USB2 Debug Device
A Functional Device Specification

Date: March 25, 2003

Revision: 0.9

The information is this document is under review and is subject to change.

Scope of this Revision

The 0.9 revision of the specification is intended for review purposes only.

Revision History

Revision	Issue Date	Comments
0.5	4/04/2002	Initial Revision
0.9	3/25/2003	for public review

Significant Contributors:

John Keys (Author)	Intel Corporation
Brad Hosler	Intel Corporation
John S. Howard	Intel Corporation
Rahman Ismail	Intel Corporation

Please send comments via electronic mail to: USBDebug@intel.com

Table of Contents

1. INTRODUCTION 1

2. OVERVIEW 1

3. DEVICE REQUIREMENTS 2

4. DEVICE FRAMEWORK 2

4.1 Debug Descriptor 2
4.1.1 Get Descriptor – Debug Descriptor 3

4.2 Debug Mode Feature Selector 4
4.2.1 Set Feature – Debug Mode 4

1. Introduction

This document provides the definition and requirements for a device that operates as a debugging device for use with the Enhanced Host Controller Interface Debug Port. This type of device is intended to replace the serial port and null-modem cables currently used, providing a significant increase in debugger throughput and a migration path for legacy-free platforms. This document is intended to be useful for two purposes:

- A hardware device vendor or firmware engineer intending to build and program debug devices which adhere to this specification, and
- A software driver developer programming to use the debug device.

This specification is organized as follows:

- Section 2 Overview
- Section 3 Device Requirements
- Section 4: Device

2. Overview

The Enhanced Host Controller Interface Specification for Universal Serial Bus includes the definition of an optional debug port. This port is intended to replace legacy COM ports for debugging use as platforms move towards legacy-free configurations. A second benefit of USB debugger connection is the significant increase in bandwidth that USB provides over industry-standard serial ports. To be useful, the debug port requires a second component, the Debug Device.

A *Debug Device* provides a USB communication channel between a target machine and a remote machine. It is two logical USB devices connected with a private communications channel. Each logical device consists of an OUT Bulk endpoint and an IN Bulk endpoint. These endpoints each represent a Stream Pipe between the device and its respective host. Data received from Host X on the OUT pipe of one device is forwarded to the IN pipe of the other device for transmission to Host Y.

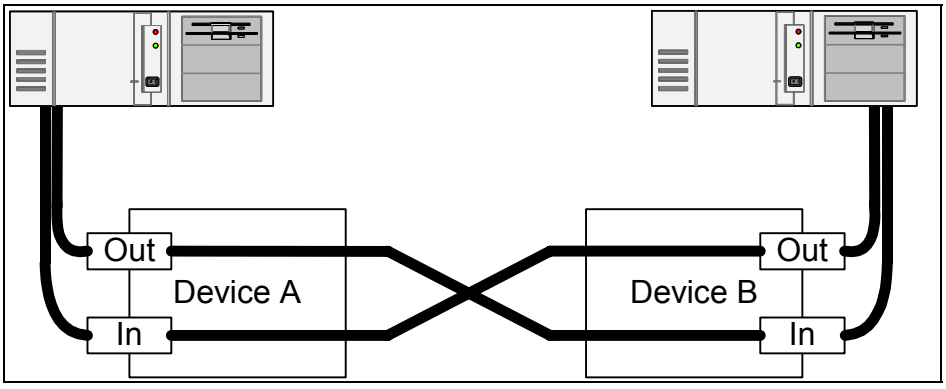

Figure 1: Logical USB Debug Device

The *Debug Pipe* is the combination of four individual Stream Pipes. It consists of two logical Stream Pipes, representing a bi-directional pipe between two hosts.

Some general restrictions and operational differences result from the use of the debug port rather than the standard host controller interface. The main operational difference is packet/transfer size. The debug port has a maximum packet size of eight bytes. Since this is smaller than the USB2 High Speed control endpoint packet size of sixty-four bytes, this also means that control transfers have a maximum transfer size of eight bytes as well. Because of this limitation, the debug port driver cannot use standard means to enumerate and configure debug devices.

This maximum packet size restriction also means that the end of the Debug Pipe that is connected to the Debug Port has to operate with a maximum packet size of eight (8) bytes. This limitation does not apply to end of the Debug Pipe connected to the remote machine. However, if a packet larger than eight bytes is received from the remote computer, the device must break the larger packet into eight-byte packets before sending the data to the Debug Port.

Dedicated Debug Devices (Debug Device functionality only) may support a single fixed device address of 127. These devices must have clearly differentiated 'ends', with one 'end' or device always connected to the Debug Port and the other end always connected to the remote computer. Only the Debug Port end of the device is allowed to contain a fixed address. The Remote end must follow standard USB semantics.

A device end that contains a fixed address never exists in *default* state. It always exits USB reset in the *addressed* state. Because it never responds to address 0, this type of device cannot be enumerated by the standard host driver stack. It can only be enumerated correctly by dedicated debug-port drivers.

3. Device Requirements

This section provides an overview of the requirements of a Debug Device. These feature requirements may be implemented utilizing any method (firmware, dedicated hardware, or combination).

- Implement the descriptors, features, and requests described in Section 4: Device Framework.
- USB 2.0 High Speed signaling compliant.
- One Bulk-type IN endpoint that supports 8-byte maximum packet size.
- One Bulk-type OUT endpoint that supports 8-byte maximum packet size.
- If a dedicated Debug Device has a fixed address, the address must be 127.
- The Control Endpoint must correctly handle short transfer requests of 8 bytes.
- NYET responses from the OUT endpoints must be equivalent to an ACK response.

4. Device Framework

All Debug Devices, with the exception of fixed address devices, must implement all required standard commands in the core device framework. In addition, all debug devices must support the following set of framework extensions.

4.1 Debug Descriptor

This descriptor is used to describe certain characteristics of the device that the host debug port driver needs to know to communicate with the device. Specifically, the debug descriptor lists the addresses of the endpoints that comprise the Debug Pipe. The endpoints are identified by endpoint number. This number is identical to bits 3-0 of the *bEndpointAddress* field in an Endpoint descriptor.

Table 4–1. Debug Descriptor Type

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Length of this descriptor in bytes: 4
1	**bDescriptorType**	1	Constant	DEBUG Descriptor type
2	**bDebugInEndpoint**	1	Number	Endpoint number of the Debug Data IN endpoint. This is a Bulk-type endpoint with a maximum packet size of 8 bytes.
3	**bDebugOutEndpoint**	1	Number	Endpoint number of the Debug Data OUT endpoint. This is a Bulk-type endpoint with a maximum packet size of 8 bytes.

Table 4–2. Descriptor Types

Descriptor Type	Value
DEVICE	1
CONFIGURATION	2
STRING	3
INTERFACE	4
ENDPOINT	5
DEVICE_QUALIFIER	6
OTHER_SPEED_CONFIGURATION	7
INTERFACE_POWER	8
OTG	9
DEBUG	10

4.1.1 Get Descriptor – Debug Descriptor

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_DESCRIPTOR	DEBUG type	0	4	Debug Descriptor

Host software uses the GET_DESCRIPTOR request to get a device's debug descriptor. This command is sent with *device* as the recipient and a wValue equal to DEBUG descriptor type. If a device supports Debug Device operation, it should return a valid Debug descriptor to the host. Devices that do not support Debug Device operation should give a Request Error response.

Debug Port host drivers use this request to determine if a connected device supports Debug Device operation.

Default state: This is a valid request when the device is in the Default state.

Address state: This is a valid request when the device is in the Address state.

Configured state: This is a valid request when the device is in the Configured state.

4.2 Debug Mode Feature Selector

Because of the 8-byte packet limit, debug port drivers cannot retrieve any more than the first eight bytes of a configuration descriptor (and the subsequent interface and endpoint descriptors that follow it). This prevents the driver from being able to parse configuration information in order to determine how to configure a debug device. The DEBUG_MODE feature selector provides a means for a debug port driver to enable a debug device connected to the debug port. When the feature is enabled, the device prepares for Debug Device operation and enables its Debug Pipe.

Table 4–3. Standard Feature Selectors

Feature Selector	Recipient	Value
DEVICE_REMOTE_WAKEUP	Device	1
ENDPOINT_HALT	Endpoint	0
TEST_MODE	Device	2
OTG b_hnp_enable	Device	3
OTG a_hnp_support	Device	4
OTG a_alt_hnp_support	Device	5
DEBUG_MODE	Device	6

4.2.1 Set Feature – Debug Mode

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_FEATURE	DEBUG MODE	0	0	none

Host software uses the SET_FEATURE request to enable Debug Device operation. This command is sent with *device* as the recipient and a wValue equal to DEBUG_MODE feature selector. If a device supports Debug Device operation, it should enable that operation before providing a Success Response to the host. This command is equivalent to issuing a Set Configuration request, specifying the configuration that contains the Debug Device data endpoints. Devices that do not support Debug Device operation should give a Request Error response.

Default state: This is a valid request when the device is in the Default state.

Address state: This is a valid request when the device is in the Address state.

Configured state: The device responds with a Request Error.