheralToHost;
        macf.control.peripheral.pollingrequest = TRUE;
        macf.payloadlen = 4;

        bindr.pfid = PFID;
        pduencode(&bindr, &macf);
        eventsend(PHYEVT_DATA_REQ, &macf);

}

/* MACSTATE_BINDING:PHYEVT_DATA_IND */
Action61(struct macframe *macfp)
{
        struct bindcnf          bindc;

        /* Check bind confirmation frame */
        if (macfp->hostaddress != HADDR ||
           macfp->control.host.peripheraladdress != 0 ||
           macfp->control.host.direction != HostToPeripheral ||
           macfp->control.host.bindtimerrestart != 1 ||
           macfp->control.host.devicehailing != 0 ||
           macfp->payloadlen != 5)
                return State;

        pdudecode(macfp, &bindc);
        if (bindc.pfid != PFID)
                return State;
        PADDR = bindc.paddr;

        eventsend(TIMEVT_MINGAPSTART_REQ, NULL);
```

```
	return MACSTATE_BINDRESPONDING;
}

/* MACSTATE_BINDRESPONDING:TIMEVT_MINGAPEXPIRE_IND */
Action62(void *evtdata)
{
	struct timerreq		tim;
	struct macframe		macf;

	/* Start bound timer */
	if (Devicetype == NCL)
		tim.time_ms = NCLKEEPTIME;
	else
		tim.time_ms = CLKEEPTIME;
	eventsend(TIMEVT_START_REQ, &tim);

	macf.hostaddress = HADDR;
	macf.control.peripheral.peripheraladdress = PADDR;
	macf.control.peripheral.direction = PeripheralToHost;
	macf.control.peripheral.pollingrequest = 1;
	pduencode(Userdata, &macf);
	eventsend(PHYEVT_DATA_REQ, &macf);

	eventsend(MACEVT_BIND_CNF, NULL);
}
```

5 Logical Link Control Bridge Layer (LLC)

5.1 OVERVIEW

The Logical Link Control Layer (LLC) provides resources for reliable communication of data to and from the MAC layer. (See Figure 2.1) IrDA Control has a goal of providing components, and a protocol, that allow it to be used in a wide range of devices, with great variation in resource and cost requirements.

The **IrDA Control Logical Link Control Layer (LLC Layer)** provides the Link Layer resources used by IrDA Control devices, regardless of what higher level protocol may be used. It enables reliability through the use of lightweight protocol controlling frames. The LLC layer specifies only simple methods for acknowledgment of delivery. Therefore, it might happen that the LLC Layer by itself cannot honor an application that requires strictly reliable data communication. It is recommended that the upper layers should implement error correction functions, re-transmission functions, and so on, when assurance of reliable communication is necessary. In some cases, such as USB-HID, much of the Link Layer actually resides with the Host Operating System, and the IrDA Control LLC layer is used as a bridge to and from the MAC layer.

The LLC Layer is **not** utilized in following situations. These procedures are described in Chapter 4 (MAC).

- Enumeration procedure
- Binding procedure

The functions of LLC Layer are:

Information Features

- Send Commands
- Receive Requests
- Send Data
- Receive Data

Reliability features:

- Prevent duplicate frames by use of data sequence number [DATA0, DATA1].
- Acknowledgment of delivery based on single frame transmission (ACK).
- Re-transmission function responding to NAK or Ignore.
- Provide notice of unsupported features or inability to handle a request at this time (STALL).

5.2 LLC Frame Structure

The structure of LLC frames is shown in Figure 5.1.

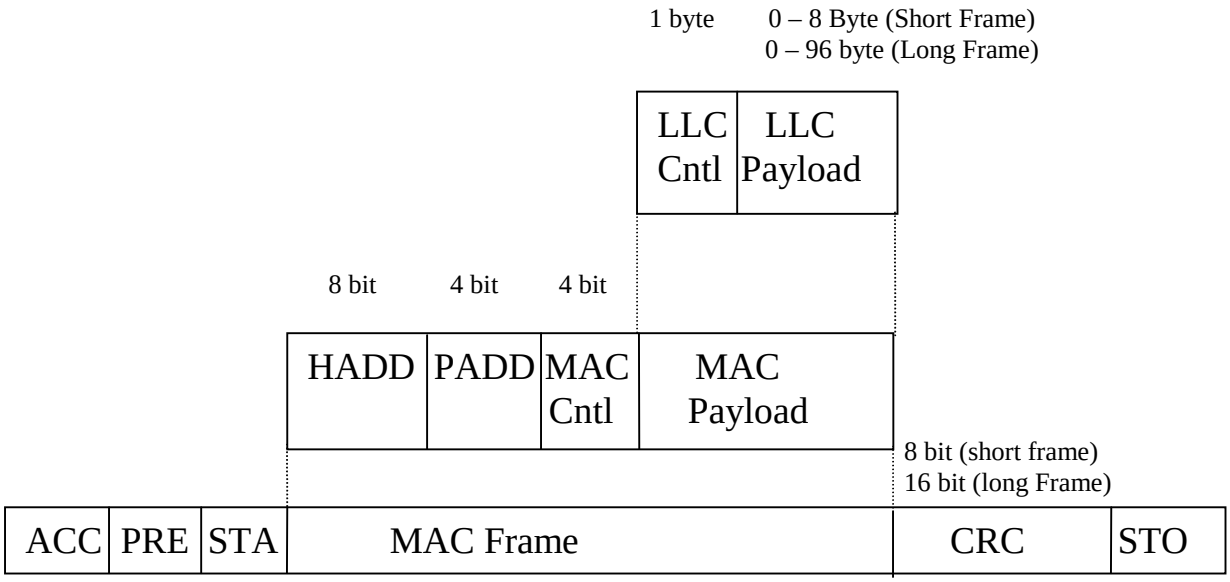


Table 5.1 HID LLC Frame Format

5.3 IrDA Control HID LLC

The control field bits used by the HID IrDA Control LLC are shown in Table 5.1.

LLC Field							
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Reserved	Endpoint		Reserved	Packet Type Code			

Table 5.1 HID LLC Control field

Stations communicating with LLC carry out the procedures of the LLC by analyzing this byte. Some of the processes of the LLC described below have dependent information included in the Payload Data of the frame.

LLC control bit D7 and D4 are reserved for future use and must be set to 0.

Hex	LLC Bits D3———D0				Packet Type	Group
0	0	0	0	0	NULL (No Command)	N/A
1	0	0	0	1	ESCAPE	Token & Data
2	0	0	1	0	IRB-Reset	Token
3	0	0	1	1	DATA0	Data
4	0	1	0	0	ACK	Handshake
5	0	1	0	1	Reserved	
6	0	1	1	0	NAK	Handshake
7	0	1	1	1	OUT/DATA0	Data
8	1	0	0	0	IN	Token
9	1	0	0	1	GET_DESCRIPTOR	Token
A	1	0	1	0	SET_MODE	Token & Data
B	1	0	1	1	DATA1	Data
C	1	1	0	0	GET_STATUS	Token
D	1	1	0	1	Reserved	
E	1	1	1	0	STALL	Handshake
F	1	1	1	1	OUT/DATA1	Data

Table 5.2 HID Packet Type Codes (Listed by Value)

Hex	LLC Bits D3———D0				Packet Type	Group
0	0	0	0	0	NULL (No Command)	N/A
2	0	0	1	0	IRB-Reset	Token
9	1	0	0	1	GET_DESCRIPTOR	
8	1	0	0	0	IN	
C	1	1	0	0	GET_STATUS	
1	0	0	0	1	ESCAPE	Token & Data
5	0	1	0	1	SET_MODE	
3	0	0	1	1	DATA0	Data
B	1	0	1	1	DATA1	
7	0	1	1	1	OUT/DATA0	
F	1	1	1	1	OUT/DATA1	
4	0	1	0	0	ACK	Handshake
6	0	1	1	0	NAK	
E	1	1	1	0	STALL	

Table 5.3 HID Packet Type Codes (Listed by Group)

The LLC Packet Codes are modeled after the USB PID (Protocol ID) codes, and are generally used in a similar fashion.

IrDA Control Endpoint		
D6	**D5**	**Description**
0	0	Control Pipe
0	1	IN or off
1	0	OUT or off
1	1	IN, OUT, or off

Table 5.4 HID Endpoint Field Encoding

Get_Descriptor IDs	
Hex	**Description**
0x01	Device Descriptor
0x02	Configuration Descriptor
0x03	String Descriptor
0x22	Report Descriptor
0x80	IrDA Control Descriptor
0xA9	Copyright Descriptor

Table 5.5 HID Get Descriptor ID Encoding

5.3.1 LLC Control Field Packet Type Descriptions

The LLC control field is described below. Details of the process of communicating with these procedures are described in Section 5.4.

ESCAPE (LLC Code 0x1)
The ESCAPE code is used to send Vendor Specific commands. The commands associated with ESCAPE are sent in the LLC Payload.

IRB-Reset (LLC Code 0x2)
This code requests a device to return to its initial reset state.

DATA0 (LLC Code 0x3)
DATA1 (LLC Code 0xB)
OUT/DATA0 (LLC Code 0x7)
OUT/DATA1 (LLC Code 0xF)
These codes are for sequenced data. DATA0 and DATA1 are from a device to a host, and OUT/DATA0 and OUT/DATA1 are from a host to a device.

ACK (LLC Code 0x4)
To avoid the loss of frames at the receiving side, this code is used to acknowledge delivery. A station that has correctly received an LLC frame transmits a delivery acknowledgment to the other station. It then expects the next LLC frame to contain a different sequence number.

NAK (LLC Code 0x6)
This code has a meaning similar to a NAK in the USB protocol. It does not mean "Negative Acknowledge" in the traditional communication sense. If a device is too busy to accept data, or has no data to send, it will send a NAK.

IN (LLC Code 0x8)
This code requests a device to send any DATA it has available. A device can Ignore this request.

GET_DESCRIPTOR (LLC Code 0x9)
This code requests a descriptor from a device. Device, Configuration, Report, IrDA Control descriptors and Copyright string are all required. The String descriptor is optional. All descriptors except the IrDA Control descriptor are the same as the USB descriptor of the same name. The IrDA Control descriptor is described in this chapter.

SET_MODE (LLC Code 0xA)
This code is used to send commands to a device. Setting the state of a keyboard LED is an example of when it would be used. See Appendices for specific usage.

GET_STATUS (LLC Code 0xC)
This code requests the status of a device. See Appendices for specific usage.
STALL (LLC Code 0xE)
This code is sent from a device to a host, and means that a host request cannot be honored.

5.3.2 LLC Control Field Endpoint and Pipe Descriptions

An IrDA Control Device can have from one to four Endpoints. An Endpoint is the address of a Pipe. A Pipe is a logical communications channel that has a type associated with it. Three types of Pipes are defined for IrDA Control devices.

The four Endpoints are:
- Endpoint 0x0 the Control Pipe, and is required for all devices.
- Endpoint 0x1 is an IN Data Pipe, and is optional.
- Endpoint 0x2 is an OUT Data Pipe, and is optional.
- Endpoint 0x3 can be an IN or OUT Data Pipe, and is optional.

The three Pipe types are:
- Control Pipe — Host Commands and Device Requests are sent via this Pipe (Endpoint zero).
- IN Pipe — Data from a device to a host uses this Pipe type.
- OUT Pipe — Data from a host to a device uses this Pipe type.

5.3.3 IrDA Control Pipe Sequence Examples

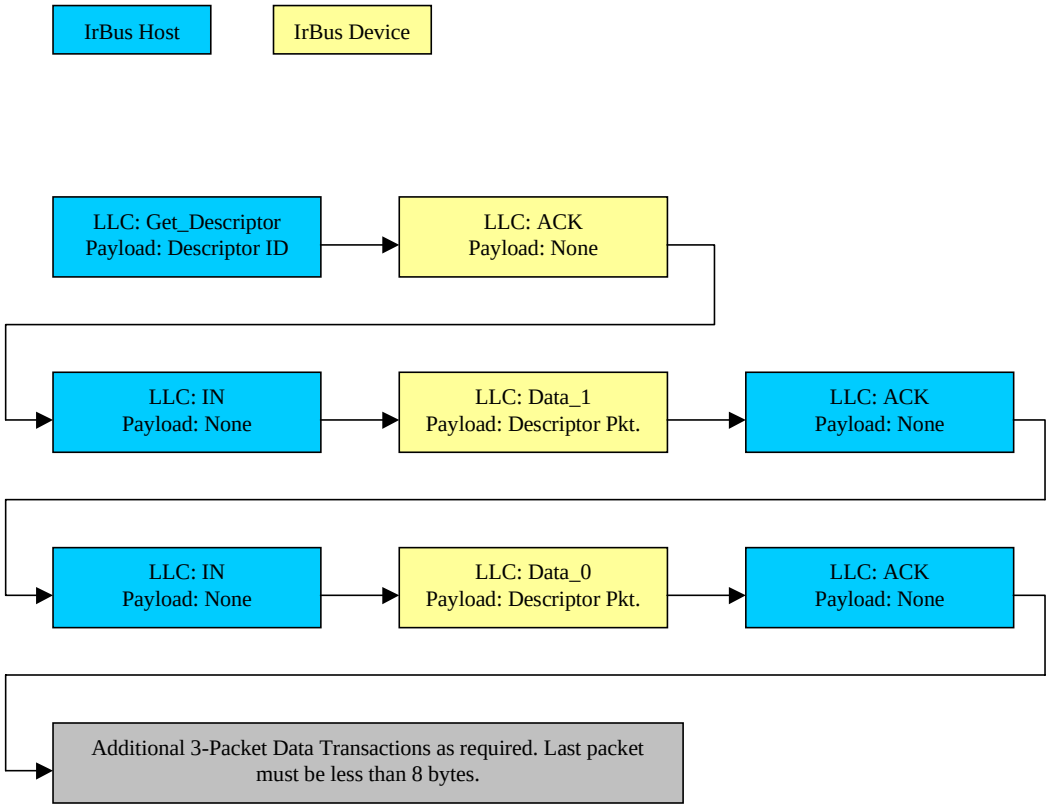


Fig. 5.2 Typical Get_Descriptor

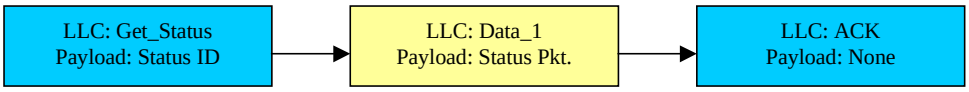


Fig. 5.3 Typical Get_Status sequence

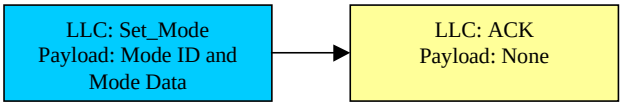


Fig. 5.4 Typical Set_Mode sequence

5.3.4 IrBus IN Data Pipe Sequence Examples

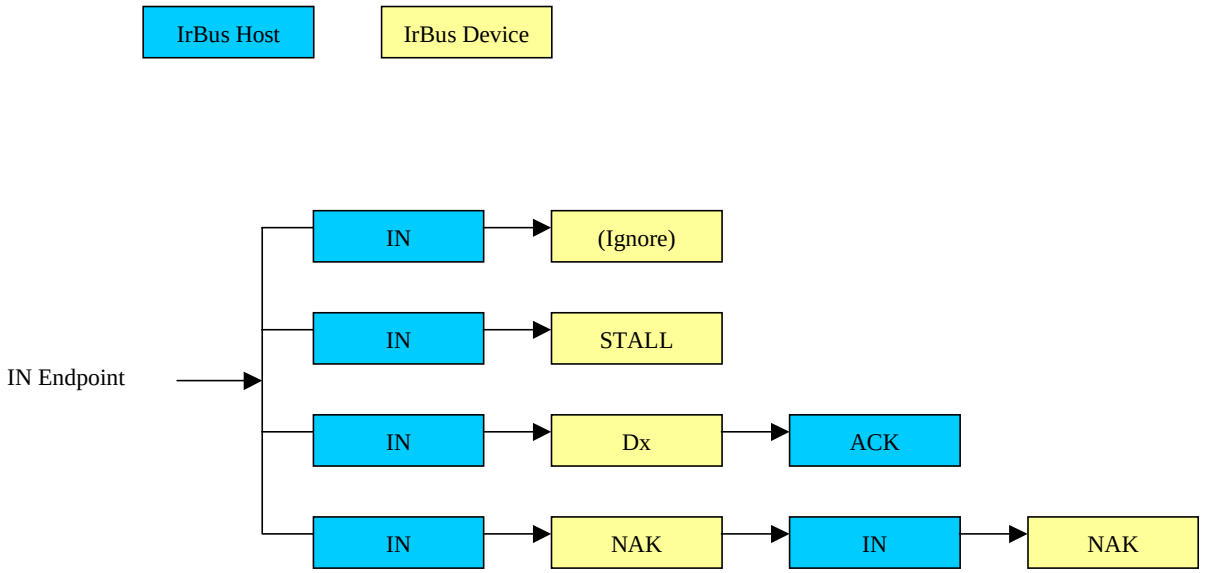


Fig. 5.5 IN Endpoint Sequences

5.3.5 IrDA Control OUT Data Pipe Sequence Examples

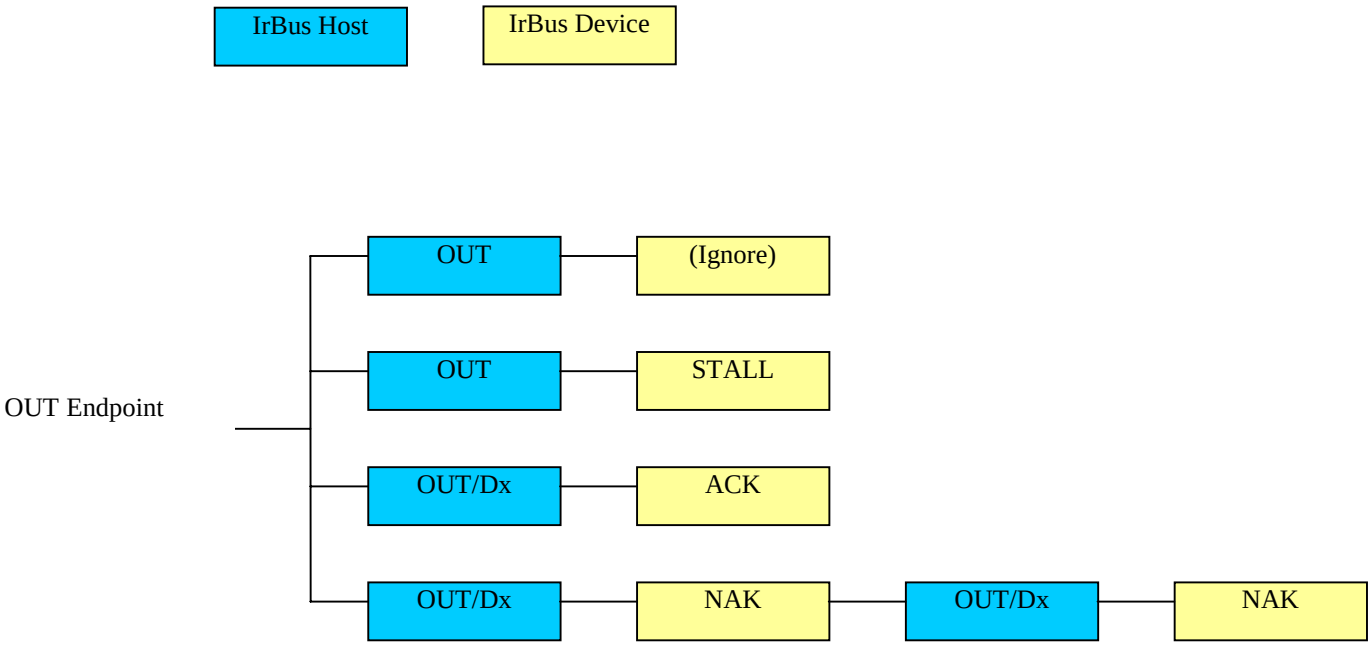


Fig. 5.6 OUT Endpoint Sequences

IrDA Control Descriptor				
Offset (dec)	**Field**	**Size**	**Value**	**Description**
0	bLength	1	Number	Size of this descriptor in bytes
1	bDescriptorType	1	Constant	IrDA Control (80h temp)
2	bcdIrBusVersion	2	BCD	IrDA Control specification release number in binary coded decimal.
4	idClass_Descriptor_1	1	Constant	Class descriptor number one ID code
5	idClass_Descriptor_2	1	Constant	Class descriptor number two ID code
6	idClass_Descriptor_3	1	Constant	Class descriptor number three ID code
7	idClass_Descriptor_4	1	Constant	Class descriptor number four ID code
8	bmEndpoint_1	1	Bitmap	Endpoint transfer type and max packet size (Endpoint 1 is either IN or off)
9	bmEndpoint_2	1	Bitmap	Endpoint transfer type and max packet size (Endpoint 2 is either OUT or off)
10	bmEndpoint_3	1	Bitmap	Endpoint transfer type and max packet size (Endpoint 3 is either IN, OUT or off)
11	bLogDevPktSize_1	1	Number	Logical device 1 packet size
12	bmLogDevAttributes_1	1	Bitmap	Logical interrupt IN device 1 attributes bitmap
13	bLogDevPktSize_2	1	Number	Logical device 2 packet size
14	bmLogDevAttributes_2	1	Bitmap	Logical interrupt IN device 2 attributes
15	bLogDevPktSize_3	1	Number	Logical device 3 packet size
16	bmLogDevAttributes_3	1	Bitmap	Logical interrupt IN device 3 attributes

Table 5.6 IrDA Control Descriptor for HID protocols

Endpoint Transfer and Packet Size Field							
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Endpoint Transfer Type		Packet Size (bytes)					

Table 5.7 HID Endpoint Transfer and Packet Size field

Endpoint Transfer Type Coding		
D7	**D6**	**Description**
0	0	OFF
0	1	IN
1	0	OUT
1	1	Reserved

Table 5.8 HID Endpoint Transfer Type Encoding

LogDevAttributes Fields							
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Boot Device Type		Link Break	Reserved	Report ID			

Table 5.9 HID LogDevAttributes

Logical Device Coding				
				Description
D7		**D6**		**Boot Device Type**
0		0		None
0		1		Keyboard
1		0		Mouse
1		1		Reserved
D5				**Link Break Action**
0				No Action on Link Break
1				Send Packet of Zeros on Link Break
D4				**Reserved**
D3	D2	D1	D0	**Report ID**
1	0	0	0	HID
x	x	x	x	Reserved

Table 5.10 HID Logical Device field Coding

Communicating with an IrDA Control peripheral can be very simple. Using a mouse as an example, and allowing that the peripheral is responding with ACKs, DATAx's, et al. The basic sequence is :

- Send a Get_Descriptor command to Endpoint zero asking for the Report Descriptor.
- Send IN LLC Codes to Endpoint zero until you have read the entire Report Descriptor.
- Decode the Report Descriptor (complex). On a Windows USB-PC, it would be done by hidclass.sys.
- Poll the device with the IN LLC code on Endpoint 1 whenever you want mouse input.

Appendix A. Spectrum of IrDA Control Signal

Figure A.1 shows the spectrum of the IrDA Control 16 PSM signal multiplied by 1.5MHz subcarriers (in a frequency band of –0.5MHz to +2MHz).

As is apparent from Figure A.1, the 16PSM scheme has low energy in the frequency band of around 33kHz to 40kHz, which is used for Remote Control Systems, and therefore is able to reduce the interference between an IrDA Control system and a Remote Control System.

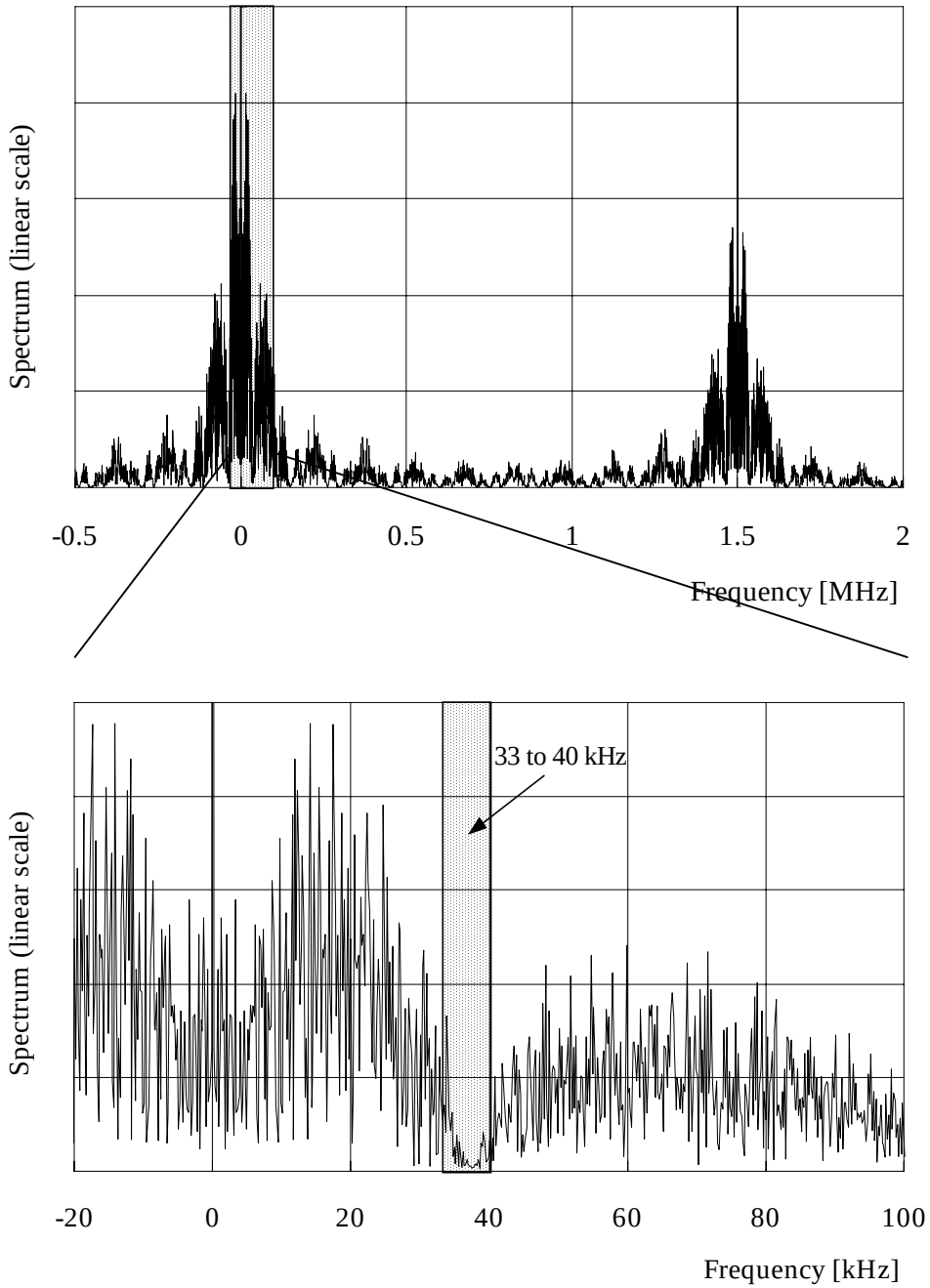


Figure A.1 : Spectrum of 16PSM Signal

Appendix B. Example of Link Budget Analysis

An example of link budget analysis is shown in Table B.1. This example shows the case of communication between Peripheral Type 1 Device and Host Device on axis.

		MIN	MAX	How to Calculate
SPECIFICATIONS				
Maximum Link Length, m	a	5.00		
Minimum Link Length, m	b		0.20	
Intensity In Angular Range, mW/sr	c	100	500	
Irradiance In Angular Range, µW/cm^2	d	0.400	1250	
Sunlight Ambient Irradiance, µW/cm^2	e		100	
Bit Error Rate	f		1.0.E-04	
Required Signal-to-Noise Ratio for BER	g	7.40		
RECEIVER DATA (Not Specifications)				
Detector Sensitivity, A/W	h	0.40		
Effective Detector Area, cm^2	i	0.172		
Receiver Input Noise Current Density, pA/(BW)^0.5	j	2.00		
Receiver 3dB Bandwidth, MHz	k	1.40	1.60	
CALCULATED PERFORMANCE				
Sunlight Photo Current, µA	l	6.88		$=e_{max}*h_{min}*i_{min}$
Sunlight Noise Current Density, pA/(BW)^0.5	m	1.48		$=(2*1.6e\text{-}19*l_{min})^{0.5}$
Sunlight Noise Current, nA	n	0.66		$=m_{min}*(k_{max}-k_{min})^{0.5}$
Receiver Input Noise Current, nA	o	0.89		$=j_{min}*(k_{max}-k_{min})^{0.5}$
Total Receiver Noise Current, nA	p	1.11		$=(n_{min}^2+o_{min}^2)^{0.5}$
Received Signal Current, nA	q	27.52		$=d_{min}*h_{min}*i_{min}$
Received Signal to Noise Ratio	r	24.71		$=q_{min}/p_{min}$
Margin (min. S/N)/(Spec. S/N), dB	s	5.24		$=10\log(r_{min}/g_{min})$

Table B.1 Example of Link Budget Analysis

Appendix C. IEC 825-1 Class 1 Eye Safety Compliance

The October 1993 edition of IEC 825-1 includes LEDs along with lasers. The standard requires classification of the Allowable Emission Limits (AEL) of all final products. AEL refers to the level of ultraviolet, visible or infrared electromagnetic radiation emitted from a product to which a person could be exposed. Any product which emits radiation in excess of AEL Class 1 must be labeled (a hazard symbol and an explanatory label would be required). Class 1 products must only be declared as such within the product literature.

This appendix summarizes IEC Class 1 AEL requirements relevant to IrDA Control and provides a table of maximum transmitter intensity for a range of corresponding source sizes. The intensity is calculated for a CW or DC condition and can be adjusted by dividing by the maximum duty cycle as appropriate.

Generally relevant issues include:

Components are not subject to IEC 825-1, only final products are.
Classification is to include the effects of any reasonably foreseeable single fault condition, process, lifetime and temperature variations.

Specific issues for IrDA Control include:

Of the three measurement conditions referenced within IEC 825-1, single pulse, pulse train and average power, average power is the most restrictive and used here.
Calculations are for a single source. Implementations with multiple sources are expected to have separations of > 10 mm between sources which permits treatment as independent sources.
Calculations are for a test time of 100 seconds and a wavelength of 875 nm.

The following Table C.1 and Figure C.1 shows the source size, D, maximum power level, Pmx, associated solid angle for maximum power, A_{pmx}, and maximum intensity for Class 1, P_{mx}/A_{pmx}.

D [mm]	P_{mx} [mW]	A_{pmx} [sr]	P_{mx}/A_{pmx} [mW/sr]
0.10	0.495562	0.2481	2.00
0.20	0.495562	0.1508	3.29
0.50	0.495562	0.0693	7.15
1.00	0.495562	0.0365	13.59
1.50	0.675766	0.0247	27.31
2.00	0.901022	0.0187	48.12
2.50	1.126277	0.0151	74.78
3.00	1.351533	0.0126	107.29
4.00	1.802044	0.0095	189.88
5.00	2.252555	0.0076	295.89
6.00	2.703066	0.0064	425.30
7.00	3.153577	0.0055	578.13
8.00	3.604088	0.0048	754.37
9.00	4.054599	0.0042	954.02
10.00	4.505110	0.0038	1177.09

Table C.1: Accessible Emission Limits

Where,

D, source size, is defined as the diameter of the circle containing 63% of the source power.

As can be seen in the table C.1, for a maximum output of 500 mW/sr Class 1, CW operation requires a source size greater than 6.5 mm or greater than 4.6 mm with a maximum duty cycle of 50%. To support a source size of 3.0 mm, the maximum duty cycle must be kept below 21%.

$$P_{mx}\ (\mathrm{W}) = 7 \times 10^{-4} \times 10^{0.002\,(wl\,-\,700)} \times C_6 \times t^{-0.25}$$

$$A_{pmx}\ (\mathrm{sr}) = 2\pi \left(1 - \cos \left(\arctan \left(\frac{3.5}{100 \sqrt{\frac{D}{10} + 0.0046}} \right) \right) \right)$$

$$wl\ (\mathrm{nm}) = 875 \quad \mathrm{nm}$$

$$C_6 = 1 \qquad \text{for} \quad D \le 1.1\,\mathrm{mm}$$

$$= \frac{D}{1.1} \qquad \text{for} \quad 1.1 < D \le 10\ \mathrm{mm}$$

$$= \frac{10}{1.1} \qquad \text{for} \quad D > 10\ \mathrm{mm}$$

$$t(s) = 100 \quad \mathrm{s}$$

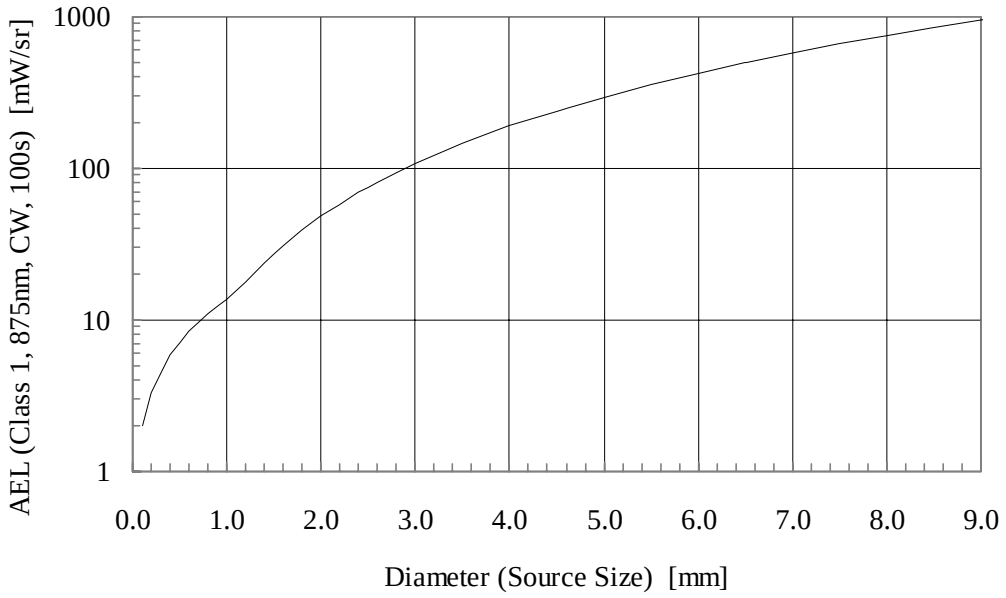


Figure C.1 : Eye Safety: IEC 825-1 Class 1 Accessible Emmission Level
(875 nm LED, 100 s Exposure) vs Source Size

Appendix D. Examples of Packet Traffic Profile

D.1 Principle

Figure D.1 shows the principle of packet exchange in MAC protocol. The host gives the timing for each peripheral to speak to the host.

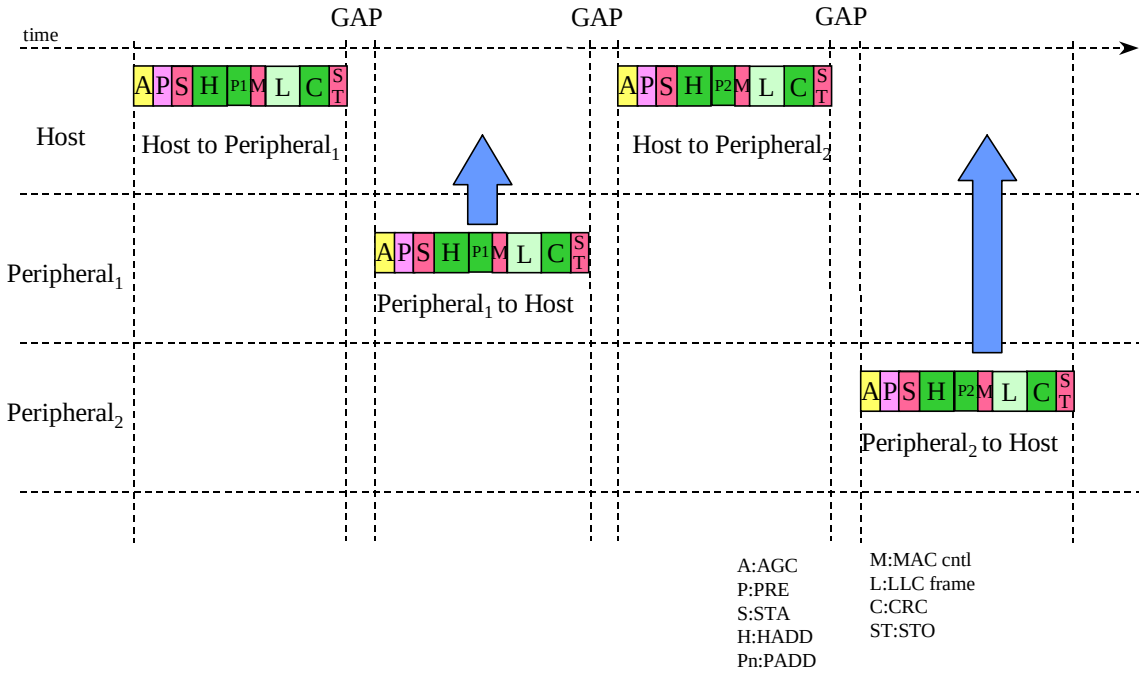


Figure D.1 Packet traffic principle

D.2 Mode-1

In Mode-1 the host polls the bound peripherals, and then performs hailing. If all polling procedures for each bound peripheral are completed within a basic polling cycle, the host waits for the basic polling cycle time to elapse. Figure D.2 shows the example of this case.

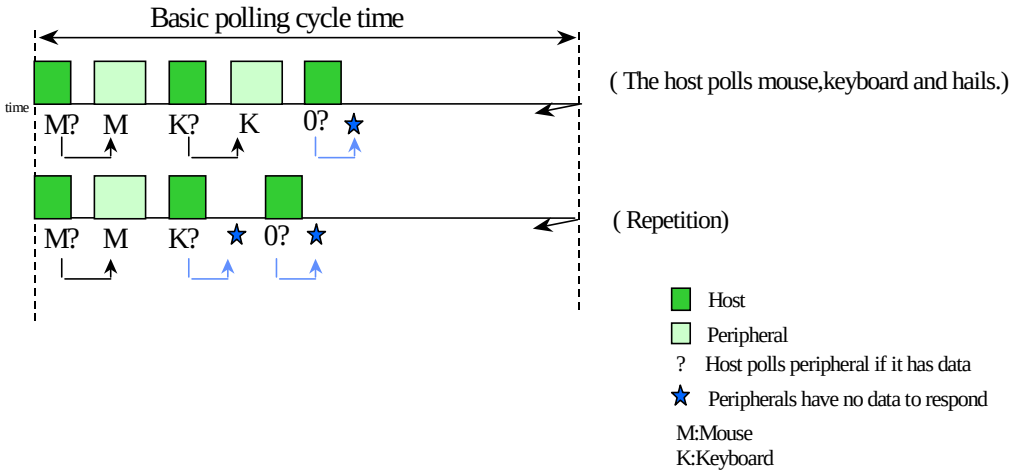


Figure D.2 Example of packet traffic in Mode-1 (within the basic polling cycle time)

In Mode-1 when any CL peripheral at the CL polling rate is bound, the polling procedure differs from the case of no CL peripheral at the CL polling rate. Newly bounded peripheral is always polled at the NCL polling rate. When a CL peripheral at the NCL polling rate has responded equal to or more than specified number of times within the latest 100 times polling from the host, the host polls the peripheral at the CL polling rate. When a CL peripheral at the CL polling rate has responded less than specified number of times within the latest 100 times polling from the host, the host polls the peripheral at the NCL polling rate. Figure D.3 shows the packet traffic when the host changes the polling rate of the joystick.

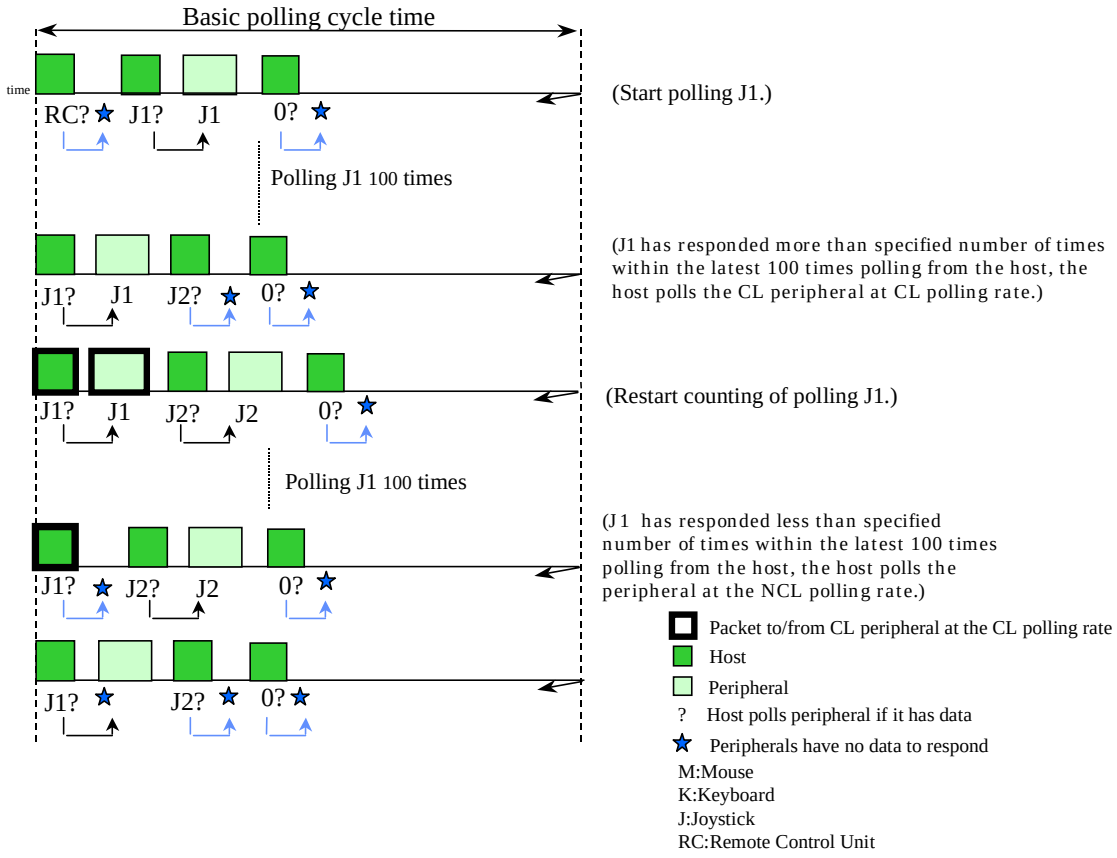


Figure D.3 Example of packet traffic in Mode-1 (within the basic polling cycle time)

If no CL peripheral at the CL polling rate is bound, the host polls all of the bound peripherals and then performs hailing in a polling cycle. The entire polling cycle may be longer than the basic polling cycle time.

If any CL peripheral at the CL polling rate is bound, the host ensures one polling in every basic polling cycle time for the CL peripheral at the CL polling rate.
One example of such a case is shown in Figure D.4.

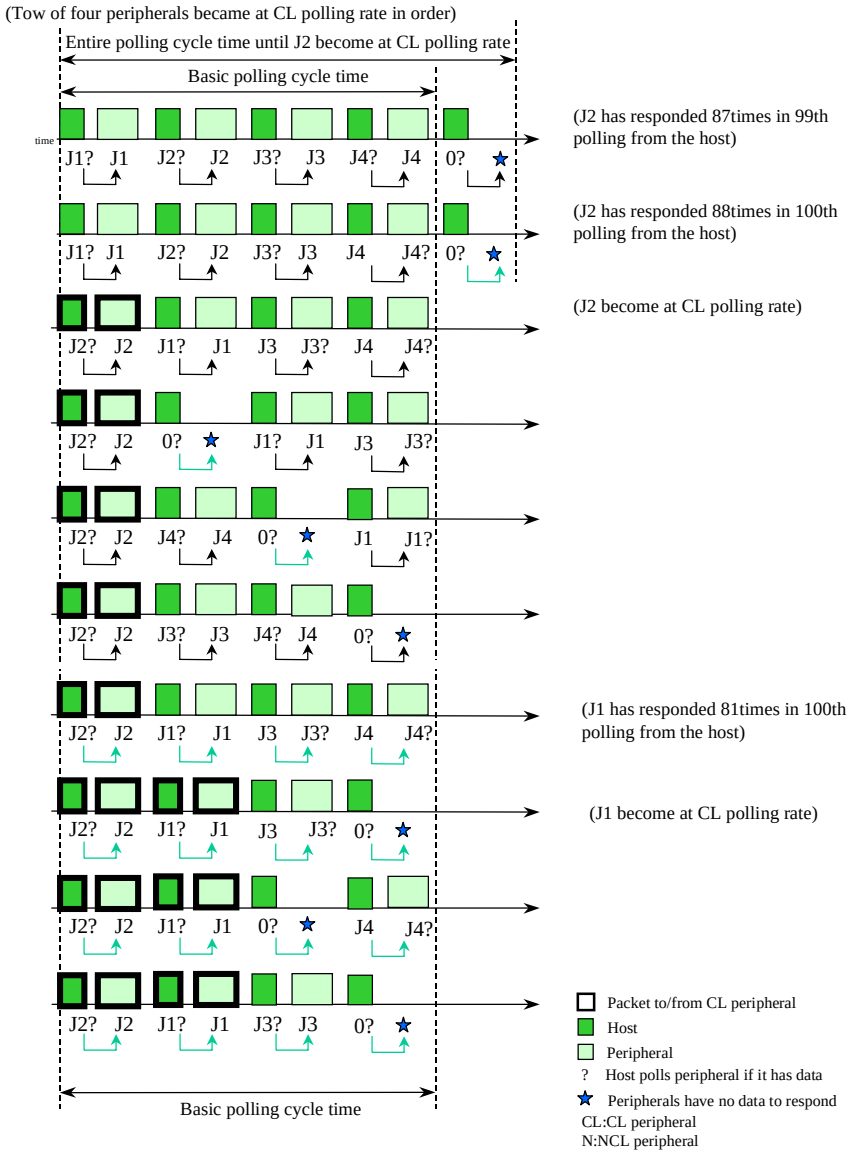


Figure D.4 Example of packet traffic in Mode-1(when some peripheral become at CL polling rate)

Figure D.5 shows that when a CL peripheral at the CL polling rate become at the NCL polling rate.

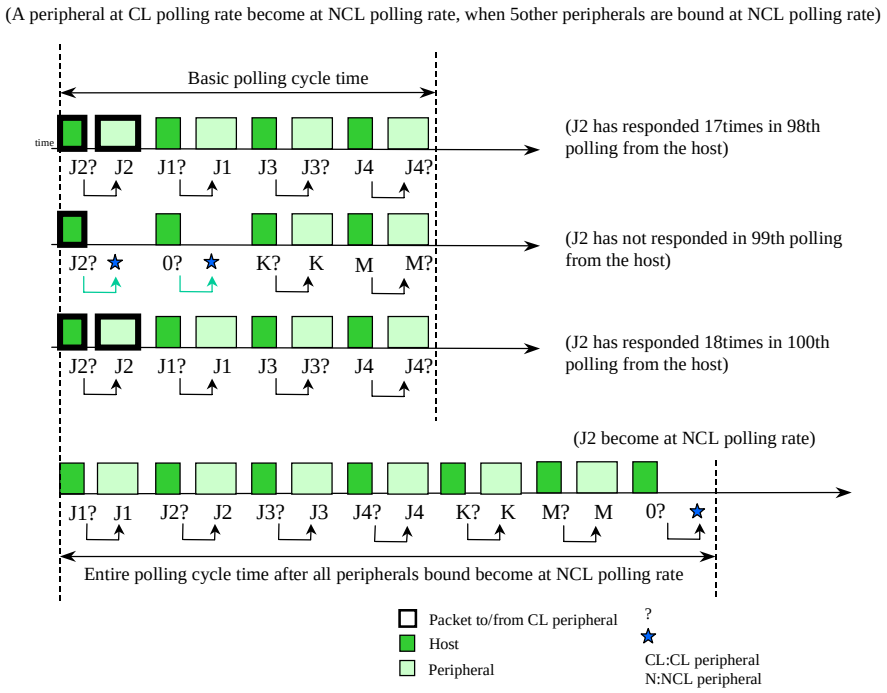


Figure D.5 Example of packet traffic in Mode-1 (when all peripherals bound are become at NCL polling rate)

[No CL peripheral at the CL polling rate]

When no CL peripheral at the CL polling rate is bound in Mode-1, some long packets can be included, and up to 8 peripherals can be polled in one polling cycle. Therefore, the entire polling cycle may be longer than the basic polling cycle time. Figure D.6 shows the example of this case.

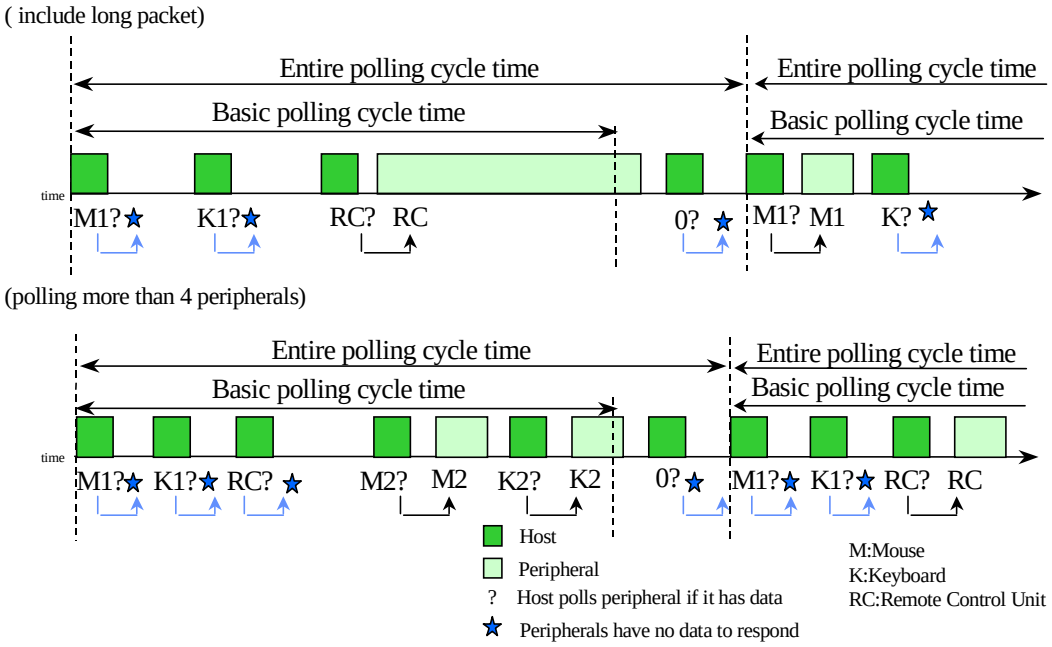


Figure D.6 Example of packet traffic in Mode-1 (longer than the basic polling cycle time)

The long packet can be transmitted from the host as well as from the peripheral. Figure D.7 shows an example of such 1 device polling.

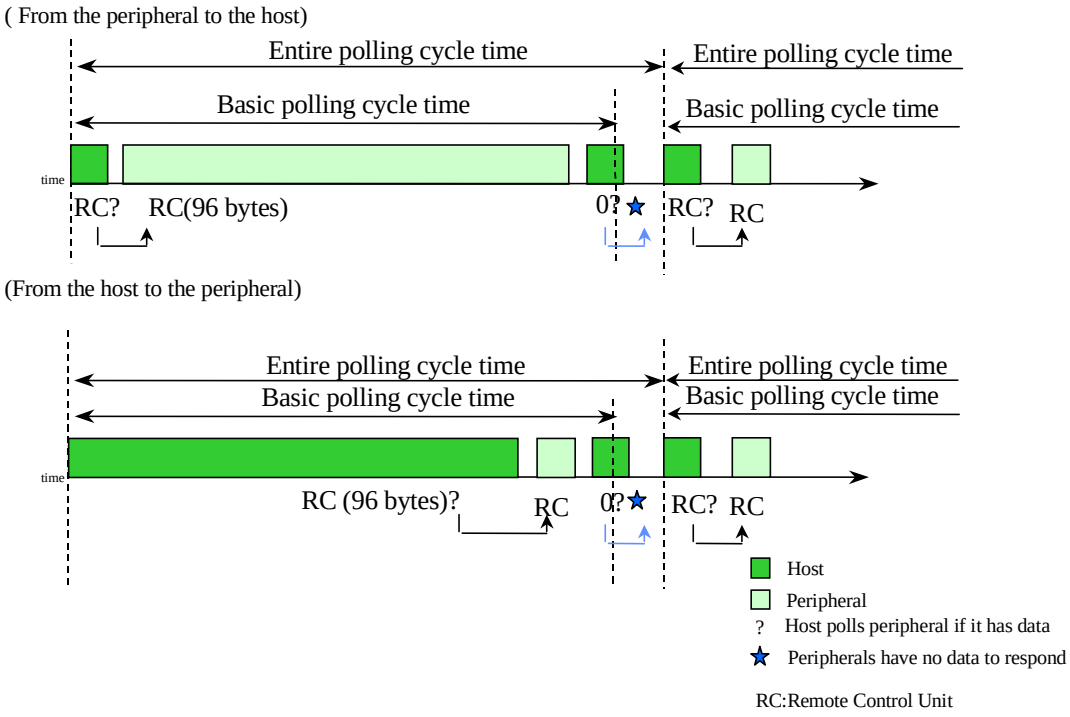


Figure D.7 Example of packet traffic of maximum long packet

[Some CL peripherals at the CL polling rate]

When some CL peripherals at the CL polling rate are bound in Mode-1, Long packet is prohibited, and four CL peripherals at the CL polling rate at maximum can be polled within the Basic polling cycle time. All CL peripherals at the CL polling rate are polled first (since priority is given to the polling of CL peripherals at the CL polling rate), and then the remaining time is taken for polling to the peripherals at the NCL polling rate and hailing.

The polling to the peripherals at the NCL polling rate or hailing which has not been performed in the first Basic polling cycle is sequentially performed in subsequent Basic polling cycles. If the host has bound the maximum number of peripherals, hailing is not performed. Otherwise, hailing is performed after polling all the peripherals at the NCL polling rate.
Figure D.8 shows the packet when CL communication takes place between two peripherals at the CL polling rate and two peripherals at the NCL polling rate.

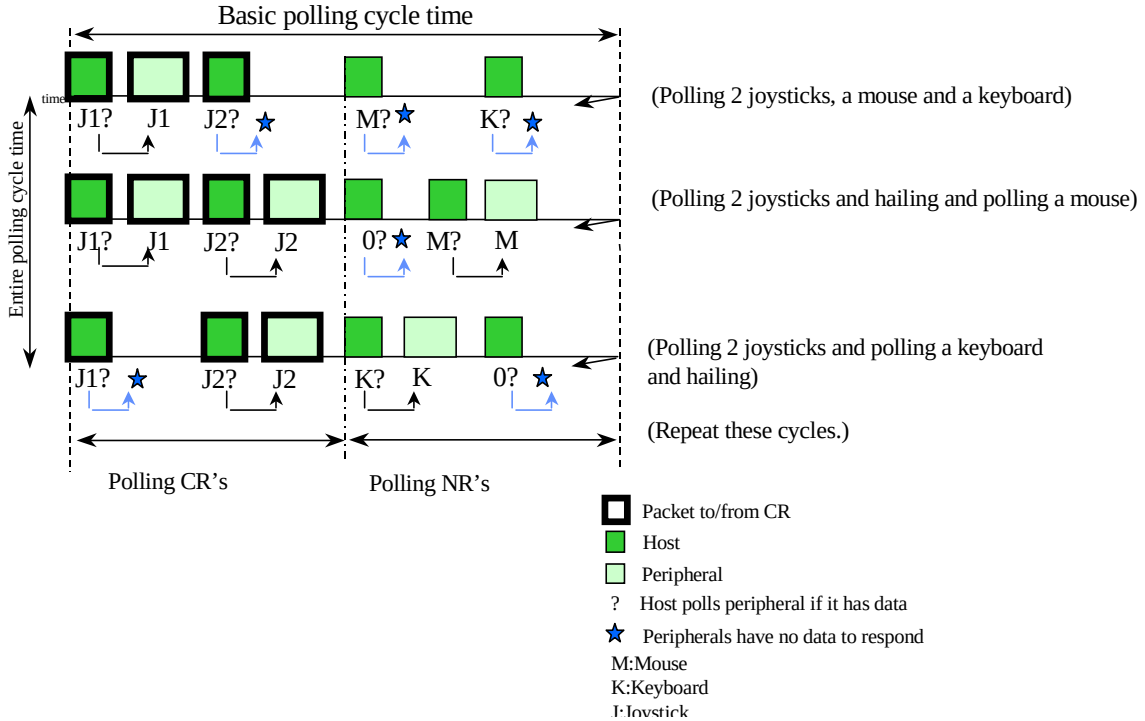


Figure D.8 Example of packet traffic in Mode-1 (2 CL and 2 NCL)

When the number of CL peripherals at the CL polling rate is one, the maximum number of peripherals at the NCL polling rate, which can be bound, is twelve. In this case, the CL peripheral at the CL polling rate is polled in each Basic polling cycle, and the peripherals at the NCL polling rate are sequentially polled in the remaining the Basic polling time. Figure D.9 shows the packet traffic thereof.

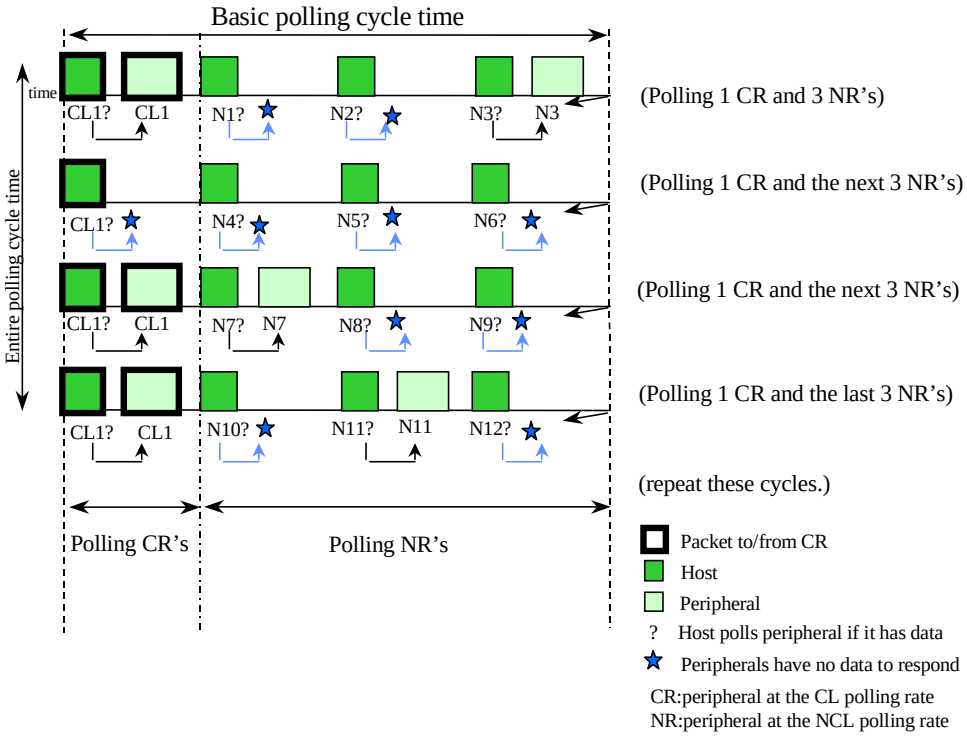


Figure D.9 Example of packet traffic in Mode-1 (1 CL and maximum NCL)

When the number of CL peripherals at the CL polling rate is two, the maximum number of peripherals at the NCL polling rate, which can be bound, is eight. In this case, the CL peripheral at the CL polling rate is polled in each Basic polling cycle, and the peripherals at the NCL polling rate are sequentially polled in the remaining of the Basic polling time. Figure D.10 shows the packet traffic thereof.

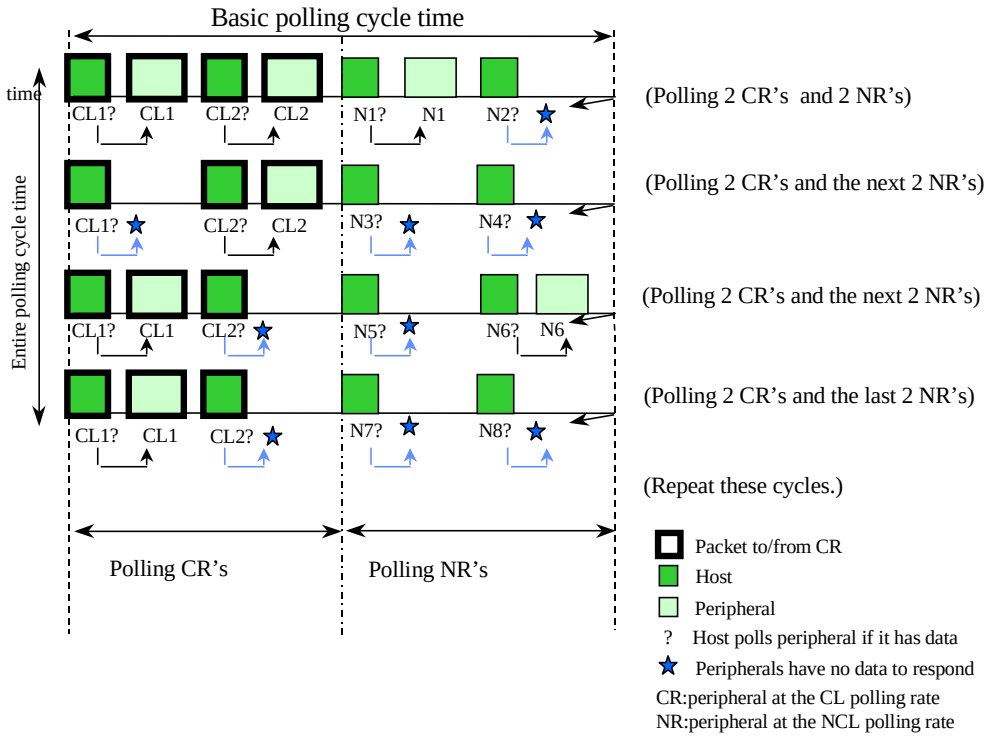


Figure D.10 Example of packet traffic in Mode-1 (2 CL and maximum NCL)

When the number of CL peripherals at the CL polling rate is three, the maximum number of peripherals at the NCL polling rate, which can be bound, is four. In this case, the CL peripheral at the CL polling rate is polled in each Basic polling cycle, and the peripherals at the NCL polling rate are sequentially polled in the remaining of the Basic polling time. Figure D.11 shows the packet traffic thereof.

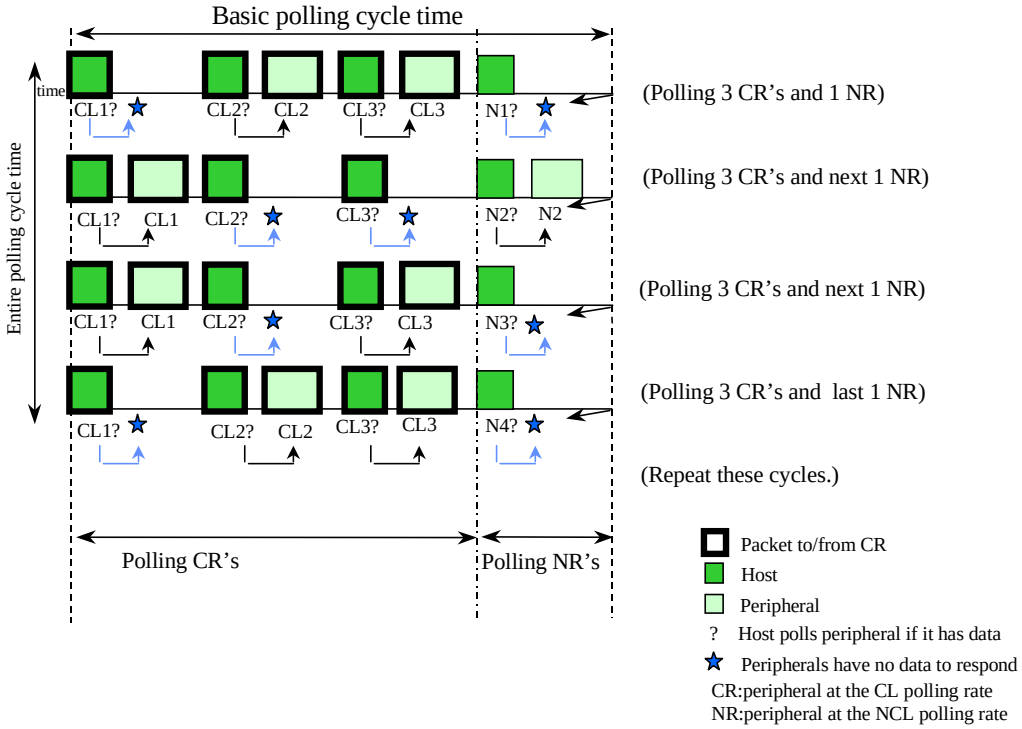


Figure D.11 Example of packet traffic in Mode-1 (3 CL and maximum NCL)

When the number of CL peripherals at the CL polling rate is four, the maximum number of peripherals at the NCL polling rate, which can be bound, is one. In this case, four CL peripherals at the CL polling rate are polled at the Basic polling cycle, whereas one peripheral at the NCL polling rate is alternately polled or hailed in one of four cycles. Figure D.12 shows the packet traffic in this case.

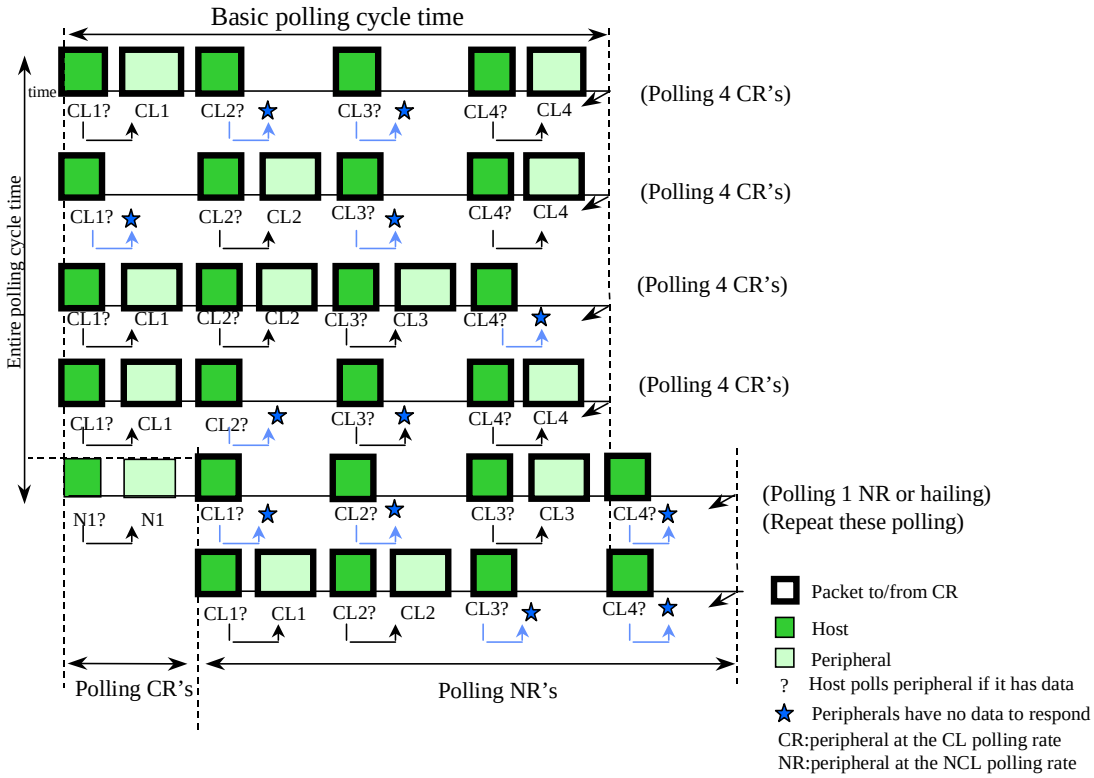


Figure D.12 Example of packet traffic in Mode-1 (4 CL and maximum NCL)

D.3 Mode-2

In Mode-2, the traffic of IrDA SIR Ver1.1 and the traffic of IrDA Control are performed by turns. Figure D.13 shows the packet traffic in Mode-2.

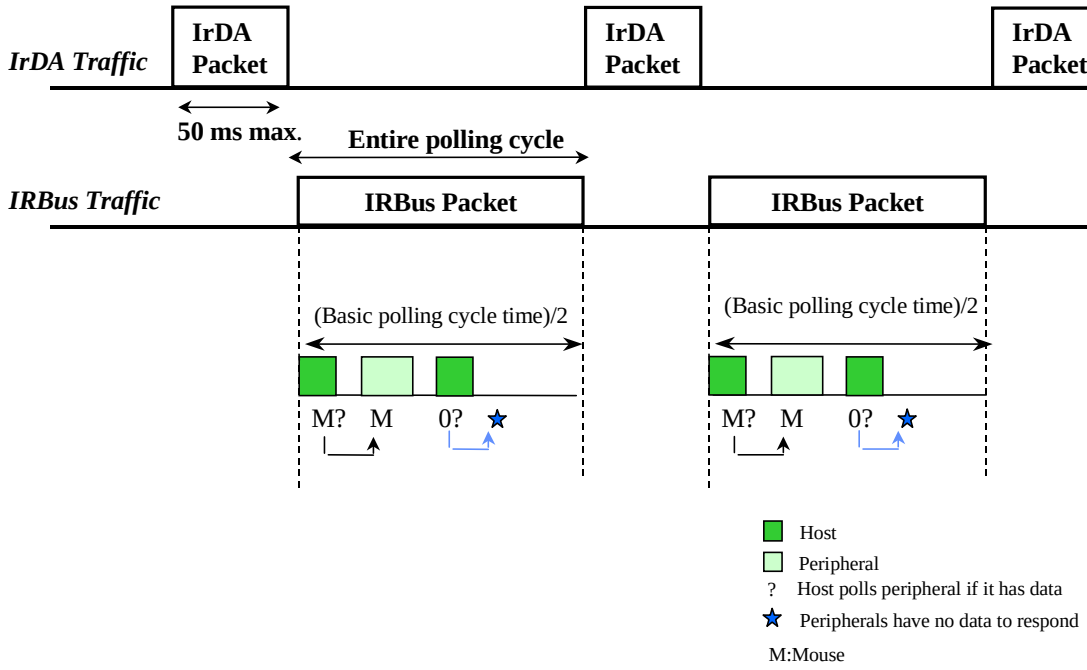


Figure D.13 Example of packet traffic in Mode-2

D.4 Binding

A binding procedure is normally performed after the completion of the polling procedure of all the peripherals bound to the host. However, the binding procedure is not performed if the peripherals already bound to the host amounts to the maximum number permissible for binding.

Figure D.14 shows the example of the case where the binding procedure is performed.

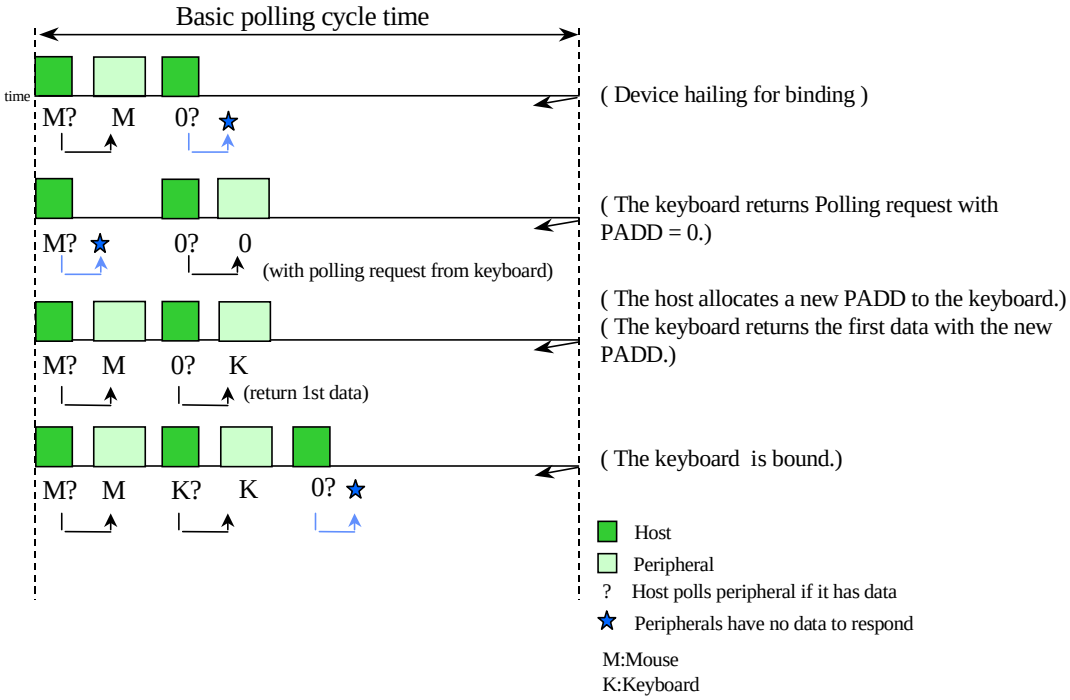


Figure D.14 Example of packet traffic in binding procedure

Figure D.15 shows the example of the case where binding and unbinding are performed.

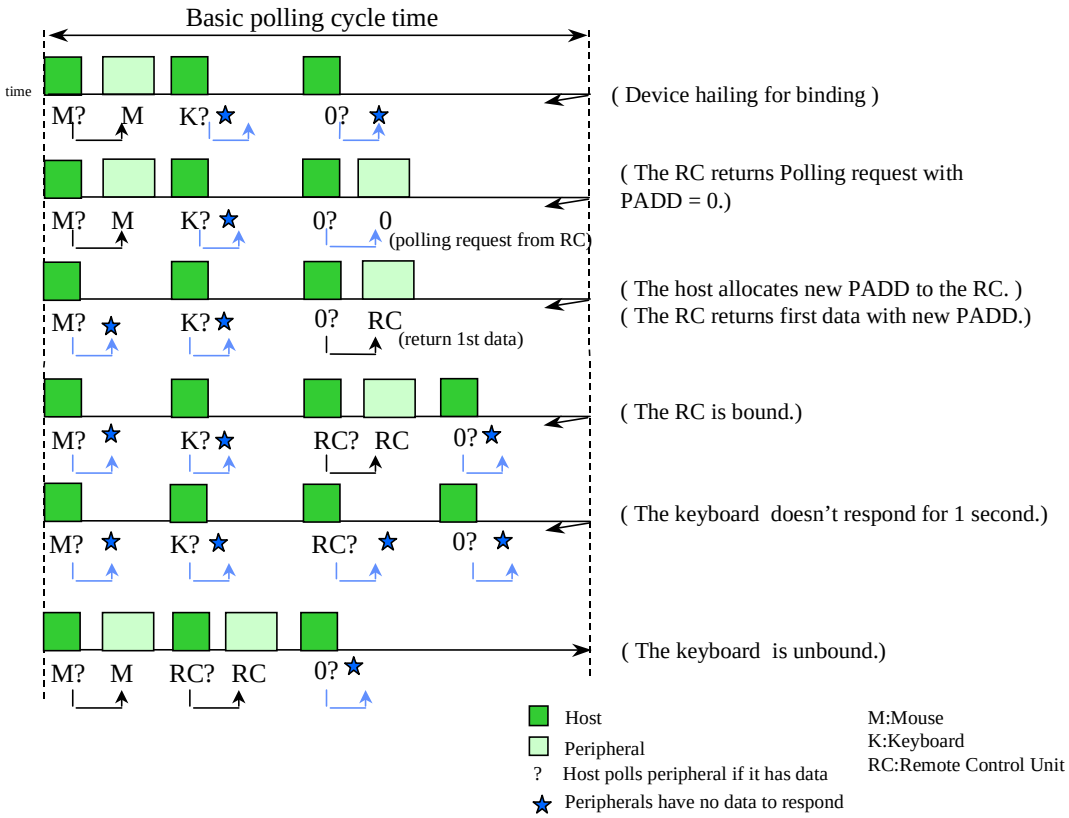


Figure D.15 Example of packet traffic with bind and unbind

Appendix E. IrDA Coexistence

E.1 Introduction

IrDA Control devices and IrDA devices interfere with each other, when they are simultaneously operated in the same user space. This means that, in absence of a coexistence scheme, an IrDA Control mouse cannot be used to control a data transfer from an IrDA camera to a personal computer. The interference between IrDA signals and IrDA Control signals is due to overlap of the two signals in the frequency domain, and cannot be eliminated at a physical level. However, a time-division-multiplexing scheme that allows sharing of the IR medium, would allow the two protocols to coexist. This appendix describes such a time-sharing scheme, the IrDA coexistence scheme, which operates at the link layer.

E.2 IrDA Coexistence scheme

In simple terms, the IrDA coexistence scheme allows IrDA and IrDA Control devices to alternately control the infrared medium. The access to the IR medium is coordinated between the two devices by exchange of messages.

In the IrDA coexistence scheme, the IrDA primary creates 'quiet' time in the IR medium for IrDA Control transfers. Further, while in coexistence mode, the IrDA Control device is allowed to transmit only during these 'quiet' times. The IrDA Control link layer is informed of the 'quiet' times by the IrDA link layer[1], namely IrLAP.

The link layer of an IrDA station, namely IrLAP, operates in one of the two modes: Normal Response Mode (NRM[2]) or Normal Disconnect Mode (NDM[3]) corresponding to the state of its data link channel.
The IrDA coexistence scheme is applicable to both the link modes. The modifications to the IrLAP stack on the IrDA primary needed to implement the coexistence scheme in NRM as well as in NDM is discussed below.

Note that the IrLAP is an asymmetric protocol. The IrDA node designated as the 'IrDA primary' controls data flow as well as accesses to the medium for all the nodes in the connection. Hence, the IrDA coexistence solution assumes that the IrDA Control device is located on the same host as the IrDA primary.

E.2.1 Coexistence in NRM

IrDA coexistence scheme allows IrDA Control transfers by creating 'quiet' periods in the IrDA transmissions. IrDA connection mode behavior is modified to create these 'quiet' periods. Figure E.1 shows the basic IrDA coexistence scheme in NRM. In this scheme, the data transmissions from the IrDA primary are delayed for 10ms to create a window for IrDA Control transfers.

[1] Coordination of traffic at the MAC layer is not possible, since the MAC layer of IrBus is implemented in hardware to meet the IrBus latency requirements.

[2] An IrDA device is in NRM when the device has established an IrDA connection with one or more IrDA nodes, and is able to exchange control/data frames.

[3] An IrDA device is in NDM when the device does not have any active connections.

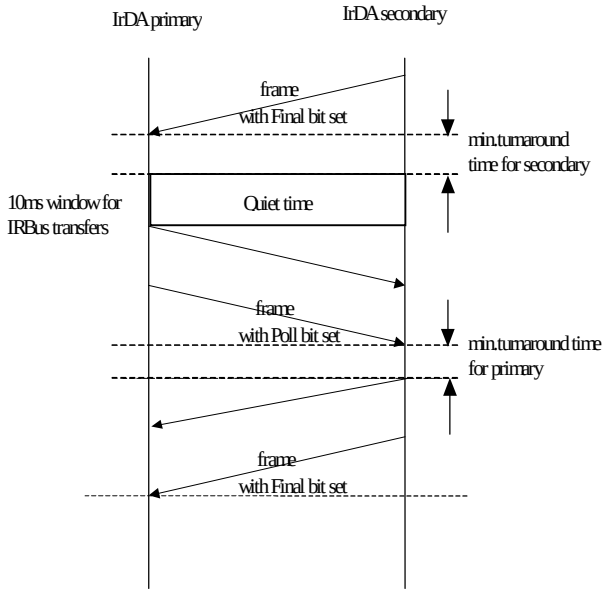


Figure E.1 IrDA coexistence scheme

One of the IrDA connection parameters is the Maximum turn around time, which determines the duration for which an IrDA station can hold a transmit permission. A typical value for this parameter is 500ms, which results in one IrDA Control transfer every 1second (500ms + 500ms). In order to improve the response time of IrDA Control, the maximum turn around time should be set to the lowest possible value of 50ms. The degradation of the IrDA link caused at 50ms can only be tolerated at baud rates of 115200 or higher[4]. Hence, the coexistence solution is limited only to those devices that can transfer at these high baud rates. Since IrDA communication parameters (other than baud rate) are negotiated independently, the primary station cannot control the maximum turn around time for the secondary. However, the primary is modified to limit the duration of its link to 50ms. Likewise, the primary also constrains the amount of data received from the secondary using other negotiation parameters such as data size and window size. For example, a 115.2kbps link could use a data size value of 512 bytes and a window size of 1 to limit the secondary transmission time to less than 50ms.

Under this scheme, IrDA Control transfer is allowed every 100ms (neglecting the minimum turn around time). The IR traffic profile of the host that implements the coexistence scheme is shown in Figure E.2.

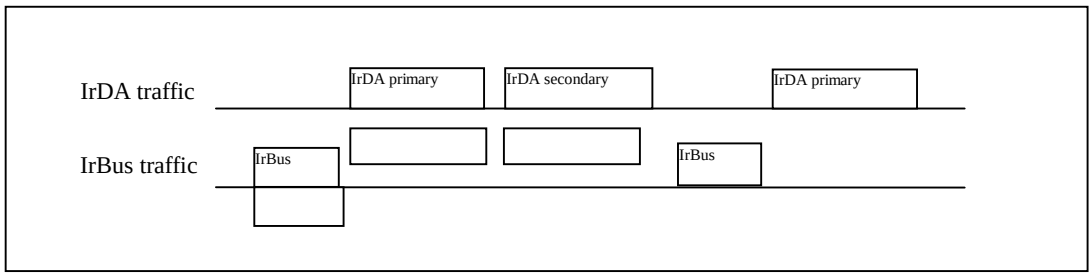


Figure E.2 IR Traffic profile in NRM

The connection parameters, once negotiated, remain valid for the duration of the connection. However, the coincident IrDA Control device may become active while an IrDA connection is in progress. Since the IrDA connection parameters cannot be renegotiated, the IrDA primary must always assume the presence of an active IrDA Control host when negotiating the connection parameters. In that case, the IrDA link performance is degraded on all IrDA stations that are also IrDA Control hosts. Alternately, the coexistence implementation may opt to break the active IrDA connection on receiving a start notification from the IrDA Control host.

[4] Both stations must agree upon baud rate since it specifies the speed at which they will transmit on the data link channel.

E.2.2 Coexistence in NDM

A coexistence scheme that allows IrDA Control traffic to coexist with IrDA contention traffic is given below. Specifically, the IrDA discovery procedures are modified to create ‘quiet time’ for IrDA Control transfers.

An IrDA device in NDM uses the discovery process to find the devices in the same space. Active device discovery is initiated only if the initiator (IrDA primary in our case) observes at least 500ms of media quiet time. If an IrDA Control device were active in the space, the IrDA Control traffic would be incorrectly interpreted as media busy by the IrDA device. However, the 500ms value for the quiet time is the upper bound for the link turn around time. Since the link is turned after 50ms under the coexistence scheme, the coincident IrDA primary proportionately reduces the media quiet time to 50ms (in other words, it ignores the default media access rule). This allows frequent IrDA Control transfers without blocking the IrDA discovery process.

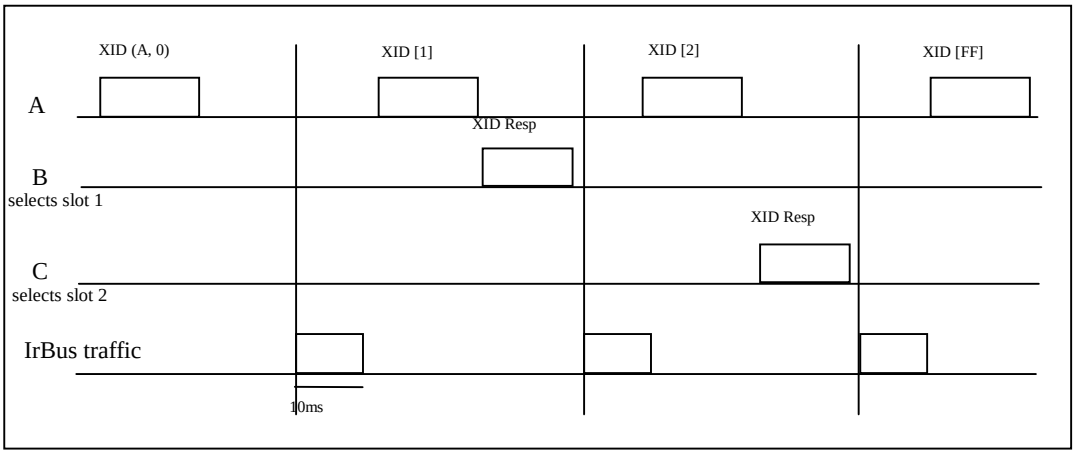


Figure E.3 IrDA-IrDA Control Coexistence in NDM

A typical IrDA implementation may use an 8-slot discovery process and takes about 1 second to complete. For the purpose of the coexistence scheme, the number of slots is reduced to 6 and the discovery frames are interleaved with the IrDA Control traffic. The discovery initiator creates ‘quiet time’ by delaying the start of the next time slot. The opening thus created is used for IrDA Control transfers. Figure E.3 shows the coexistence scheme for a hypothetical 3-slot discovery process when nodes A, B, C are all in NDM, with node A being the initiator of the discovery. The start of the time slot is delayed by 10ms each time to accommodate the IrDA Control transfer.

In addition to the above scheme, the IrDA device in NDM could also take advantage of the IrDA Control quiescent periods for opportunistic device discovery.

E.3 Impact on Performance

The IrDA coexistence scheme avoids the interference between IrDA and IrDA Control signals by distributing the signals in time. However, the interleaving of the two signals effectively degrades the performance of both IrDA and IrDA Control.

Typically, an IrDA Control host polls its bound peripherals no faster than every 13ms. However, the coexistence mode allows IrDA Control transfers only every 100ms, and the IrDA Control transfers could last for up to 10ms. Although the performance degradation may appear rather large (~80%), the IrDA Control peripherals used in the coexistence mode are limited to low speed peripherals such as keyboard and mouse. Also, Low speed IrDA Control peripherals must be polled only once every 69ms. Hence, the effective degradation is tolerable to the IrDA Control implementation.

Typically, IrDA link is turned around every 500ms to minimize the communication overhead. The IrDA coexistence scheme reduces the link time to 50ms thus increasing the communication overhead by at east a factor of 10. The resulting degradation in performance of the IrDA link is a function of the link speed. The faster links suffer larger degradation in performance. However, the overall degradation in performance is still within the market requirement of 30%.

E.4 IrDA Coexistence Implementation

The IrDA coexistence scheme assumes that the IrDA Control host and the IrDA primary are located on the same host. An implementation of the IrDA coexistence scheme requires modifications to the IrDA stack on the co-located host. Further, the existing implementation of the IrDA stacks impacts the coexistence scheme. This section provides recommendations for the implementation of the coexistence scheme that allows the scheme to work with the vast majority of the existing IrDA devices.

E.4.1 Passive Discovery

An IrDA device discovers other IrDA devices in the same space by the IrDA discovery process. Any IrDA device in the connectionless state could initiate the discovery process, provided there are no active connections in the IR medium. However, due to the media access rule that requires 500ms of media quiet before initiating a discovery, IrDA devices in the vicinity of an active IrDA Control host are unable to initiate a discovery process. Hence, the IrDA device on the coincident host (host that supports both IrDA and IrDA Control) is required to initiate periodic discovery during periods of IrDA Control inactivity. Other IrDA devices in the neighborhood utilize this process to discover devices in a passive manner.

Further, IrDA implementations for PC platforms may assume a ‘push model’, where the initiation of discovery is responsibility of the client IrDA device – for example, an IrDA camera initiates the process by which it is discovered. In order to accommodate such implementations, a monitor that periodically initiates discovery from the coincident host is implemented.

E.4.2 Role Exchange

The IrDA coexistence scheme assumes that the coincident IrDA Control host is always the IrDA primary station. However, an IrDA device with primary capabilities may initiate a connection to the coincident host. In this case, the coincident IrDA device is forced to assume the role of a secondary, thus nullifying the assumptions of the coexistence scheme. To overcome this problem, the coincident host rejects all incoming connections after noting the address of the connection initiator. The host turns around and initiates a connection with the requester immediately. This allows the coincident host to retain the IrDA primary role while at the same time honoring connection requests from other hosts.

Note that the above scheme effectively performs of a role exchange between the two IrDA stations. This could also be accomplished by using the role exchange commands in IrLAP. However, since the role exchange is an optional feature in IrLAP, the scheme described in the above paragraph is used instead.

E.4.3 IrDA Control Wakeup Frame

IrDA Control peripherals wake a sleeping IrDA Control host device by issuing a wakeup command. However, if the IrDA device on the same host is active, the wakeup frames may not reach the IrDA Control host due to the interference from the IrDA signals. To overcome this problem, the coexistence implementations must maintain the IrDA Control host in an active state as long as the IrDA connection is active. That is, the IrDA Control host is not allowed to enter or remain in a sleep state when the IrDA connection is active. Since the IrDA Control host is actively polling interleaved with the IrDA transmission, IrDA Control wakeup frames are not needed. There is no additional degradation of the IrDA link, since the IrDA connection parameters are already chosen to accommodate an IrDA Control host during link turn around time.

E.4.4 Real-Time Requirements

The IrDA Control-IrDA coexistence scheme operates at the link layer. An implementation of this scheme allows IrDA and IrDA Control devices to alternately control the IR medium for duration of 100ms and 10ms respectively. Thus, indication of start of an IrDA Control period must be communicated to the IrDA Control host in real-time to allow the IrDA Control host to utilize the entire 10ms period. In absence of real-time guarantees from the operating system, the IrLAP implementation should be implemented such that the communication with the IrDA Control stack is fast and reliable. For example, the two stacks could be implemented as a monolithic piece of software. Moreover, IrDA Control implementations based on USB must be able to communicate the state changes to the IrDA Control hardware with minimum delay. Since USB interrupt pipes do not guarantee latency, an isochronous pipe may be needed in order to implement the coexistence scheme.

E.5 Limited Interoperability

The discussion so far focused on coexistence—the ability to operate in the same space without 'noticeable' mutual interference—between IrDA and IrDA Control. However, it may be beneficial to support limited interoperability between the two protocols. Even limited interoperability between the two protocols would enable a better user model. For example, consider the case of a user who walks up to an IrDA Control host device with an IrDA-enabled laptop with the intention of communicating between the two devices. The user has mistaken the IrDA Control host for an IrDA device, since they appear very similar (similar LEDs and red plastic filters). In this case, a degree of mutual awareness between the two devices could avoid user confusion. For instance, if the IrDA Control device could communicate with the IrDA device, the IrDA device, in turn, could inform the user of the mismatch.

Full interoperability between IrDA and IrDA Control would require that both devices be able to receive each other's signal as well as transmit each other's signal. It is not efficient to implement a physical layer that is capable of receiving signals from both IrDA and IrDA Control. However, an IrDA device (IrDA Control device) that could transmit IrDA Control signals (IrDA signal) is possible. Since the goal of IrDA Control-IrDA interoperability is to reduce user confusion, the limited interop is acceptable.

The interoperability matrix for IrDA Control and IrDA is shown in Table E.1. There are 4 distinct interops of interest. In case of interop-1 and interop-3, an IrDA host is attempting to communicate with an IrDA Control host. Here the IrDA Control physical layer transmits an IrDA signal that corresponds to an invalid IrLAP frame. For example, a discovery IrLAP frame is sent to the IrDA broadcast address with a source device address of 0. The IrLAP layer infers an IrDA Control device on reception of such a packet. Since the IrDA Control device is unable to receive the IrDA signal, it could periodically repeat the invalid discovery packet to announce the presence of the IrDA Control device to the neighboring IrDA devices.

Interop table	***IrBus host***	***IrBus peripheral***	***IrDA primary***	***IrDA secondary***
IrBus host		Normal	Interop-1	Interop-3
IrBus peripheral	Normal		Interop-2	Interop-4
IrDA primary	Interop-1	Interop-2		Normal
IrDA secondary	Interop-3	Interop-4	Normal	

Table E.1 IrDA Control-IrDA Interoperability

Similarly, in case of interop-2 and interop-4, an IrDA Control peripheral is attempting to communicate with an IrDA host. The IrDA station sends an IrDA Control signal that corresponds to a device hail on peripheral address 0xE. Since only hails on peripheral address 0x0 and 0xF are valid in IrDA Control, the IrDA Control peripheral infers that it is in the vicinity of an IrDA device. The peripheral, in turn, could communicate the error to the user through visual indication, such as an LED. While it is possible to handle this interop by allowing the IrDA Control peripherals to send IrDA signals, it may not be cost effective.

The interop schemes discussed above must be used judiciously to avoid interference with the coexistence scheme.

Appendix F. IrDA Control on a USB System

F.1 Introduction

Familiarity with USB and HID systems is assumed. The goal for IrDA Control on a USB system is to behave similarly to a “Wired” USB system. The Descriptors that an IrDA Control peripheral sends to a host are similar to the Descriptors that a wired version of the same device would send. As an example, IrDA Control peripherals send Descriptors that can be parsed by standard USB host software and the standard HID parser that is used for wired devices. An IrDA Control Transceiver Module (IRB-TM) is a USB device that can be implemented in many different ways, ranging from a total hardware solution to a mostly software solution. At different times (BIOS or Operating System in control) the IRB-TM may look like a simple device, a complex device, or a hub. The IRB-TM is responsible for enumeration and binding of peripherals at the Physical level, as well as taking part in the normal USB enumeration. Peripherals are more limited in how they can be implemented, and must interoperate with a variety of host side implementations.

F.2 Hardware based IRB-TM Implementation

In a Hardware-Firmware IRB-TM implementation, most (or all) of the compatibility with USB is handled by a fairly complex IrDA Control Transceiver Module (IRB-TM) implementation. Characteristics of a USB wired-device are emulated by the IRB-TM to make the connection and disconnection of IrDA Control peripherals appear the same as wired devices. To the host, the IRB-TM appears as a USB hub. One characteristic of this method is that every IrDA Control device requires its own unique USB device address, which implies that a full implementation would require 9 addresses; 8 for the IrDA Control devices, plus 1 for the USB hub function (note that this could be an integrated solution). The main advantage of a hardware solution is that the standard USB drivers that ship with an operating system work as is (with the exception that the USB and IrDA stacks must be modified for coexistence). The disadvantages include higher cost, and added complexity in the firmware; the IRB-TM must store information about devices, in order to handle all transactions within the timing constraints that USB requires.

F.3 Host Software based IRB-TM Implementation

In a Host Software implementation, the IRB-TM has the equivalent USB interface hardware of one high-speed device. It only requires one USB device address. Single USB data IN and OUT endpoints are shared by all IrDA Control devices. The IRB-TM puts identification wrappers on each device's data. The host software unpacks them. To the upper layers of the USB and HID drivers, the IrDA Control devices enumerate, connect, and disconnect much like wired devices.

F.4 IrDA Control USB-HID Compatible Protocol

IrDA Control peripherals for the PC host include keyboard, mouse, joystick and other gaming devices. These devices are described via USB style Descriptors including a HID Report Descriptor to leverage the Plug-N-Play features of the existing HID class driver framework in existing operating systems.

The USB-HID compatible protocol is a command and response protocol.

- Multiple interfaces are not supported.
- Only Short (8 byte or less data payload) Packets should be used. (required)
- No more than four endpoints per IrDA Control device. (required)

F.5 USB-HID Enumeration

After an IrDA Control device binds and enumerates with a USB-HID IrDA Control Transceiver Module, the module must get a copy of the device's USB-HID descriptor set. This is accomplished by sending a series of Get_Descriptor commands to the device.

Source	HADD	MACC	PADD	LLC	DATA
Host	Host Addr	0xC	0xN	0x09	Descriptor ID

If the device supports a specific descriptor, it responds with an ACK packet. If a device requires more time to service a request it must respond with a NAK packet. A device must respond with a STALL packet for any unsupported request.

The descriptor requests that must be supported by USB-HID compatible IrDA Control devices are:

- Device descriptor (ID = 0x01)
- Configuration descriptor (ID = 0x02)
- HID Report descriptor (ID = 0x22)
- IrDA Control descriptor (ID = 0x80)
- Copyright string (ID = 0xA9)

Descriptor data is sent to the host in eight-byte (or less) data packets in response to IN packets from the host. The first packet must have a toggle value of one. The host must ACK each data packet in order to receive the next packet. The final packet from the device must be less than eight bytes long (may be zero bytes long). This serves to notify the host that this is the last packet of the descriptor data.

F.6 Polling During Upper Layer Enumeration

If it is desired to keep a device bound before USB data pipes, or USB polling become active, it can be accomplished with periodic polling using the following packet:

Source	HADD	MACC	PADD	LLC	DATA
Host	Host Addr	0xC	0xN	None	None

F.7 After Upper Layer Enumeration

The IN polling process for a data pipe is performed as follows:

1. Bound peripherals will be polled periodically by the host with the following packet. This packet requests data from endpoint 1.

Source	HADD	MACC	PADD	LLC	DATA
Host	Host Addr	0xC	0xN	0x28	None

2. A peripheral which, has data ready to send to the host, responds with a DATA1 or DATA0 packet. The data packets should toggle between DATA0 and DATA1 packets. . This packet sends data with a toggle value of zero (DATA0) from the peripheral’s endpoint 1, to the host.

Source	HADD	MACC	PADD	LLC	DATA
Peripheral	Host Addr	0x4	0xN	0x23	Report Data

If the peripheral has no data to send, it may respond with a NAK packet, or choose not to respond at all.

Source	HADD	MACC	PADD	LLC	DATA
Peripheral	Host Addr	0x4	0xN	0x26	None

If the host receives either a DATA0 or DATA1 packet, it should respond with an ACK packet. The ACK packet is a handshake to validate that it's data has been received. If the peripheral does not receive an ACK packet, it will assume that the previous packet was not received and send the same type of data packet at the next request.

Source	HADD	MACC	PADD	LLC	DATA
Host	Host Addr	0xC	0xN	0x24	None

The process for an OUT data pipe on Endpoint 2 is performed as follows [optional]:

1. Bound peripherals will be sent data periodically by the host with the following packet. This packet sends data to endpoint 2, the peripherals output channel from the host. The data packets should toggle between DATA0 and DATA1 packets. A DATA1 packet is shown below. A DATA0 packet would have an LLC of 0x47.

Source	HADD	MACC	PADD	LLC	DATA
Host	Host Addr	0xC	0xN	0x4F	Data

2. If the peripheral receives either a DATA0 or DATA1 packet, it should respond with an ACK packet. The ACK packet is a handshake to validate that it's data has been received. If the host does not receive an ACK packet, it will assume that the previous packet was not received and send the same type of data packet at the next request.

Source	HADD	MACC	PADD	LLC	DATA
Peripheral	Host Addr	0x4	0xN	0x24	None

F.8 Additional USB-HID IrDA Control Commands

Two additional IrDA Control commands are provided to allow setting modes or options within a device, and getting internal status information.

This example of a Set_Mode command is used to turn on and off a keyboard's LED indicators, such as the 'Caps Lock', 'Scroll Lock', and 'Num Lock' LEDs:

Source	HADD	MACC	PADD	LLC	DATA
Host	Host Addr	0xC	0xN	0x0A	0x01 (keyboard LEDs),(LED data)

This example of a Get_Status command would return a keyboard's current LED status:

Source	HADD	MACC	PADD	LLC	DATA
Host	Host Addr	0xC	0xN	0x0C	0x01 (keyboard LEDs)

F.9 Example Descriptors for an IrDA Control Mouse

```
;////////////////////////////////////////////////////////////////////////
;// Sample 3D Mouse Descriptors.
;// This is a sample only. Not all object bytes are
;// valid. (Vendor ID, et al., are set to 0), etc.
;////////////////////////////////////////////////////////////////////////

EXTRN bcdVersion

;============================================================================
; Descriptor ID Codes
;============================================================================
idDEVICE        EQU     01h
idCONFIG        EQU     02h
idSTRING        EQU     03h
idREPORT        EQU     22h
idIRBUS         EQU     80h
idCOPYRIGHT     EQU     0A9h

;////////////////////////////////////////////////////////////////////////
;//   Descriptor Tables
;//   3-axis, 3-button Mouse
;////////////////////////////////////////////////////////////////////////

;============================================================================
; Copyright String
;============================================================================
psCOPYRIGHT:
   DB    '© 1997 BigTime IrBus Corporation. All rights reserved.'
cCOPYRIGHT      EQU     ($-psCOPYRIGHT)

;============================================================================
; Mouse (3D) Descriptors
;============================================================================
pdDEVICE:                       ; Device Descriptor
   DB    cDEVICE                ; bLength
   DB    idDEVICE               ; bDescriptorType
   DW    0100h                  ; bcdUSB
   DB    00h                    ; bDeviceClass
   DB    00h                    ; bDeviceSubClass
   DB    00h                    ; bDeviceProtocol
   DB    08h                    ; bMaxPacketSize0
   DW    0000h                  ; idVendor (BigTime IrBus Corporation)
   DW    0000h                  ; idProduct Assigned by vendor
   DW    bcdVersion             ; bcdDevice
   DB    00h                    ; iManufacturer
   DB    00h                    ; iProduct
   DB    00h                    ; iSerialNumber
   DB    01h                    ; bNumConfigurations
CDEVICE         EQU     ($-pdDEVICE)
;============================================================================
pdCONFIGURATION:                ; Configuration Descriptor
   DB    cCONFIGURATION         ; bLength
   DB    idCONFIG               ; bDescriptorType
   DW    cCONFIG_REQ            ; wTotalLength
   DB    01h                    ; bNumInterface
   DB    01h                    ; bConfigurationValue
   DB    00h                    ; iConfiguration    none
   DB    01100000b              ; bmAttributes (Self Powered, Remote Wakeup)
   DB    01h                    ; MaxPower 2mA. Just in case some driver chokes on zero power.
CCONFIGURATION          EQU     ($-pdCONFIGURATION)
;============================================================================
pdREPORT:                       ; HID Report Descriptor
   DB    05h,01h                ; Usage Page (Generic Desktop),
   DB    09h,02h                ; Usage (Mouse),
   DB    0A1h,01h               ; Collection (Application),
   DB    09h,01h                ;   Usage (Pointer),
   DB    0A1h,00h               ;   Collection (Physical),
   DB    05h,09h                ;     Usage Page (Buttons),
```

```
    DB      19h,01h             ;    Usage Minimum (01),
    DB      29h,03h             ;    Usage Maximum (03),
    DB      15h,00h             ;    Logical Min (0),
    DB      25h,01h             ;    Logical Max (1),
    DB      75h,01h             ;    Report Size (1),
    DB      95h,03h             ;    Report Count (3),
    DB      81h,02h             ;    Input (Data, Variable, Absolute),
    DB      75h,05h             ;    Report Size (5),
    DB      95h,01h             ;    Report Count (1),
    DB      81h,01h             ;    Input (Constant),
    DB      05h,01h             ;    Usage Page (Generic Desktop),
    DB      09h,30h             ;    Usage (X),
    DB      09h,31h             ;    Usage (Y),
    DB      09h,38h             ;    Usage (Wheel),
    DB      15h,81h             ;    Logical Min (-127),
    DB      25h,7Fh             ;    Logical Max (127),
    DB      75h,08h             ;    Report Size (8),
    DB      95h,03h             ;    Report Count (3),
    DB      81h,06h             ;    Input (Data, Variable, Relative),
    DB      0C0h                ;   End Collection,
    DB      0C0h                ; End Collection
CREPORT             EQU     ($-pdREPORT)
;===========================================================================
cCONFIG_REQ EQU             ($-pdCONFIGURATION)
;===========================================================================
pdIRBUS:            ; IrBus Descriptor
    DB      cIRBUS              ; Descriptor byte count
    DB      idIRBUS             ; Descriptor ID
    DW       0100h              ; bcdIrBusVersion
    DB      22h                 ; idClass_Descriptor_1 (HID Report)
    DB      00h                 ; idClass_Descriptor_2
    DB      00h                 ; idClass_Descriptor_3
    DB      00h                 ; idClass_Descriptor_4
    DB      01000100b           ; bmEndpoint_1 (Interrupt, 4-bytes)
    DB      00000000b           ; bmEndpoint_2 (Off)
    DB      00000000b           ; bmEndpoint_3 (Off)
    DB      04h                 ; bLogDevPktSize_1 (Boot mouse, no break packet)
    DB      10000000b           ; bmLogDevAttributes_1
    DB      00h                 ; bLogDevPktSize_2 (Off)
    DB      00000000b           ; bmLogDevAttributes_2
    DB      00h                 ; bLogDevPktSize_3 (Off)
    DB      00000000b           ; bmLogDevAttributes_3
CIRBUS              EQU     ($-pdIRBUS)
;===========================================================================
```

Appendix G. Multiple Hosts

G.1 Introduction

A likely user environment for IrDA Control is multiple host systems interacting within the same user space. This could occur in many different situations, including:

- A PC-TV and a “traditional” PC in the same room
- Multiple Home Automation (HA) hosts in the same room
- An office with two PCs or similar devices
- Showroom, trade show, etc.

In any of these situations, the transmissions from any one host could potentially interfere with the transmission of one or more hosts. This appendix describes a media access scheme that allows the multiple IrDA Control hosts in a user space to share the IR medium. While media access technologies such as FDMA, CDMA, and TDMA are well known, they are not appropriate for IrDA Control; wavelength and subcarrier division are not practical with infrared technologies available in the near term, for example. For this reason, the IrDA Control multiple hosts solution implements a Packet Reservation Multiple Access (PRMA) scheme that is based on a statistical slotting algorithm.

G.2 Operational Scenario

There are a number of situations that the multi-host aspects of IrDA Control must be able to accommodate:

- Occasions where a host can detect the presence of other hosts in the IR medium,
- Occasions where hosts cannot detect each other, but can only infer the presence of multiple hosts because of interference at one or more peripherals.

An example is shown in Figure G.1.
The following is a list of possible connections in the IR medium:

1. Hosts cannot communicate with each other, and hosts do not interfere at peripherals (disjoint hosts - can be operated independently)
2. All the hosts in a given user space could communicate with each other either directly or indirectly (via peripherals)
3. None of the hosts in a given user space can communicate with each other, but there is interference between at least two hosts at one or more peripherals (hidden host model)

While it is possible to adopt different strategies for each of the above situations, a single multiple hosts scheme best serves the implementation of IrDA Control. Further, a multiple host scheme that addresses the hidden host model is equally effective for the other multiple host scenarios.

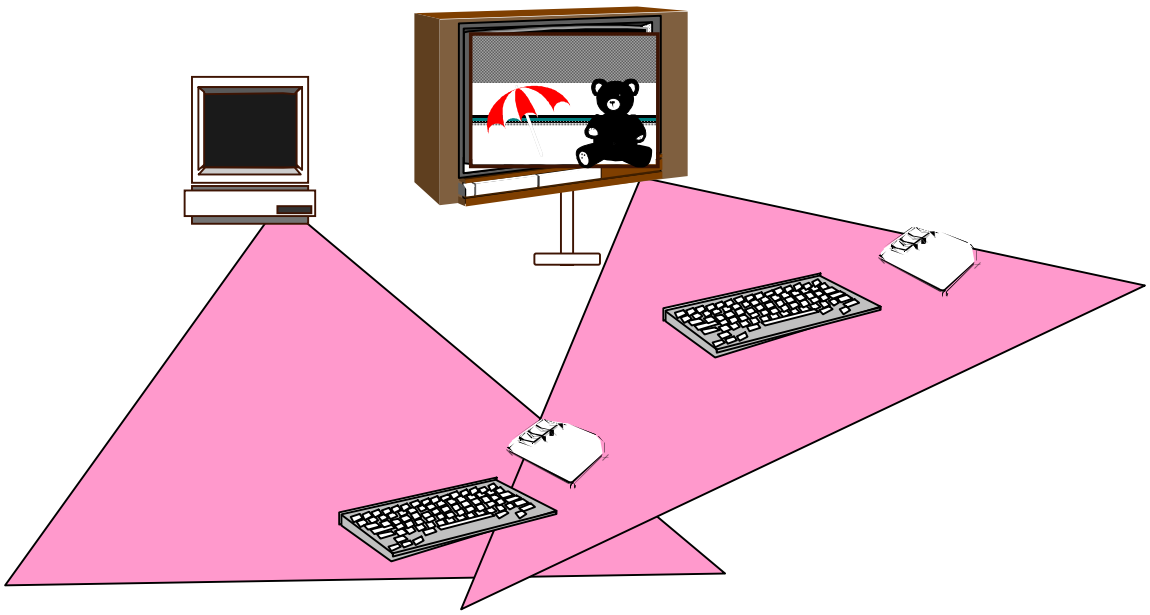


Figure G.1: PC-TV and PC with overlapping IrDA Control coverage areas

G.3 Dithering Scheme

The multiple host scheme for IrDA Control is based on a dithering algorithm that models a random walk in one dimension. In this scheme, a host randomly dithers the start of the host polling cycle in an attempt to create a clear slotting of frames within a period. This is a form of Packet Reservation Multiple Access (PRMA), but without confirmation of the reservation from other hosts. This scheme forces a host to dynamically adjust its frame timing in an attempt to fit within the parameters of a 13.8ms polling period.

An example of the multiple host interference for three hosts is shown in Figure G.2. This figure shows the interference caused by overlapping of the host frames (captioned *Host frames before dithering*), and the non-interfering slotting for the three hosts found by the dithering scheme (captioned *Host frames after dithering*).

The dithering algorithm is based on minimizing the errors seen at each host. The interference caused by hidden hosts may result in CRC errors or framing errors in either the poll or response packets of a host. These errors manifest themselves at the host as CRC errors or framing errors in the response packet, or, in the extreme case, as a lack of response to a host poll[5]. A host maintains a count of the errors seen at the host over a number of host frames. When a host detects a sufficient number of errors have occurred over several frames, the host infers that there is interference in the space. It delays the start of its next cycle by a random time chosen uniformly from the interval of 0 to 12 milliseconds, and resumes sending IrDA Control frames. Once the number of errors drops below a specified threshold, the host stops dithering if a steady state has been reached. Consequently, there is a period of time where the host will dither the start of the host polling cycle in order to search for a clear time gap in the IR space. If the sum of all the frame times from all the hosts overlapping in the space exceeds 13.8ms, this dithering process will continue indefinitely, since no optimal timing solution is possible. In this scheme, the overall IrDA Control system dithers randomly to search for a solution within the 13.8ms period constraint.

Every host in the multi-host environment whose error rate exceeds the specified threshold, the “high error threshold”, must dither its frame until the error rate observed by that host drops below another threshold, the “low error threshold”. The hysteresis of the two levels of error thresholds prevents false triggers and contributes to the stability of the algorithm.

[5] Note that a peripheral may not always respond to a device hail –viz, an enumeration hail or a binding hail. Hence, lack of response to a device hail is not treated as an error.

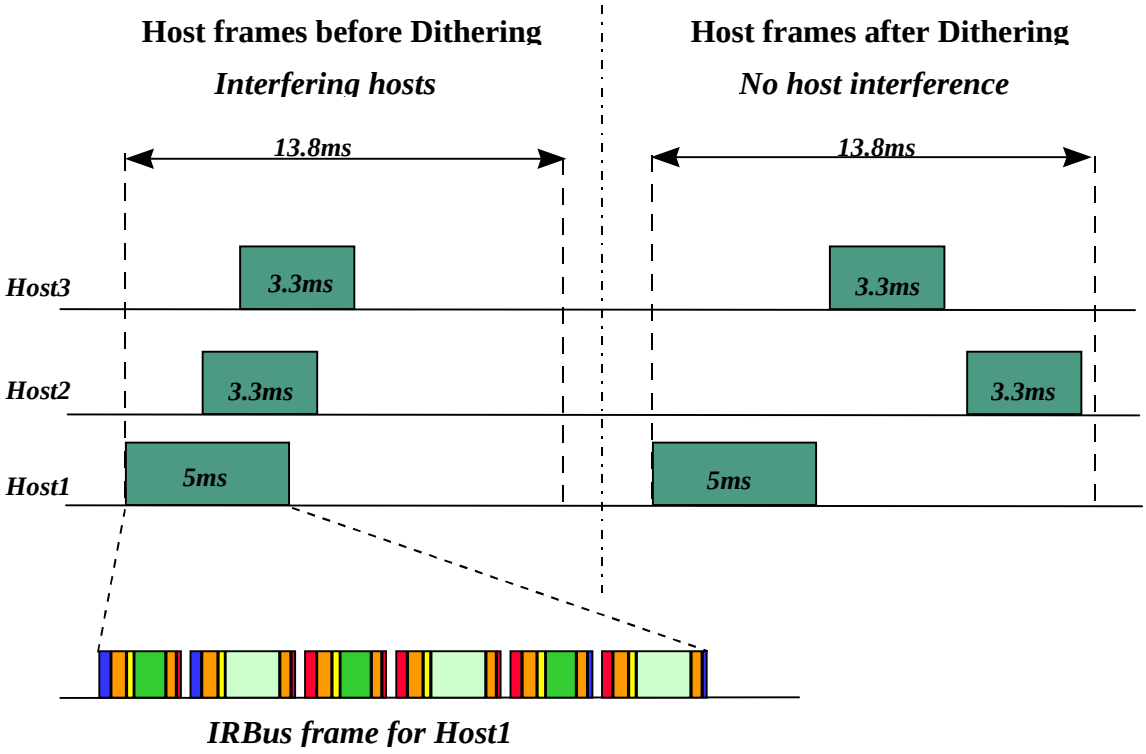


Figure G.2: Multiple host interference with three IrDA Control hosts

The multiple host scheme applies only when a host is the Normal Mode (or Mode1). For a host in IrDA Coexistence Mode (or Mode 2), the dithering of the host frame caused by the multiple hosts scheme will interfere with IrDA communications, voiding the benefits of the coexistence mode. Besides, a non-interfering slotting of the hosts can be found only if the combined frame times of all the hosts in the user space does not exceed 13.8ms, an assumption that is clearly not valid in the IrDA coexistence mode.

G.4 Implementation Notes

The effectiveness of the dithering scheme is determined by the convergence time for a given combined frame time (the combined frame time for the three hosts in figure G.2, for example, is 5+3.3+3.3 = 11.6ms).

The following parameters affect the convergence time of the dithering scheme: the cycle count, the high error threshold, and the low error threshold. The effective values for these parameters are determined through simulations. In addition, the following empirical rule is used to control the dithering for a host – "For a dithering scheme with a frame count of n, if no errors are detected on the n^{th} frame, dithering is stopped".

Cycle Count – is the number of 13.8ms cycles over which the errors are measured by a host. A host could potentially measure the error rate after every host polling cycle. However, errors are measured over a number of cycles to overcome the effects of transient errors.

High error threshold – When the percentage of errors observed at a host exceeds this value, the host starts to dither its frame. This parameter is chosen so that dithering is not triggered due to transient errors.

Low error threshold – A host stops dithering as soon as the percentage of errors fall below the low error threshold. In absence of this parameter, the dithering scheme may become unstable due to errors varying about the high error threshold value. The absolute value of the low error threshold is less than that of the high error threshold and is chosen to optimize the 'stable' convergence time.

G.5 Host Algorithm

The host algorithm for implementing the multiple hosts scheme is given below:

```
#define TRUE        1
#define FALSE       0

#define DITHER_CYCLE_COUNT      5       // 5 polling cycles of 13.8ms each
#define DITHER_LOW_THRESHOLD    5       // 5%
#define DITHER_HIGH_THRESHOLD   25      // 25 %

int     dodither;                   // boolean
int     cyclecount;
int     responsecount;              // number of responses to polls (including errors)
                                    // responses to device hails are excluded

int     errorcount;                 // number of responses in error
int     lastcycleerrorcount;        // errors encountered on the 5th cycle
```

On successful reception of a frame:

```
        responsecount++;
```

When a receive error (CRC error, framing error) occurs,
or no peripheral response to a host poll (except for host hails):

```
        responsecount++;
        errorcount++;
        lastcycleerrorcount++;
```

When a 13.8ms cycle expires:

```
void doMultiHostProcessing()
{
        cyclecount++;
        if (cyclecount >= DITHER_CYCLE_COUNT) {
                if (dodither) {
                        if (responsecount * DITHER_LOW_THRESHOLD >
                            errorcount * 100) {
                                dodither = FALSE;
                        }
                } else  {
                        if (responsecount * DITHER_HIGH_THRESHOLD <
                            errorcount * 100) {
                                dodither = TRUE;
                        }
                }
                cyclecount = 0;
                responsecount = 0;
                errorcount = 0;

                if (lastcycleerrorcount == 0) {
                        dodither = FALSE;
                }

                if (dodither) {
                        /* Wait a random time (0-12ms) */
                }
        }
        lastcycleerrorcount = 0;
}
```