Microsoft Corporation

HID Over SPI Protocol Specification

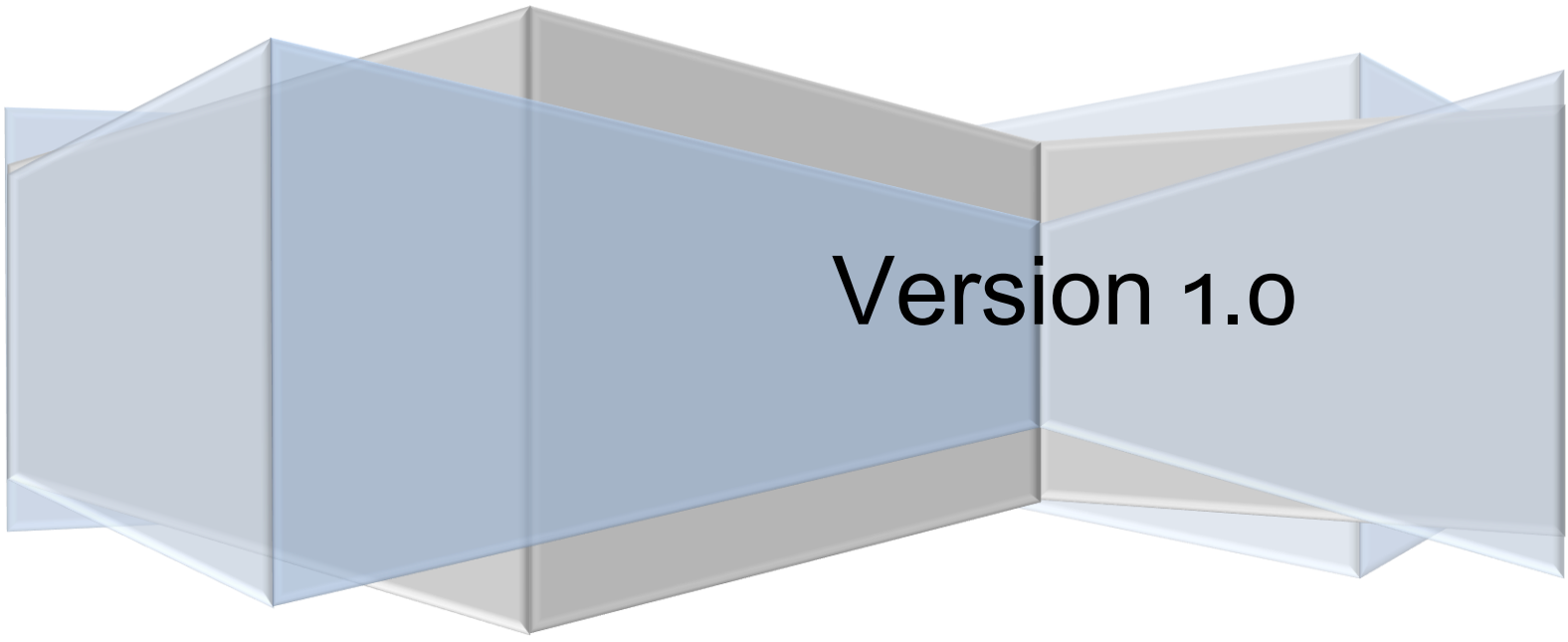


Table of Contents

1 PREFACE

The Human Interface Device protocol was first defined for USB attached input devices (e.g. keyboards, mice, remote controls, buttons, etc.). The protocol itself is bus agnostic and has been ported across other transports, including Bluetooth™ and other wired and wireless technologies.

This specification document will build on the USB defined specification and identify the protocol, procedures and features for simple input devices to talk HID over SPI. This specific document will provide details on the DEVICE side of the protocol. Details on HOST side optimizations are captured in a separate specification.

1.1 VERSION HISTORY

Date	Changes
Sep 5, 2019	First Draft and overall layout identified
Mar 2, 2020	Updated to version 0.97 Added 4 byte alignment padding to output reports
May 27, 2020	Updated to version 0.98 - Clarify applicability of certain sections to ACPI / Generic SPI - Further define connection/bus characteristics - Add Platform Specific Controllers - Misc. clarifications
June 2, 2020	Updated to version 0.99 - Update placeholder bytes to dummy clock cycles - Clarify 64K report size support is optional for HOST & DEVICE if both can support the maximums of each other - Add fragment pending bit - Update ASL example
June 30, 2020	Updated to version 0.100 - Remove fragment pending bit - Remove interrupt between fragments - Move report type field from input report header to body
August 21, 2020	Updated to version 0.101 - Fix reference to HIDI2C in example - Remove reference to RR HID usages in report descriptor example - Restore requirement for interrupt-per-fragment - Add interrupt timing restrictions - Fix diagrams - Update protocol version references to 0x3 (from 0x2) - Add flags field in descriptor, make optional output report acknowledgement public - Content lengths are 0-based (rather than 3)
October 16, 2020	Updated to version 0.102 - Relax interrupt assertion timing requirements - Consistency of terms (power ON, register -> address) - Add optional single-SPI write mode - Specify interrupt / reset line drive & pull-up behavior
October 30, 2020	Updated to version 0.103

	- Clarify interrupt line requirements - Decouple DEVICE power states from D-states for platform specific controller scenarios
December 11, 2020	Updated to version 0.104 - Remove DEVICE response on set power sleep command - Add set power command for power OFF state - Update section 8.2 (HIPO) to clarify power state transition behavior
February 2, 2021	Update to version 1.0 - Clarify that HOST must rest DEVICE upon receipt of an invalid packet (9.1)
March 26, 2021	- Correct flag bits in ACPI definition (bit 13 was referenced twice) - Clarify wMaxFragmentLength field of device descriptor, correct wMaxFragmentLength field in example in 11.2 - Clarify wording on timeouts in 9.2 - Explicitly state in 7.1.6 that DEVICE must provide fragments within 1 second of a prior fragment (clarification – was already a requirement as per 9.2) - Misc. layout improvements - Clarify how input/output reports correspond to HID specification reports in 7.

1.2 DEFINITIONS

HID – [Human Interface Device] This term is commonly used to refer to either the protocol or the device itself. In this document, we will use the word “HID” when referring to the device and “HID Protocol” when referring to protocol in definition.

SPI - SPI (Serial Peripheral Interface) is a synchronous serial communication bus that is used to attach peripherals to a motherboard.

PID – [Physical Interface Device] A special class of HIDs that send physical (tactile) output as well as input along with definition of how the HIDs interact with human hands. This class has not been very popular and has not been supported in many modern operating systems.

SPB – [Simple Peripheral Bus] For the purpose of this document the specific SPB are I^2C , SPI, etc. This specification is focused on supporting HID over SPI.

USB – [Universal Serial Bus] Universal Serial Bus (USB) is an industry standard which defines the cables, connectors and protocols used for connection, communication and power supply between USB compliant Hosts and Devices.

HOST – The term HOST refers to the SPI controller or the software on the operating system that governs the operation of the controller and the HID over SPI protocol.

DEVICE – The term DEVICE refers to the SPI peripheral that is connected to the SPI controller and functions in compliance with the HID over SPI protocol specification.

Platform Specific Controller – A HOST-side controller implementation which does not utilize generic ACPI SPB resources, or ACPI enumeration.

Class Driver – A software driver that is provided in an operating system that is capable of working with different hardware devices that are built in compliance to a class specification.

Fragment – A HID input report can be transmitted in multiple pieces. Each piece is called a fragment.

1.3 DOCUMENT CONVENTIONS

This specification uses the following typographic conventions

Example of Convention	Description
Get_Report, Report	Words in bold with initial letter capitalized indicate elements with special meaning such as requests, descriptors, descriptor sets, classes, or subclasses.
Data, Non-Data	Proper-cased words are used to distinguish types or categories of things. For example Data and Non-Data type Main items.
bValue, bcdName, wOther	Placeholder prefixes such as 'b', 'bcd', and 'w' are used to denote placeholder type. For example: • *b* bits or bytes; dependent on context • *bcd* binary-coded decimal • *d* descriptor • *i* index • *w* word
[bValue]	Items inside square brackets are optional.
…	Ellipses in syntax, code, or samples indicate 'and so on...' where additional optional items may be included (defined by the developer).
{this (0) | that (1)}	Braces and a vertical bar indicate a choice between two or more items or associated values.
`Collection` **`End Collection`**	This font (`Courier`) is used for code, report descriptor, pseudo-code, and samples.

This document is intended to provide a set of general and unambiguous rules for implementing the HID protocol over SPI for a DEVICE, with the goals of hardware software compatibility (including software reuse), performance and power management.

In this specification document, the following symbols are used to identify required and optional features.

Symbol	Description
[M]	Mandatory to support in hardware and software
[O]	Optional to support in hardware and/or in software
[C]	Conditional support (required to fully implement an optional feature)

1.4 RELATED DOCUMENTS

The following are related documents and provide guidance and background on HID.

Specification Name	Specification Location	Notes
HID over USB	USB.org hid1_11.pdf	This is the first document that a new reader should review.
HID over I²C	HID over I²C specification	Provides information on implementing HID over I²C
HID Usage Tables	https://usb.org/document-library/hid-usage-tables-12	Provides a summary of the Usage Tables that can be leveraged over any transport.
SPI information	SPI page on Wikipedia	Information about the serial peripheral interface.
ACPI 6.3 Specification	ACPI 6.3	This specification describes the structures & mechanisms to design operating system-directed power management & advanced configuration architectures.

2 INTRODUCTION

Why use HIDSPI? There's HID over I2C and HID over USB, why HID over SPI. SPI offers the following features.

- Faster than I2C – more bandwidth, higher clock rates.
- Low latency.
- Easy and inexpensive to implement in hardware.
- Works well for a device that is integrated into the platform and is not removable.

This document describes how to use Human Interface Device (HID) class devices over a simple peripheral bus transport, with an immediate focus on SPI. This specification shares some similar concepts with the USB HID specification, but they are not reused and are not intended to be identical. The HID USB, HID I^2C, or HID Bluetooth Specifications are recommended pre-reading for understanding the content of this document. See the referenced documents section at the beginning of this document. The HID class consists primarily of devices that are used by humans to control the operation of computer systems. Typical examples of HID class devices include:

• Keyboards and pointing devices; for example, standard mouse devices, trackballs, and joysticks

• Front-panel controls; for example, knobs, switches, buttons, and sliders

• Controls that might be found on devices such as telephones, VCR remote controls, games or simulation devices, for example, data gloves, steering wheels, phone's keypads and rudder pedals

• Devices that may not require human interaction but provide data in a similar format to HID class devices, for example, bar-code readers, thermometers or other forms of sensors.

The HID protocol was originally targeted at human interface devices; however, HID protocol is very useful for any application that requires low-latency input-output operations to an external interface, and the ability for that device to describe itself. Many typical HID class devices include indicators, specialized displays, audio feedback, and force or tactile feedback.

The HID protocol is an asymmetric protocol that identifies roles for the **HOST** and the **DEVICE**. The protocol will define a format (**Descriptors**) for the DEVICE to describe its capabilities to the HOST. Once the HOST understands the format of communication with the DEVICE, it programs the DEVICE for sending data back to the HOST. The HID protocol also identifies ways of sending data to the DEVICE as well as status checks for identifying the current state of the device.

The remainder of this protocol specification is broken out in to the following parts.

- **Scenarios** – A brief description of the potential scenarios and the goals of this specification to address existing problems around these scenarios.
- **Descriptors** – A summary of the data elements that will be exchanged between the HOST and the DEVICE to enable enumeration and device identification.
- **Reports** – A summary of data elements that will be exchanged between the HOST and the DEVICE to enable transfer of data in the form of INPUT and OUTPUT reports.
- **Requests** – A summary of the commands and response between the HOST and the DEVICE.

- **Power Management and Error Handling** – A summary on the different commands that are sent from HOST to DEVICE to set/get state information and to address any protocol errors that may occur over the selected transport.

The Appendix section provides examples and for a specific device end to end.

3 SCENARIOS

The following section provides example of user and developer scenarios that are addressed via this protocol specification.

Software Developer Scenario (HID Touch Screen)

Consider a scenario where independent software vendor (ISV) is developing software for a touch screen. Perhaps the screen is getting too large to work well with I2C – too much data. With the introduction of HID over SPI, the ISV can leverage their existing software to work with the new HID SPI touch screen. This solution also allows the operating system to leverage existing software support for touch screens on a new transport.

The expected experience and the associated benefits are as follows:

- Ability to leverage existing software that is HID ready. No special drivers needed to talk a vendor proprietary protocol for SPI touch screens.
- Ability to reuse the ISV's regression tests and validation test environment for HID.
- The software driver support for the touch screen over HID on the HOST can work seamlessly with the internal or the external sensors.
- Ability to reuse the ISV's simulation environment for HID.

4 DEVICE-HOST CONNECTION

This section of the specification identifies the physical connection/communication interface which is to be supported by both the HOST and DEVICE.

4.1 SPI BUS CHARACTERISTICS

The DEVICE and HOST must both support the same SPI interface and characteristics.

SPI encompasses regular SPI, dual SPI, quad SPI, and double data rate. The HOST and DEVICE must both support the same variant of SPI, as well as the same clock polarity, phase, and frequency. When the protocol is operating in Quad or Dual SPI mode, the controller must transfer the opcode in a single SPI mode, and all remaining bytes (including address) in Dual or Quad SPI mode.

If the DEVICE is ACPI-enumerated by the HOST, the specific SPI bus characteristics will be specified in ACPI (including Bus Speed, Clock Polarity, and Clock Phase).

4.2 SCHEMATIC LAYOUT

The following is an example block diagram of the HOST-DEVICE interface, when connected over a typical single-SPI bus configuration.

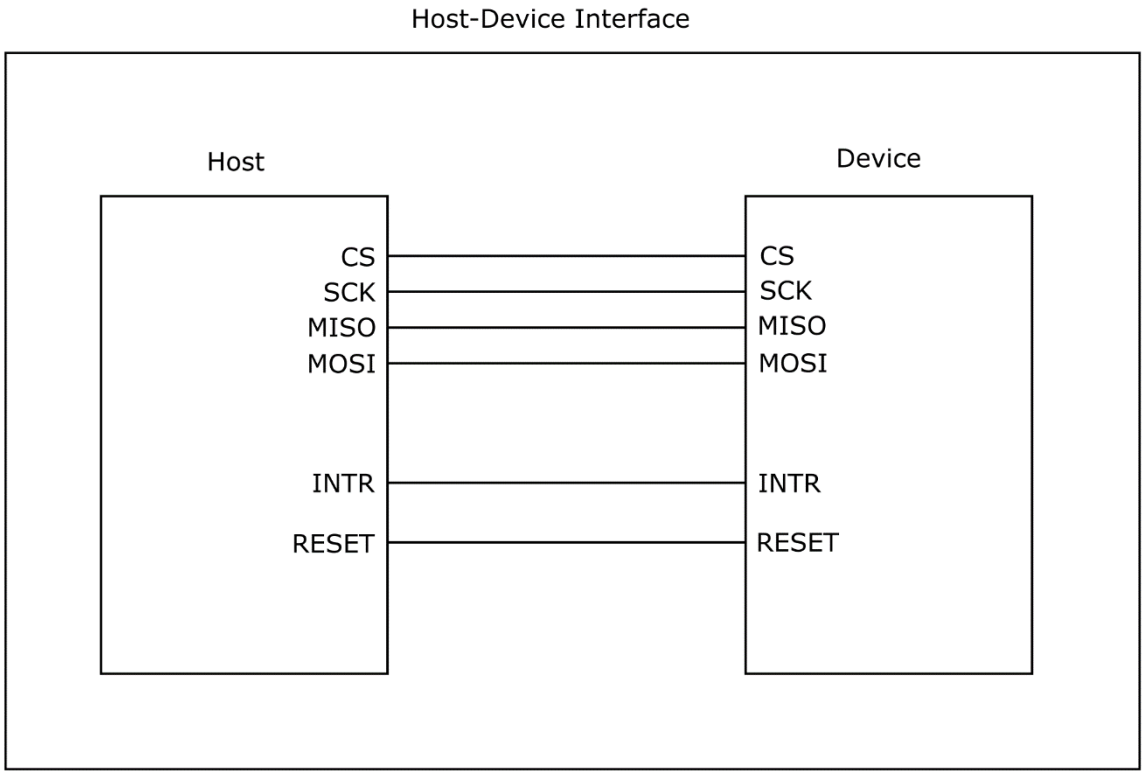


Figure 1: HOST-DEVICE Interface Layout

Notes

- HID SPI peripheral (DEVICE) must leverage the regular SPI lines.
- HID SPI peripheral must provide a dedicated falling (HIGH-TO-LOW) edge triggered interrupt line to indicate when data or status is ready. This interrupt line must be pulled HIGH by hardware otherwise, meeting both HOST and DEVICE electrical requirements.
- HID SPI peripheral must provide a dedicated chip select line (active LOW)
- HID SPI peripheral must provide a dedicated reset line, driven by the HOST, which, when toggled (pulled LOW for at least 10ms, normally HIGH), will have the effect of resetting the device. If a HID SPI peripheral is enumerated via ACPI, the device ASL configuration must expose an ACPI FLDR (_RST) method to control this line.
- The HOST must be able to support the maximum transfer size required by the device in a single transition of the CS line.
- The HID SPI protocol operates in a half-duplex fashion (even on full-duplex SPI buses such as single SPI)
- Additional details around the mechanical & electrical requirements are beyond the scope of this specification.

4.3 BYTE ORDERING

All multiple byte fields (e.g. in standard descriptors, requests, responses, etc), with the exception of the ADDRESS field (big endian, MSB first), are to be interpreted in little-endian notation. The data bit length is 8 bits, 1 byte.

Data sent or received on the SPI bus is goes byte by byte starting at the lowest address first. Each byte of data is then shifted onto or from the SPI bus bit by bit starting with the most significant bit.

4.4 OVERALL HID SPI THROUGHPUT

The table below identifies the maximum size of a report (based on available bus bandwidth) that a HID SPI device can send to the HOST for several speeds using standard SPI, not double or quad SPI or double data rate. Each cell in the table is calculated as follows.

$$Max\ Report\ Size\ in\ bytes = \frac{Bus\ Speed}{Report\ Rate} * \frac{1byte}{8\ bits}$$

It is an exercise to the system integrator to identify and program the correct SPI Bus Speed based on the HID device(s) connected to that SPI controller. Do note that the table below is a theoretical maximum and a system implementer should anticipate about 50% of the values below. This table is meant to be a high-level description of the perceived report sizes and not meant to be an accurate value.

Rate of Sending Reports	12 MHz	24 MHz	48 MHz
1KHz (1ms)	1.5 KBytes	3 KBytes	6 KBytes
100Hz (10ms)	15 KBytes	30 KBytes	60 KBytes
10Hz (100ms)	150 KBytes	300 KBytes	600 KBytes

The section below provides a more accurate calculation of HID SPI efficiency for the most common case, the input report.

- An input report consists of n fragments, where n greater than or equal to 1. Each fragment contains the following.
  - Every input report is initiated with an interrupt.
  - Time, t1, is required to process the interrupt and to initiate reading the input report header.
  - The input report header is 9 bytes.
  - Time, t2, is required to process the input report header and initiate reading the input report body.
  - The input report body is 8 bytes plus the input report.

This leads to the following formula to consider latencies and overhead when calculating the max report size.

$$Max\ Report\ Size\ in\ bytes = Bus\ Speed\left(\frac{1}{Report\ Rate} - n(t1 + t2 + overhead\ time)\right)\frac{1byte}{8\ bits}$$

$$= Bus\ Speed\left(\frac{1}{Report\ Rate} - n\left(t1 + t2 + \frac{17\ bytes * 8}{Bus\ Speed}\right)\right)\frac{1\ byte}{8\ bits}$$

The table below shows a more accurate representation of the maximum report size in bytes based on report rate, bus frequency, number of fragments per input report and latencies. The table below uses 1msec as the time from interrupt assertion to reading data (t1) and 100 usec as the time from reading the input report header to starting to read the input report (t2). Negative numbers in the table below show that there is not enough time to transfer any data.

Report Rate	12 MHz Bus Speed n = 1	12 MHz Bus Speed n = 2	12 MHz Bus Speed n = 3	24 MHz Bus Speed n = 1	24 MHz Bus Speed n = 2	24 MHz Bus Speed N = 3
1 KHz	-167 Bytes	-1834 Bytes	-3501 Bytes	-317 Bytes	-3634 Bytes	-6951 Bytes
500 Hz	1333 Bytes	-334 Bytes	-2001 Bytes	-2683 Bytes	-634 Bytes	-3951 Bytes
100 Hz	13 KBytes	12 KBytes	10 KBytes	27 KBytes	23 KBytes	20 KBytes
10 Hz	148 KBytes	147 KBytes	145 KBytes	297 KBytes	293 KBytes	290 KBytes

As an example, imagine designing a system that requires an 8K byte input report and a report rate of 100 Hz. The system also has a maximum input length of 4K bytes. From the table above the system could use two fragments, n = 2, or three fragments, n = 3, at either 12 or 24 MHz and have margin. This is also assuming the latencies, t1 and t2, used to generate the chart above are reasonable for the system.

5 HOST-DEVICE DISCOVERY

The specification defines two ways in which a device may be logically connected to a host: enumeration through ACPI (allowing use of a generic ACPI compliant SPB resource), or a platform specific controller.

5.1 PLATFORM SPECIFIC CONTROLLER

A platform specific hardware and software implementation of the HID SPI controller or protocol may be used on the HOST. In this case, mechanisms of enumeration and HOST-side configuration are not defined by the specification and should be defined by the platform vendor.

5.2 ACPI ENUMERATION

ACPI enumeration on the HOST allows for the use of generic SPB bus and in-box OS support for the HID SPI protocol. A HID SPI DEVICE may be enumerated on the host in a plug-and-play manner through device definitions made in platform ACPI. The definitions for a HID over SPI device are defined in the following table:

Field	Value	M / O	ACPI object	Format	Comments
Hardware ID	Vendor Specific	M	_HID	String in the format of VVVVdddd (e.g. MSFT0011)	VendorID + DeviceID
Compatible ID	[PNP0C51]	M	_CID	String in the format of ACPIxxxx or PNPxxxx	CompatibleID
Subsystem	Vendor Specific	O	_SUB	String in the format of VVVVssss (e.g. MSFQ1234)	SubVendorID + SubSystemID
Hardware Revision	Vendor Specific	M	_HRV	0xRRRR (2byte revision)	Hardware Revision Number
Current Resource Settings	Vendor Specific	M	_CRS	Byte stream	-SpiSerialBus for access to the device -GpioInt for interrupts.
Device Specific Method	[6e2ac436-0fcf-41af-a265-b32a220dcfab]	M	_DSM	This GUID defines a structure that contains device specific information	
				Function	Item
				0	Buffer: Function bitfield
				1	Integer: Input Report Header Address
				2	Integer: Input Report Body Address (may be the same as the Input Report Header Address)

				3	Integer: Output Report Address
				4	Buffer(1): Read OpCode
				5	Buffer(1): Write OpCode
				6	Integer: Flags Bit 0: RESERVED [INTERNAL:NextReadOpcode] Bit 1-12: Reserved (must be 0) Bit 13: SPI Write Mode - 0b0 – Writes are carried out in Single-SPI mode - 0b1 – Writes are carried out in the Multi-SPI mode specified by bit 14-15 Bit 14-15: Multi-SPI Mode - 0b00 – Single SPI - 0b01 – Dual SPI (single SPI opcode) - 0b10 – Quad SPI (single SPI opcode)
Device Reset Method		M	_RST	ACPI 6.0 7.3.25 compliant device reset method, to be called by the HOST OS as an ACPI FLDR.	

5.2.1 SAMPLE ASL

```
// _DSM - Device-Specific Method
//
// Arg0:    UUID       Unique function identifier
// Arg1:    Integer    Revision ID - Will be 3 for HidSpi V1
// Arg2:    Integer    Function Index (0 = Return Supported Functions)
// Arg3:    Package    Parameters
//
Function(_DSM,{BuffObj, IntObj},{BuffObj, IntObj, IntObj, PkgObj}){
    // HIDSPI UUID
    If(LEqual(Arg0,ToUUID("6E2AC436-0FCF-41AF-A265-B32A220DCFAB"))) {
        //
        // Switch on the function index
        //
        switch(Arg2) {
            case(0) {
                // Switch on the revision level
                switch(ToInteger(Arg1)) {
                    case (3) {
                        // HidSpi v1 : Functions 0-6 inclusive are supported (0b01111111)
                        Return(Buffer(One)) { 0x7F })
                    }
                    default {
                        // Unsupported revision
                        Return(Buffer(One)) { 0x00 })
                    }
                }
            }
            case(1) {
                // Input Report Header address
                Return (0x1000)
            )
            case(2) {
                // Input Report Body address
                Return (0x1004)
            }
            case(3) {
                // Output Report Header address
                Return (0x2000)
            }
            case(4) {
                // Read opcode
                Return (Buffer(1) {0x0B})
            }
            case(5) {
                // Write opcode
                Return (Buffer(1) {0x02})
            }
            case(6) {
                // Flags
                Return (0x0000)
            }
            default {
                // Unsupported function index
            }
        }
    }
    else {
        //
        // No functions are supported for this UUID.
        //
        return (Buffer() {0})
    }
}
Function(_RST) {}  // Placeholder for function-level device reset
```

6 DESCRIPTORS

The following section identifies the key data structures (referred to as descriptors) that need to be exchanged between the HOST and the DEVICE during the startup phase and the data delivery phase.

6.1 DEVICE DESCRIPTOR

The HID Descriptor is the top-level mandatory descriptor that every SPI based HID DEVICE must have. The purpose of the HID Descriptor is to share key attributes of the DEVICE with the HOST. These attributes accurately describe the version that the protocol is compliant towards as well as additional data fields of the device.

6.1.1 DEVICE DESCRIPTOR FORMAT

The device must expose a HID Descriptor with the following fields.

Byte Offset	Field	Size (Bytes)	Type	Description
0	**wDeviceDescLength**	2	WORD	The length, in unsigned bytes, of the complete Device Descriptor. The HOST uses this value to validate the size of the Device Descriptor in Bytes and to retrieve the complete descriptor. It is fixed to 0x18 (24).
2	**bcdVersion**	2	BCD	The version number of the HID over SPI protocol supported, in binary coded decimal (BCD) format. For version 1.0 of the specification, the DEVICE should set this field to 0x0300
4	**wReportDescLength**	2	WORD	The length, in unsigned bytes, of the Report Descriptor.
6	**wMaxInputLength**	2	WORD	This field identifies in unsigned bytes the length of the largest possible HID input (or feature) report to be read from the device. This HID report may be transferred in multiple fragments.
8	**wMaxOutputLength**	2	WORD	This field identifies in unsigned bytes the length of the largest output (or feature) report. An output report contains only one fragment.
10	**wMaxFragmentLength**	2	WORD	This field identifies in unsigned bytes the length of the largest fragment, where a fragment represents the body of an input report. For devices not implementing fragmentation, this value should be at least wMaxInputLength + the size of an Input Report Body (4 bytes), with padding bytes as necessary to make final value a multiple of 4.
12	**wVendorID**	2	WORD	This field identifies the DEVICE manufacturers Vendor ID. Must be non-zero.

14	**wProductID**	2	WORD	This field identifies the DEVICE's unique model / Product ID
16	**wVersionID**	2	WORD	This field identifies the DEVICE's unique version.
18	**wFlags**	2	WORD	This field is used to specify flags for the DEVICE's operation. **Bit 0**: *NoOutputReportAck* - When set, device does not acknowledge output reports (output report type=0x5) - It is recommended that this flag only be used by devices which require multiple output reports to be received during transmission of a fragmented input report **Bit 1-16**: Reserved
20		4	N/A	This field is reserved and should be set to 0.

The Device Descriptor format in this specification is different from the HID descriptor in other transport specifications (e.g. USB). Only the critical common fields for the HID descriptor across transports are captured in the table. It is the responsibility of the HID HOST software to identify and fill in the rest of the fields leveraging OS-specific or ACPI specific data structures.

Notes

1. The *wVendorID, wProductID, wVersionID* fields may not be used to generate the device's hardware ID, if it is generated from the _HID field from ACPI.

6.1.2 DEVICE DESCRIPTOR RETRIEVAL

The following steps shall take place when the device is reset by the host during operation or upon startup (though not when resumed from the SLEEP low power state):

1. The host shall invoke the ACPI reset method to clear the device state.
2. Within 1 second, the device shall signal an interrupt and make available to the host an input report containing a device reset response (see Input Report section below).
3. The host shall read the reset response from the device at the Input Report addresses specified in ACPI.
4. The host shall then write an Output Report to the device at the Output Report Address specified in ACPI, requesting the Device Descriptor from the device.
5. Within 1 second, the device shall signal an interrupt and make available to the host an input report containing the Device Descriptor.
6. The host shall read the Device Descriptor from the Input Report addresses specified in ACPI.
7. The device and host shall then enter their "Ready" states – where the device may begin sending Input Reports, and the device shall be prepared for Output Reports from the host.

6.1.3 DEVICE ERROR HANDLING

When the device detects an error condition, it may interrupt and make available to the host an Input Report containing an unsolicited Reset Response. After receiving an unsolicited Reset Response, the host shall initiate the request procedure from step (4) in the above process.

6.2 REPORT DESCRIPTOR

A Report Descriptor is composed of report items that define one or more top-level collections. Each top-level collection defines one or more HID reports. The Report Descriptor is a mandatory descriptor that is stored in the HID DEVICE and may be retrieved by the HOST on initialization. The concept of a Report Descriptor for HID over SPI is identical to the notion of a Report Descriptor in the original USB HID Class Specification V1.11 or later.

6.2.1 REPORT DESCRIPTOR FORMAT

The Report descriptor is unlike other descriptors in that it is not simply a table of values. The length and content of a Report descriptor vary depending on the number of data fields required for the device's report(s). The Report descriptor is made up of items that provide information about the device. For more details on report descriptors, please review the HID over USB Specification.

> The following descriptors are outside of the scope of this specification and are not supported:
>
> - Physical Descriptor
> - Vendor Specific Descriptor
>
> Vendors wishing to add these or other proprietary descriptors need to expose a secondary SPI device.

6.2.2 REPORT DESCRIPTOR RETRIEVAL

A single Report Descriptor must be stored on the DEVICE. During DEVICE initialization, the HOST gets the length of the report descriptor from **wReportDescLength** in the Device Descriptor.

To retrieve the Report Descriptor from the device, the HOST will first write to the Output Report Address an Output Report with content type specifying a request for report descriptor. The DEVICE shall then signal an interrupt to the host. The HOST shall then read an input report from the Input Report addresses, containing the report descriptor.

If the HOST driver is unable to retrieve the entire report descriptor or if there are errors, a HOST may reset/reinitialize the DEVICE and start the entire enumeration sequence again.

If the DEVICE provides a malformed report descriptor, the behavior of the HOST is not defined and may vary from one operating system to another. It is not necessary for the HOST to re-retrieve a report descriptor.

In general, the HOST software/driver will first retrieve the report descriptor before reading the reports from the device. However, it is acceptable for a HOST to retrieve the reports from a DEVICE without retrieving the Report Descriptor. This may be the case for closed configuration platforms or vertical systems where the format of the reports is known to the HOST at system initialization.

7 REPORT PROTOCOL

The Report is the fundamental data element that is exchanged between the HOST and the DEVICE. The definition of HID reports (including input, output and feature) will be synonymous to their definition and naming convention in the USB HID Class Specification.

The HID over SPI protocol makes use of the two uni-directional report types listed below (Input / Output) for all communication with the device, including feature reports and commands. A Report Type field specified in these reports indicates the type of operation a report is associated with.

The following convention is used for reports.

- **Input Reports** – The uni-directional Report that is sent from the DEVICE to HOST.
- **Output Reports** – The uni-directional Report that is sent from the HOST to the DEVICE.

7.1 INPUT REPORTS

The input reports are generated on the DEVICE and are meant for communication in the direction of DEVICE to HOST over the SPI transport. When the DEVICE has active data, it wishes to report to the HOST, it will assert the Interrupt line associated with the HID protocol on the DEVICE. When the HOST receives the Interrupt, it will then perform the read sequence to read data from the device. The DEVICE shall not signal the interrupt line while a bus transaction is in progress.

An input report is segmented into two distinct parts: the Input Report Header, and the Input Report Body. Each of the two parts is read in a separate operation by the host. Each operation will be preceded by a Read Approval.

7.1.1 INTERRUPT TIMING

When the DEVICE has data to report, it shall set the interrupt line LOW, until the host initiates a bus transaction. The following timing requirements apply:

DEVICE Interrupt Timing Properties	Requirement
Minimum interrupt assertion post falling-edge hold (LOW) duration	At least until the start of an input report (the HOST has written the read approval opcode of the input report header).
Maximum interrupt assertion post falling-edge hold (LOW) duration	There is no maximum duration. The interrupt is falling-edge triggered.
Minimum interrupt duration that interrupt shall be held HIGH before an interrupt (falling-edge)	Defined by the HOST implementation.
Earliest interrupt re-assertion time	No new interrupt (falling-edge) during a input report transmission until after the read approval portion of the input report body.

7.1.2 READ APPROVAL

The host transmits a read approval to signal the device to prepare for the host to read data from the device. The host transmits a read approval before the input report header and before the input report body. The Input Report Address (header or body) and READ opcode are retrieved from ACPI.

Read Approval		
Byte offset	**# bytes**	**Field**
0	1	Opcode (READ)
1	3	Input Report Address (header or body) (big-endian)
4	8 clock cycles (one byte single-SPI, two bytes dual-SPI, four bytes quad SPI)	Placeholder Bytes – (0b1 x 8 cycles)

7.1.3 INPUT REPORT HEADER

In response to an interrupt, the host shall conduct a bus transfer in order to retrieve the Input Report header from the device. It shall do this by transmitting a Read Approval (0x05 bytes) to the device, and reading in an Input Report Header (0x04) bytes as part of a single 9-byte SPI transaction.

Input Report Header			
Byte offset	**# bytes**	**bits**	**Field**
0	1 byte	3:0	Version
		7:4	Reserved (must be 0)
1	2 bytes	13:0	Input Report Length
		14	Last Fragment Flag
		15	Reserved (must be 0)
3	1 byte	7:0	Sync Constant

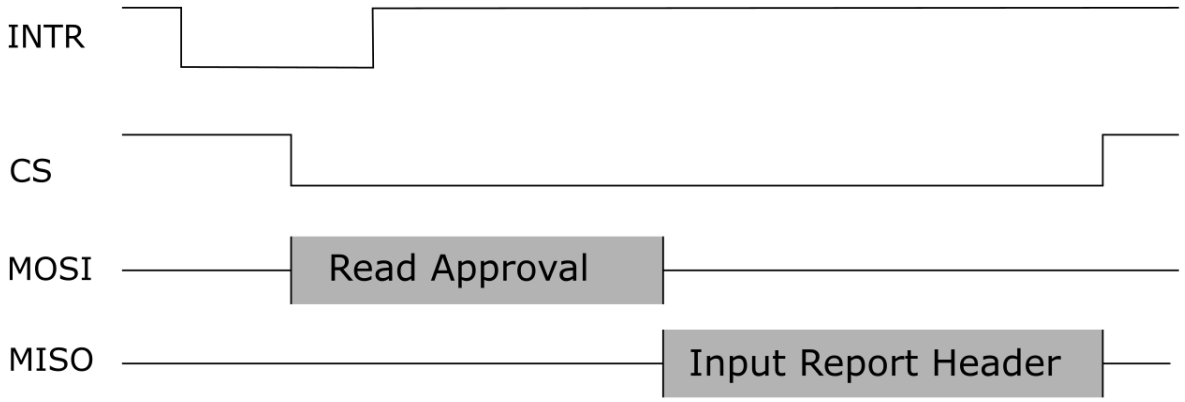
Field Summary:

1. Version – protocol version (must be 0x3)
2. Input Report Length - the length (in number of bytes divided by 4) of the Input Report Body. This makes the maximum input fragment length equal to 64K bytes. A HOST may support a smaller maximum input report length, provided the maximum meets or exceeds that required by a given DEVICE.
   If there is more than one fragment then each fragment will contain Input Report Length bytes of data and the last fragment will have pad bytes added to achieve modulo 4 = 0 length if necessary. If there is only one fragment then it will have pad bytes added to achieve modulo 4 = 0 length if necessary. In a fragmented transfer, the last fragment is the only fragment that will have pad bytes.
3. Last Fragment Flag – A value of one indicates this is the last, or only fragment. A value of zero indicates additional fragments are to follow.
4. Sync constant – used to validate input report header. Must be set to 0x5A by the DEVICE [INTERNAL: unless the NextReadOpcode flag is set (nonzero) in ACPI. If the

NextReadOpcode flag is set this field will contain the opcode to use to read the input report body]. The HOST is not required to validate this field, though it is advised.

7.1.4 RETRIEVING THE INPUT REPORT HEADER

The diagram below shows the process to retrieve the Input Report Header. The process begins when the device asserts the interrupt. The host then asserts chip select, CSN, and performs a 9 byte transfer. In the first 5 bytes of the transfer the host transmits the Read Approval to the device. In the last 4 bytes of the transfer the device returns the Input Report Header to the host.



7.1.5 INPUT REPORT BODY

After the host has read and validated the Input Report Header, it shall issue another SPI transaction (complete with preceding Read Approval) for the Input Report Body, from the Input Report Body Address, as provided in ACPI. The number of bytes to be read is taken from the Input Report Length field of the Input Report Header. If the size of the Input Report Body is less than that reported by the Input Report Header, pad bytes must be added at the end of the report.

An input report will consist of one or more fragments. The first four bytes of the first fragment contains information about the input report as described below. All subsequent fragments only contain data, until the last fragment which may contain pad bytes to make the length zero mod 4.

Read Approval		
Byte offset	**# bytes**	**Field**
0	1	Opcode (READ)
1	3	Input Report Body Address
4	8 clock cycles (one byte single-SPI, four bytes quad SPI)	Placeholder Bytes – (0b1 x 8 cycles)

Input Report Body *(first and only fragment / non-fragmented transfer)*		
Byte offset	**# bytes**	**Field**
0	1 byte	Input Report Type
1	2 bytes	Content Length
3	1 byte	Content ID
4	n bytes	Content
4+n	0-3 bytes	Padding

Input Report Body *(first fragment – multi-fragment transfer)*		
Byte offset	**# bytes**	**Field**
0	1 byte	Input Report Type
1	2 bytes	Content Length
3	1 byte	Content ID
4	n bytes	Content

Input Report Body *(subsequent fragments – multi-fragment transfer)*		
Byte offset	**# bytes**	**Field**
0	n bytes	Content

Input Report Body *(last fragment – multi-fragment transfer)*		
Byte offset	**# bytes**	**Field**
0	n bytes	Content
n	0-3 bytes	Padding

Field Summary:

1. Input Report Type – describes the type of data this input report contains.
    a. 0x0 – Reserved
    b. 0x1 – Data. This indicates an unsolicited HID input report being sent by the device.
    c. 0x3 – Reset response.
    d. 0x4 – Command response. Signals the command has completed.
    e. 0x5 – Get feature response. The input report contains a HID report relating to the feature requested by the host.
    f. 0x7 – Device Descriptor. The input report contains the device descriptor.
    g. 0x8 – Report descriptor. The input report contains the HID report descriptor.
    h. 0x9 – Set feature response. Signals the set feature has completed.
    i. 0xA – Set output report response. Signals the set output report has completed.
    j. 0xB – Get input report response. Provides a way for HOST to manually fetch an input report.
2. Content Length – the size of the Content field, across all fragments in unsigned bytes.
    a. This does not include the size of any other fields, including Input Report Type, Content ID, or Content Length.

   b. Reset responses, and output report / set feature acknowledgements have a Content Length of 0, representing an empty content field.
   c. This does not include any padding bytes added to the end of the report body.
3. Content ID
   a. For descriptors, and reset responses, this field shall be 0x00.
   b. For HID content, this will contain the Report ID of the Set Feature, Get Feature, or Input/Output Report of the content. If the device only supports a single report, this field shall be zero.
   c. For command acknowledgement and responses, this field shall contain the Content ID of the command sent by the host.
4. Content – the report, descriptor, or other raw data to be consumed by the host.
5. Padding bytes – The DEVICE shall add 0-3 pad bytes to the end of the Input Report Body, such that the total length of the Input Report Body is a multiple of 4 bytes. These pad bytes will not be included in the Content Length field.

7.1.6 INPUT REPORT FRAGMENTATION

The DEVICE may report a single HID input report (of Input Report Type Data) as a number of smaller HID SPI Input Reports in the form of fragments. Fragmentation may be used by a device to perform flow control, or to allow output reports to be written with lower latency during transmission of a large input report. Fragmentation may only be used for Data – when Input Report Type = 0x1. For all other Input Report Types, the entirety of the report must be sent in a single fragment.

Fragments have the same header-followed-by-body structure as normal input reports. The first fragment will contain the Content Length & Content ID fields in the Input Report Body, as well as a portion of the report, and following fragments, Input Report Body will contain only subsequent report Content (no Length/ID fields).

The HOST may write an output report (including a set feature) to the DEVICE between incoming fragments. The DEVICE shall not acknowledge or respond to this output report until all fragments in a transfer have been sent. However, the DEVICE shall process / acknowledge the output report within the defined response timeout.

The HOST will read the first fragment of a HID report when the device asserts an interrupt as it would as a regular Input Report. The HOST will read subsequent fragments upon subsequent interrupts - the DEVICE shall assert an interrupt for every fragment in a transfer.

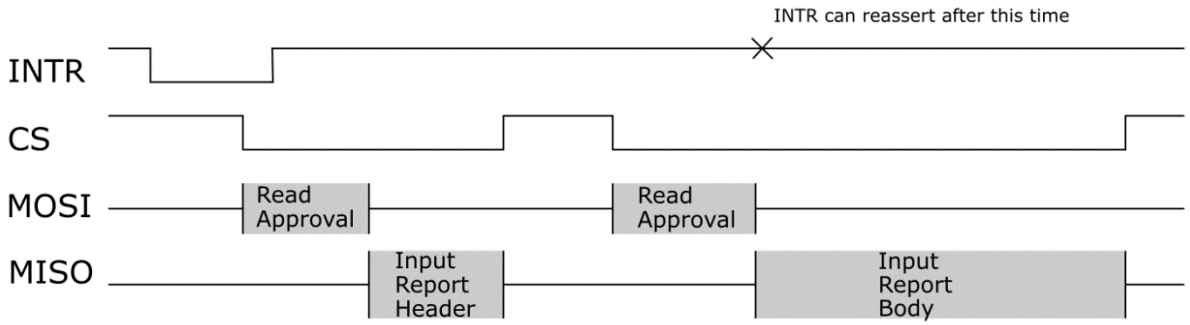
The DEVICE shall raise an interrupt for a subsequent fragment within 1 second of completion of the bus transaction of a prior fragment.

7.1.7 RETRIEVAL OF INPUT REPORTS

The input report is one of the most frequent forms of communication between the DEVICE and HOST. An input report can consist of one or more fragments. The following sequence of operations explains the process of retrieval of the Input Report.

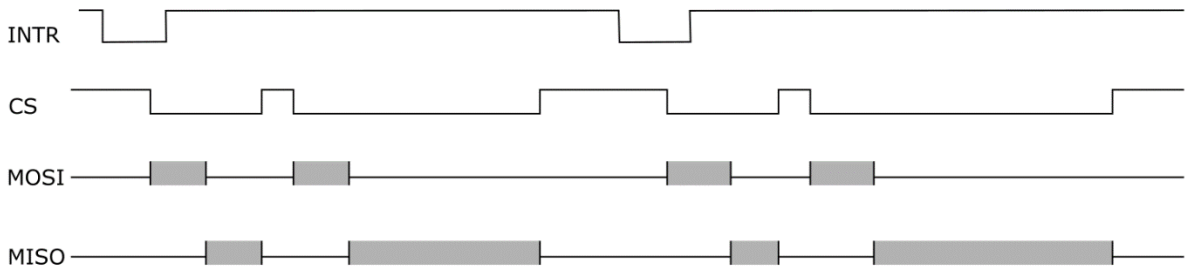
7.1.7.1 SINGLE FRAGMENT EXAMPLE (SINGLE-SPI)

Sequence	HOST Side	DEVICE Side
Step 1		DEVICE asserts the Interrupt indicating that it has an Input Report to send to HOST
Step 2	HOST transmits a 5 byte Read Approval and reads the 4 byte Input Report Header as one 9 byte transfer.	DEVICE reads the 5 byte Read Approval and returns the 4 byte Input Report Header. The last fragment bit in the Input Report Header is set.
Step 3	HOST transmits a 5 byte Read Approval and reads the Input Report Body fragment as one transfer	Device reads the 5 byte Read Approval and returns the Input Report Body. The first 4 bytes of the input report body contain the Input Report type, Content Length and Content ID. The remaining bytes contain the report content and padding (if required). The length of the Input Report Body is Input Report Length from the Input Report Header, padded out with zeros if necessary.



7.1.7.2 MULTIPLE FRAGMENT EXAMPLE (SINGLE-SPI)

Sequence	HOST Side	DEVICE Side
Step 1		DEVICE asserts the Interrupt indicating that it has an Input Report to send to HOST
Step 2	HOST issues the 5 byte Read Approval and reads the 4 byte Input Report Header as one 9 byte transfer.	DEVICE reads the 5 byte Read Approval and returns the 4 byte Input Report Header. The last fragment bit in the Input Report Header is clear.
Step 3	HOST issues a Read Approval and reads the Input Report Body fragment as one transfer	Device reads the 5 byte Read Approval and returns the Input Report Body. The first 4 bytes of the input report body contain the Input Report type, Content Length and Content ID. The remaining bytes contain the report content. The length of the Input Report Body is Input Report Length from the Input Report Header. There should be no padding since there are more fragments to read. Example: if fragment size / Input Report Length is 64 bytes, the first fragment will contain 4 bytes for Report Type, Content Length and Content ID, and the remaining 60 bytes will contain HID report data. The reported Content Length, which is only present in the first fragment, will be 64.
Step 4		DEVICE asserts the Interrupt, indicating that it has the next fragment ready to send to the HOST.
Step 5	HOST issues a 5 byte Read Approval and reads the 4 byte Input Report Header as one 9 byte transfer.	DEVICE reads the 5 byte Read Approval and returns the 4 byte Input Report Header. The last fragment bit in the Input Report Header is set.
Step 6	HOST issues a Read Approval and reads the Input Report Body fragment as one transfer	Device reads the 5 byte Read Approval and returns the Input Report Body. The first 4 bytes of the input report body do not contain the Input Report Type, Content Length or Content ID because this is not the first fragment. The length of the Input Report Body is Input Report Length from the Input Report Header, padded out with zeros if necessary.



7.2 OUTPUT REPORTS

The output report is generated on the HOST and is meant for communication in the direction of HOST to DEVICE over the SPI transport. When the HOST has active data it wishes to report to the DEVICE, it will write the output report to the Output Report Address. The host sends an output report to the device to request data from the device, or send the device a command.

The host will always expect a response to each output report, except in the case where the flag is set in the device descriptor indicating that for output reports of type 0x05, the DEVICE is not expected to acknowledge. For some output reports (GET_FEATURE, descriptor requests, some commands), the host shall always expect an Input Report that contains data. For other output reports the host will expect an empty Input Report where the Input Report Type field indicates a it is a response and the Content field of the Input Report Body contains no data. There is always both an Input Report Header and Input Report Body in response to an Output Report. When the host writes an output report to the device, the host shall not send any further output reports to the device until a response is received. The device shall respond to an output report within 1 second of completion of the bus transaction.

The HOST writes all output reports to the Output Report Address as specified in ACPI.

7.2.1 OUTPUT REPORT STRUCTURE

The data structure of an output report is as follows:

Output Report			
Byte offset	**# bytes**	**bits**	**Field**
0	1	7:0	Opcode
1	3	23:0	Output Report Address (big-endian)
4	1	7:0	Output Report Type
5	2	15:0	Content Length
7	1	7:0	Content ID
8	n		Content
8 + n	0-3		Padding bytes

Field Summary:

1. Opcode – is mandatory and value is limited according to SPI type (indicating WRITE).
2. Output Report Address - As specified in ACPI.
3. Output Report Type – Content type of the Output Report.
    a. 0x00 – Reserved
    b. 0x01 – Request for device descriptor
    c. 0x02 – Request for report descriptor
    d. 0x03 – HID SET_FEATURE content.
    e. 0x04 – HID GET_FEATURE content.
    f. 0x05 – HID OUTPUT_REPORT content.
    g. 0x06 – Request for input report (HID Get Report)
    h. 0x07 – Command content

4. Content Length – the size of the Content field.
    a. This does not include the size of any other fields, including Output Report Type, Content ID, or Content length.
    b. Descriptor requests, set features, and other reports not containing any Content, have a Content Length of 0, representing an empty content field.
    c. This does not include any padding bytes added to the end of the report body.
5. Content ID
    a. For descriptor request, this field shall be 0x00.
    b. For HID content, this will contain the Report ID of the Set Feature, Get Feature, or Input/Output Report of the content. If the device only supports a single report, this field may be zero.
    c. For commands, this field will contain a command opcode (see command section).
6. Content – the raw HID report (for Set Feature & Output Reports) or command parameters
7. Padding bytes – The HOST shall add 0-3 pad bytes to the end of the Output Report such that the total length of the Output Report is a multiple of 4 bytes. These pad bytes will not be included in the Content Length field.

7.2.2 COMMANDS

The host may issue a command to the device in a form of an output report with content type 0x07. The host will set the content ID field of the output report containing the command to the command ID specified in the list below.

There are slightly different semantics to how each command shall be handled by the device. The device shall respond to a command (with the exception of Set Power SLEEP and Set Power OFF) by asserting the interrupt line and making available an input report of content type 0x04, with the content ID field matching the content ID of the command sent by the host.

List of commands and command IDs:

1. 0x00 – Reserved
2. 0x01 – Set Power
    a. Command content (from HOST) shall contain a single byte value:
        i. 0x00 – Reserved
        ii. 0x01 – Set power ON
        iii. 0x02 – Set power SLEEP
        iv. 0x03 – Set power OFF
    b. DEVICE shall respond to a Set Power ON command. DEVICE shall not respond to a Set Power SLEEP, or Set Power OFF command.
    c. Command response content (from DEVICE) shall contain the same single byte value as received from the HOST in the command.
    d. See Power Management section of this document for more information on how power commands are used.
3. 0x02 – 0xEF – Reserved
4. 0xF0 – 0xFF – Reserved for use by platform specific controller implementations

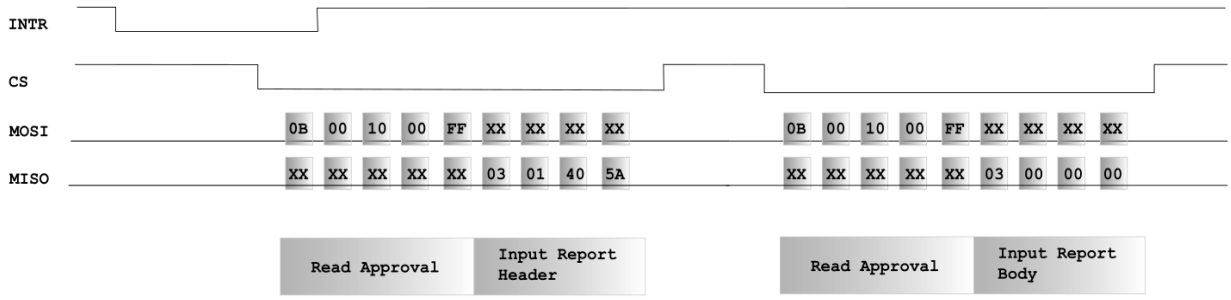
7.2.3 SENDING OUTPUT REPORTS

The following sequence of operations explains the process of sending the Output Report to the DEVICE.

Sequence	HOST Side	DEVICE Side
Step 1	HOST writes an Output Report Header followed by the Output Report Body	
Step 2		DEVICE consumes Output Report
Step 3		DEVICE asserts interrupt within 1 second after completion of the bus transaction.
Step 4	HOST issues a Read Approval and reads the Input Report Header	The Input Report Type field of the Input Report Header will contain a response type indicating a response to the Output Report Type field of the output report.
Step 5	HOST issues a Read Approval and reads the Input Report Body	The Content field of the Input Report Body is empty for a response, or it contains the requested data.

7.3 EXAMPLES (SINGLE-SPI)

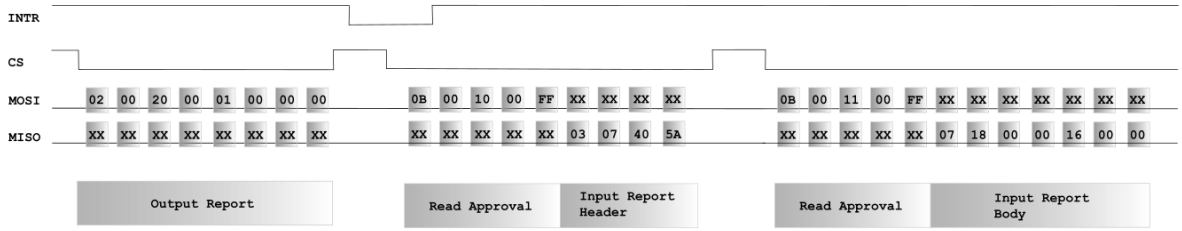
1. Reading the device reset



   a. Device asserts INTR
   b. Host reads Input Report Header
      i. Read Approval:
         1. Opcode: 0x0B (READ) (or other opcode as specified in ACPI)
         2. Input Report Header Address: 0x001000 (or other address as specified in ACPI)
         3. Pad Byte: 0xFF
      ii. Input Report Header
         1. Version: 0x3
         2. Length: 0x001 (4 * 1 = 4)
         3. Last Fragment Flag: Set
         4. Sync Byte: 0x5A
   c. Host reads Input Report Body

i. Read Approval
   1. Opcode: 0x0B (READ) (or other opcode as specified in ACPI)
   2. Input Report Body Address: 0x001000 (or other address as specified in ACPI)
   3. Pad Byte: 0xFF
ii. Input Report Body
   1. Input Report Type: 0x3 (RESET)
   2. Content Length: 0x0000
   3. Content ID: 0x00

2. To request device descriptor



a. Host sends output report to request device descriptor
   i. Opcode: 0x02 (WRITE)
   ii. Output Report Address: 0x002000 (or other address as specified in ACPI)
   iii. Output Report Type: 0x01 (Device descriptor request)
   iv. Content Length: 0x0000
   v. Content ID: 0x00 (Device descriptor request)
b. Device asserts interrupt to indicate the Device Descriptor is ready
c. Host reads the Input Report Header
   i. Read Approval
      1. Opcode: 0x0B (READ) (or other opcode as specified in ACPI)
      2. Input Report Header Address: 0x001000 (or other address as specified in ACPI)
      3. Pad Byte: 0xFF
   ii. Input Report Header
      1. Version: 0x3
      2. Input Report Length: 0x007 (4 * 7 = 28 bytes)
      3. Last Fragment Flag: Set
      4. Sync Constant: 0x5A
d. Host reads Input Report Body
   i. Read Approval
      1. Opcode: 0x0B (READ) (or other opcode as specified in ACPI)
      2. Input Report Body Address: 0x001000 (or other address as specified in ACPI
      3. Pad Byte: 0xFF
   ii. Input Report Body
      1. Input Report Type: 0x7 (Device Descriptor)
      2. Content Length: 0x0018 (24)
      3. Content ID: 0x00

4. 24 bytes of content (device descriptor)
5. 0 bytes of padding as body length is 4+24=28 bytes

7.4 DESIGN CONSIDERATIONS

There are some race conditions that must be taken into consideration.

- Host requests a descriptor after device asserts interrupt.
  The condition that could occur is the device asserts an interrupt to indicate an input report is ready. However, the host is already in the process of sending an output report to request a descriptor from the device. The host will send the output report to request the device descriptor and then process the interrupt to read the input report to handle the interrupt raised by the device. The device can continue sending input reports as long as the device raises an interrupt within the timeout to return the requested descriptor.

7.5 FEATURE REPORTS

Getting and setting features is implemented using Output Reports and Input Reports, where the report type field is used to denote that a feature report is being operated on.

7.6 HID REPORT OPERATIONS

The table below provides an overview of the HID report operations supported by the HID SPI protocol and the input and output reports which are used to carry out the operation.

HID Report Type	Operation	Output Report Type	Input Report Type
Input Report	GET	0x06 (Request – empty content)	0x0B (Response)
Input Report	SET (Not supported)	N/A	N/A
Input Report	INTERRUPT IN	N/A – No request	0x01
Feature Report	GET	0x04 (Request – empty content)	0x05 (Response)
Feature Report	SET	0x03	0x09 (Acknowledgement – empty content)
Output Report	GET (Not supported)	N/A	N/A
Output Report	SET	0x05	0x0A (Acknowledgement – empty content)

8 POWER MANAGEMENT

The following section identifies the details around HOST and DEVICE power management. The following sections identify the two specific scenarios for power management:

- Device Initiated Power Optimizations
- Host Initiated Power Optimizations

8.1 DEVICE INITIATED POWER OPTIMIZATIONS (DIPO)

The DEVICE is responsible for optimizing its power utilization in the absence of any power settings from the HOST. This will enable the DEVICE to enter its lowest power state on its own (i.e. without HOST intervention) while ensuring that the DEVICE is able to continue to communicate with the HOST in a timely manner.

To correctly comply with Device Initiated Power Optimizations, the following shall be observed:

- The DEVICE is responsible for preserving its state across its low power mode(s).
- All DEVICE power optimizations must be transparent to the HOST and end user(s).
- The DEVICE shall respond to all requests from HOST in a timely manner (i.e. the low power mode(s) shall be transparent to the HOST). The DEVICE is responsible for bringing itself to higher power modes on user or system interactions in a timely manner.
- The DEVICE shall notify the HOST on any INPUT report changes in a lossless manner (i.e. no events should be lost, or deleted by the DEVICE).
- The power states described in HIPO do NOT apply to DIPO.

Scenarios where DEVICE Initiated Power Optimizations are generally deployed include the following scenarios.

- ***Scenario 1 – DEVICE is idle for a short interval of time***: The DEVICE determines that it is idle (e.g., accelerometer is lying flat on a table and there is no measurable change in acceleration) and puts itself in to its lowest power state where it reduces its internal sensing frequency until motion is reinitiated. As soon as motion starts, data is immediately sent to host.
- ***Scenario 2 – DEVICE reduces its sensing frequency***: The DEVICE reduces the frequency at which it scans for data (e.g., if the digitizer does not sense a human finger is present, it samples at 10Hz as compared to 100Hz. However once user interaction is detected, it increases its sensing interval).

8.2 HOST INITIATED POWER OPTIMIZATIONS (HIPO)

The HOST is responsible for optimizing the power of the overall system and the DEVICE. This method of power optimization is to be used when the HOST wishes to provide power optimization notifications to devices.

The following power states are defined for HIPO and are not to be confused with vendor specific DIPO states.

- ON
- SLEEP (DEVICE may wake the system)
- OFF (DEVICE cannot wake the system, power may be removed from the DEVICE)

In the ON state, device behaves normally, and may use DIPO to reduce power consumption. The DEVICE is responsible for being in the ON state when HIDSPI communications are initiated, after a HOST initiated reset.

The HOST instructs the DEVICE to enter a low power state from the ON state by issuing the defined Set Power command. The HOST will choose to do this based on the existing/configured operating system power policy for the device.

The HOST will place the DEVICE into SLEEP state when platform power policy wishes to allow the DEVICE to wake either itself and/or the system (e.g. wake on a touch gesture). Support for power state SLEEP is optional and shall be indicated to the HOST operating system through ACPI, or in a manner appropriate to the bus for a platform specific controller. Upon receiving a Set Power SLEEP command, the DEVICE shall immediately enter a lower-power state, where it will wait for user interaction and shall not assert interrupts thereafter, except to initiate a wake. If the DEVICE detects input, it shall assert an interrupt, and wait for the host to command it into the ON state by sending a Set Power ON command. The DEVICE shall then respond to the Set Power ON command, at which point it may resume sending input to the HOST.

The HOST will place the DEVICE into the OFF state when communication with the DEVICE is no longer required. ACPI (or platform specific controller) shall be configured to provide a “cold” OFF state. Upon receiving a Set Power OFF command, the DEVICE shall immediately enter its lowest-power state, and stop communication with the HOST. When the HOST wishes to bring the device to the ON state, it shall initiate a reset (no power command is written), at which point the initialization process will begin.

For ACPI enumerated devices, it is required that the following power states are implemented:

- D0 – Normal working state
- D2 – Used for the SLEEP state if supported. DEVICE should indicate wake support from this power state.
- D3 – This should be used for the OFF state. DEVICE should not indicate wake support from this power state.

For platform specific controllers, alternate D-state mappings may be used in order to account for the power requirements of the controller hardware.

The platform level D-state mappings are not visible or communicated to the DEVICE.

The table below identifies the properties a DEVICE and a HOST must follow:

Power State	HOST Responsibility	DEVICE Responsibility
ON	• The HOST is responsible for addressing interrupts and issues IO to the device as necessary.	• The DEVICE is responsible for being in the ON power state after Reset. • The DEVICE shall process, but not provide a response to a SET POWER SLEEP or SET POWER OFF command from the HOST.
SLEEP	• The HOST is responsible for instructing the device to enter SLEEP state. • The HOST is responsible for setting the device into ON state if the DEVICE alerts via the interrupt line. • If a HOST needs to communicate with the DEVICE it MUST issue a SET POWER command (to ON) before any other command.	• The DEVICE must de-assert the interrupt line if asserted, before HIPO. • The DEVICE may send an interrupt to the HOST to request servicing (e.g. DEVICE wishes to remote wake the HOST). The device must then not reassert the interrupt until the HOST has sent a SET POWER command to enter the ON state, which the DEVICE has responded to, at which point the DEVICE should assert the interrupt again to notify the host of an input report (if there is a report pending). • The DEVICE must reduce the power draw to an absolute minimum to maintain state and optionally support remote wake. • The DEVICE must respond to a SET POWER ON command from the HOST.
OFF	• The HOST is responsible for instructing the DEVICE to enter OFF state. • The HOST will direct the platform (via ACPI or otherwise) to put the DEVICE into the OFF state. • The HOST will put the device into this state when it should not be capable of waking itself.	• The DEVICE must de-assert the interrupt line if asserted, before HIPO. • The DEVICE will not be able to initiate wake or provide interrupts in this state. • The DEVICE must reduce the power draw to an absolute minimum. It is not required to maintain state. • The DEVICE should treat an OFF -> ON transition as it would a regular power up.

Scenarios where HOST Initiated Power Optimizations (HIPO) are generally deployed include the following example flows:

Scenario 1 – No application communication with the device (Idle Detection)
The HOST determines that there are no active applications that are currently using the specific HID DEVICE. The HOST is recommended to issue a HIPO command to the DEVICE to force the DEVICE in to a lower power state (e.g. there is no application needed to talk to the gyroscope, so the HOST informs the gyroscope to go into OFF state).

1. The HOST writes a Set Power OFF Command to the DEVICE.
2. The DEVICE enters its lowest power state.

3. Power is removed from the DEVICE by the platform / ACPI

When an application on the HOST wishes to use the DEVICE, the HOST will wake the DEVICE:

1. Power is applied to the DEVICE by the platform / ACPI
2. The HOST initiates a reset
3. The DEVICE responds to reset, entering the ON state, and normal startup proceedings occur

Scenario 2 – System Suspend: The HOST is going into a deep power optimized state and wishes to put all the devices into a low power state also. The HOST is recommended to issue a HIPO command to the DEVICE to force the DEVICE into a lower power state (e.g. a PC enters standby state and wishes to put the consumer control buttons on the keyboard to OFF power state).

1. The HOST writes a Set Power OFF Command to the DEVICE
2. The DEVICE enters its lowest power state
3. Power is removed from the device by the platform / ACPI

When the HOST returns to operating state (e.g. through power button or lid sensor):

1. Power is applied to the DEVICE by the platform / ACPI
2. The HOST initiates a reset
3. The DEVICE responds to reset, entering the ON state, and normal startup proceedings occur

Scenario 3 – Modern Standby: The HOST is entering modern standby and wishes to put the DEVICE into a low power state which allows an input from the DEVICE to wake the system (such as a touch gesture). The host may issue a HIPO command to instruct the DEVICE to enter SLEEP.

1. The HOST writes a Set Power SLEEP Command to the DEVICE
2. The DEVICE enters its lower power state, where it can still detect input and maintain internal state.

If the user performs a wake gesture on the screen:

1. DEVICE asserts a single interrupt to the HOST
2. HOST writes a Set Power ON Command to the DEVICE
3. DEVICE interrupts and provides a Set Power ON Response to the HOST
4. DEVICE resumes providing input to the HOST

The following figure shows the allowed DEVICE power state transitions.

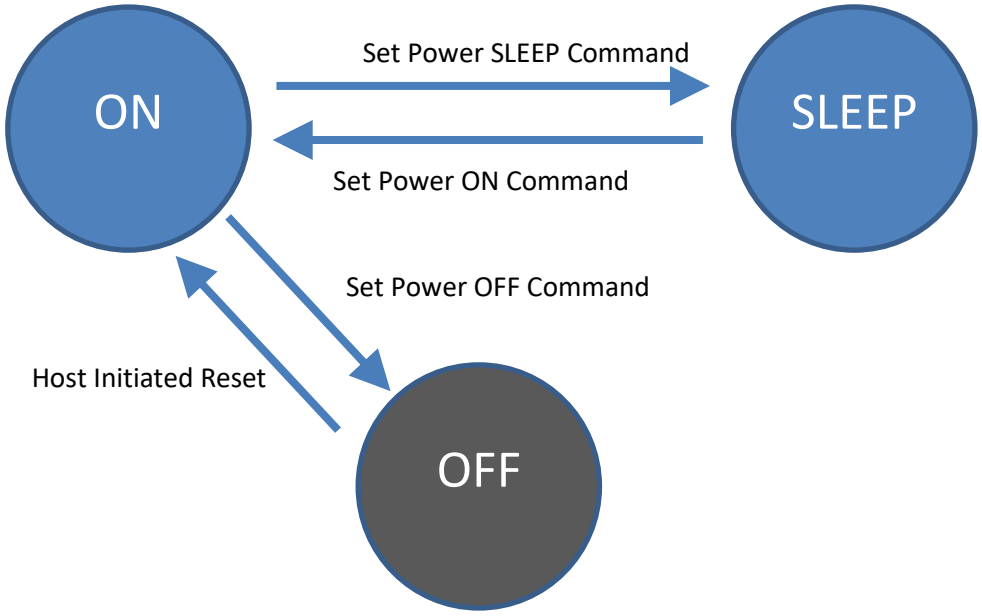


Figure 2: DEVICE Power State Transitions

Notes:

- It is the HOST responsibility to ensure that the HOST Initiated Power Optimization commands comply with the allowed state transitions.
- On the completion of the RESET operation, the DEVICE should reside in the ON Power State.
- The DEVICE is recommended to transition to the new Power State in a reasonable amount of time. This may vary between device types but should never exceed 1sec. Having this limit ensures that HOSTS can power optimize their power in a reasonable timeframe.
- If the DEVICE receives a Power Setting that it is already in (e.g. DEVICE is in ON Power State, and receives a SET_POWER command to turn to ON), the device need not take any action.

9 ERROR HANDLING

The following section summarizes the error detection and handling procedures that the HOST and the DEVICE must follow. Errors on the SPI bus for the purposes of this specification will be broken out into the following categories:

- Protocol Errors
- Timeout Errors

9.1 PROTOCOL ERRORS

Protocol errors are possible on SPI and are more frequent on a poorly designed system. To guard against basic protocol errors, they are further characterized into the following classifications for the purposes of this specification:

- Short Packet Errors
- Bit Level Errors

9.1.1 SHORT PACKET ERRORS

Short packet errors occur when the HOST or the DEVICE does not return the number of bits as identified in the HID SPI protocol request and length field. The HOST is expected to clock in the specified number of bits. The HOST has no way to know if the DEVICE has stopped sending data since the HOST will read whatever happens to be on the bus. The HOST is expected to check sync fields and other fields to see if the data is reasonable. The HOST behavior for unexpected data or invalid data shall be to initiate a reset of the DEVICE.

9.1.2 BIT LEVEL ERRORS

Bit level errors may occur on the SPI bus. These errors are generally as a result of noise on the bus or interference from other buses in the system. This specification does not support CRC or other detection mechanism for single/multiple bit errors on the SPI data line.

However, it is often possible for the HOST parser to identify a malformed report and discard it. It is the responsibility of the HOST HID driver stack to guard against a malformed report that is not conformant with the Report Descriptor. The HOST behavior for unexpected data shall be to initiate a reset of the DEVICE.

9.2 TIMEOUT ERRORS

The HID over SPI protocol is sequential with the expectation that the DEVICE must respond to HOST requests in a timely fashion. In most cases, the responses from DEVICE to HOST will complete in a matter of milliseconds. In the event that the DEVICE is stuck and is unable to revert itself, there is a forced timeout delay after which the HOST may RESET the device and restart operations.

TIMEOUT_HostInitiatedReset = 1 second.

Different HOSTs may allow for proprietary methods to adjust the value of this timeout for their specific devices but it is mandatory for the HOST to support a timeout value (if optimized for the specific system).

9.3 HOST INITIATED RESET

The HOST may reset the DEVICE when an error is detected to re-establish communication with the device. This mechanism is intended for error recovery and should be in response to exceptional event such as re-establishing communication with a device that was exposed to an ESD discharge.

10 SIZES AND CONSTANTS

The following section identifies the minimum & maximum sizes of various parameters:

Data Element	Minimum Value	Maximum Value	Notes
Report Descriptor	0 Bytes	65535 Bytes (2^{16}-1)	The maximum length is determined by the size of ***wReportDescLength***, as defined in the HID Descriptor.
Report (Input, Output)	0 Bytes	65535 Bytes (2^{16}-1)	The maximum length is determined by the size of ***wMaxInputLength*** or ***wMaxOutputLength***, as defined in the HID Descriptor.
Content ID	0	255	

The following table identifies the average size of a report descriptor and the pertinent reports (input, output, and feature) for that device type. This information is NOT specific to SPI and should be roughly identical on any HID Transport.

Device Type	Report Descriptor	Input Report	Output Report	Feature Descriptor
Basic Keyboard	50-60 Bytes	~8 Bytes	~1 Byte	0 Byte
Keyboard with consumer controls)	~200 Bytes	~8 + 2 Bytes*	1 Byte	0 Byte
Basic Mouse	~40 Bytes	~5 Bytes	0 Byte	0 Byte
Accelerometer Sensor	~80 Bytes	~6 Bytes	0 Byte	~4 Bytes
Touchscreen	~100 – 500 Bytes	~20 – 40** Bytes	0 Byte	~2 Bytes

*These devices generally use multiple Top-Level Collections
**This is based on 2-finger touch points per report.

11 EXAMPLE: SAMPLE ACCELEROMETER SENSOR

This example demonstrates how to build a HID Compliance Accelerometer Sensor over SPI.

11.1 ASL LAYOUT

The following table identifies the ASL layout for the Accelerometer Device.

```
Scope (\_SB) {
Device(HIDSPI_DEVICE) {
    Name(_ADR,0)
    Name (_HID, "MSFT1234”)
    Name (_CID, "PNP0C51")
    Name (_UID, 3)
    Name(_HRV, 1)

    Method(_CRS, 0x0, NotSerialized) {
        Name (RBUF, ResourceTemplate () {
            SpiSerialBus (0x0000, PolarityLow, FourWireMode, 0x08,
                          ControllerInitiated, 0x004C4B40, ClockPolarityLow,
                          ClockPhaseFirst, "\\_SB.SPI1", 0x00, ResourceConsumer, ,)
            GpioInt(Edge, ActiveLow, Shared, PullUp, 0,
                    "\\_SB.TGD1", 0 , ResourceConsumer, ,) {40}
        })

        Return(RBUF)
     }

     Function(_DSM,{BuffObj, IntObj},{BuffObj, IntObj, IntObj, PkgObj}){
        // HIDSPI UUID
        If(LEqual(Arg0,ToUUID("6E2AC436-0FCF-41AF-A265-B32A220DCFAB"))) {
            //
            // Switch on the function index
            //
            switch(Arg2) {
                case(0) {
                    // Switch on the revision level
                    switch(ToInteger(Arg1)) {
                        case (3) {
                            // HidSpi v1 : Functions 0-6 inclusive are supported (0b01111111)
                            Return(Buffer(One)) { 0x7F })
                        }
                        default {
                            // Unsupported revision
                            Return(Buffer(One)) { 0x00 })
                        }
                    }
                }
                case(1) {
                    // Input Report Header address
                    Return (0x1000)
                )
                case(2) {
                    // Input Report Body address
                    Return (0x1004)
                }
                case(3) {
                    // Output Report address
                    Return (0x2000)
                }
                case(4) {
                    // Read opcode
                    Return (Buffer(1) {0x0B})
                }
                case(5) {
                    // Write opcode
                    Return (Buffer(1) {0x02})
                }
                case(6) {
```

```
                    // Flags
                    Return (0x0000)
                }
                default {
                    // Unsupported function index
                }
            }
        }
        else {
            //
            // No functions are supported for this UUID.
            //
            return (Buffer() {0})
        }
    }

    Function(_RST) {}  // Placeholder for function-level device reset
}
}
```

Notes:

- The *_CID* field represents the compatible ID for the device and must always be set to “PNP0C51” for a HID SPI compliant DEVICE. This enables the HOST to identify the DEVICE’s class and load the correct software (driver) without any additional software from the device manufacturer.
- *GpioInt* identifies the example of a GPIO based interrupt (and its characteristics) associated with the DEVICE.
- *The _DSM field represents a device specific method that is associated with the HID over SPI specification. Additional Details on _DSM are available in the ACPI specification (Section 9.14.1). The HIDSPI _DSM GUID is* "6E2AC436-0FCF-41AF-A265-B32A220DCFAB".
- Example fields are taken from ACPI enumeration example, including addresses, opcodes and flags. See section for more information.

11.2 DEVICE DESCRIPTOR

The Device Descriptor for the Accelerometer Device is described below.

Byte Offset	Field	VALUE (HEX)	Notes
0	**wDeviceDescLength**	0018	Length of HID Descriptor. Fixed to 24 bytes.
2	**bcdVersion**	0300	Compliant with Version 1.00
4	**wReportDescLength**	00E5	Report Descriptor is 229 Bytes (0x00E5)
6	**wMaxInputLength**	0009	Largest input/feature report is 13 bytes for the feature report, as input report is only 9 bytes.
8	**wMaxOutputLength**	0000	Largest output/feature report is 13 bytes for the feature report, as no output report is present
10	**wMaxFragmentLength**	0010	Input Report of 9 bytes is transferred in single fragment of 9 bytes + 4 bytes header, and 3 bytes padding = 16 bytes
12	**wVendorID**	049F	Vendor ID 0x049F
14	**wProductID**	0101	Device ID 0x0101
16	**wVersionID**	0100	Version 1.00
18	**wFlags**	0000	No flags for this device
20	**RESERVED**	00 00 00 00	RESERVED

11.3 REPORT DESCRIPTOR

The Report Descriptor comprises of sensors part of the HID Sensors Usage Page.

```
0x05, 0x20,                 //Sensor Usage Page
0x09, 0x73,                 //HID_USAGE_SENSOR_TYPE_MOTION_ACCELEROMETER_3D,
0xa1, 0x00,                 //HID_COLLECTION(Physical),
//<FEATURE>
0x05, 0x20,                 //HID_USAGE_PAGE_SENSOR
0x0a, 0x16, 0x03,           //HID_USAGE_SENSOR_PROPERTY_REPORTING_STATE
0x15, 0x00,                 //LOGICAL_MINIMUM (0)
0x26, 0xff, 0x00,           //LOGICAL_MAXIMUM (255)
0x75, 0x08,                 //HID_REPORT_SIZE(8)
0x95, 0x01,                 //HID_REPORT_COUNT(1)
0xb1, 0x02,                 //HID_FEATURE(DATA_VAR_ABS)
0x0a, 0x03, 0x03,           //HID_USAGE_SENSOR_PROPERTY_SENSOR_STATUS
0x15, 0x00,                 //LOGICAL_MINIMUM (0)
0x26, 0xff, 0x00,           //LOGICAL_MAXIMUM (255)
0x75, 0x08,                 //HID_REPORT_SIZE(8)
0x95, 0x01,                 //HID_REPORT_COUNT(1)
0xb1, 0x02,                 //HID_FEATURE(DATA_VAR_ABS)
0x0a, 0x09, 0x03,           //HID_USAGE_SENSOR_PROPERTY_SENSOR_CONNECTION_TYPE
0x15, 0x00,                 //LOGICAL MINIMUM (0)
0x26, 0xff, 0x00,           //LOGICAL_MAXIMUM (255)
0x75, 0x08,                 //HID_REPORT_SIZE(8)
0x95, 0x01,                 //HID_REPORT_COUNT(1)
0xb1, 0x02,                 //HID_FEATURE(DATA_VAR_ABS)
0x0a, 0x0f, 0x03,           //HID_USAGE_SENSOR_PROPERTY_CHANGE_SENSITIVITY_ABS
0x15, 0x00,                 //LOGICAL MINIMUM (0)
0x27, 0xff, 0xff, 0x00, 0x00,//LOGICAL_MAXIMUM (65535)
0x75, 0x10,                 //HID_REPORT_SIZE(16)
0x95, 0x01,                 //HID_REPORT_COUNT(1)
0x65, 0x1a,                 //HID_USAGE_SENSOR_UNITS_G
0x55, 0x02,                 //HID_UNIT_EXPONENT(2)
0xb1, 0x02,                 //HID_FEATURE(DATA_VAR_ABS)
0x0a, 0x0e, 0x03,           //HID_USAGE_SENSOR_PROPERTY_REPORT_INTERVAL
0x15, 0x00,                 //LOGICAL_MINIMUM (0)
```



```
0x27, 0xff, 0xff, 0xff, 0xff,   //LOGICAL_MAXIMUM (4294967295)
0x75, 0x20,                     //HID_REPORT_SIZE(32)
0x95, 0x01,                     //HID_REPORT_COUNT(1)
0x65, 0x19,                     //HID_USAGE_SENSOR_UNITS_MILLISECOND
0x55, 0x00,                     //HID_UNIT_EXPONENT(0)
0xb1, 0x02,                     //HID_FEATURE(DATA_VAR_ABS)
0x0a, 0x52, 0x24,
//HID_USAGE_SENSOR_DATA(HID_USAGE_SENSOR_DATA_MOTION_ACCELERATION,HID_USAGE_SENSOR_DATA_MOD_MA
X)
0x16, 0x01, 0x80,               //LOGICAL_MINIMUM (-32767)
0x26, 0xff, 0x7f,               //LOGICAL_MAXIMUM (32767
0x75, 0x10,                     //HID_REPORT_SIZE(16)
0x95, 0x01,                     //HID_REPORT_COUNT(1)
0x65, 0x1a,                     //HID_USAGE_SENSOR_UNITS_G
0x55, 0x02,                     //HID_UNIT_EXPONENT(2)
0xb1, 0x02,                     //HID_FEATURE(DATA_VAR_ABS)
0x0a, 0x52, 0x34,
//HID_USAGE_SENSOR_DATA(HID_USAGE_SENSOR_DATA_MOTION_ACCELERATION,HID_USAGE_SENSOR_DATA_MOD_MI
N)
0x16, 0x01, 0x80,               //LOGICAL_MINIMUM (-32767)
0x26, 0xff, 0x7f,               //LOGICAL_MAXIMUM (32767)
0x75, 0x10,                     //HID_REPORT_SIZE(16)
0x95, 0x01,                     //HID_REPORT_COUNT(1)
0x65, 0x1a,                     //HID_USAGE_SENSOR_UNITS_G
0x55, 0x02,                     //HID_UNIT_EXPONENT(2)
0xb1, 0x02,                     //HID FEATURE(DATA VAR ABS)
//</FEATURE>
//<INPUT>
0x05, 0x20,                     //HID_USAGE_PAGE_SENSOR
0x0a, 0x01, 0x02,               //HID_USAGE_SENSOR_STATE
0x15, 0x00,                     //LOGICAL_MINIMUM (0)
0x26, 0xff, 0x00,               //LOGICAL_MAXIMUM (255)
0x75, 0x08,                     //HID_REPORT_SIZE(8)
0x95, 0x01,                     //HID_REPORT_COUNT(1)
0x81, 0x02,                     //HID_INPUT(DATA_VAR_ABS)
0x0a, 0x02, 0x02,               //HID_USAGE_SENSOR_EVENT
0x15, 0x00,                     //LOGICAL_MINIMUM (0)
0x26, 0xff, 0x00,               //LOGICAL_MAXIMUM (255)
0x75, 0x08,                     //HID_REPORT_SIZE(8)
0x95, 0x01,                     //HID_REPORT_COUNT(1)
0x81, 0x02,                     //HID_INPUT(DATA_VAR_ABS)
0x0a, 0x53, 0x04,               //HID_USAGE_SENSOR_DATA_MOTION_ACCELERATION_X_AXIS
0x16, 0x01, 0x80,               //LOGICAL_MINIMUM (-32767)
0x26, 0xff, 0x7f,               //LOGICAL_MAXIMUM (32767)
0x75, 0x10,                     //HID_REPORT_SIZE(16)
0x95, 0x01,                     //HID_REPORT_COUNT(1)
0x65, 0x1a,                     //HID_USAGE_SENSOR_UNITS_G
0x55, 0x0e,                     //HID_UNIT_EXPONENT(14)
0x81, 0x02,                     //HID_INPUT(DATA_VAR_ABS)
0x0a, 0x54, 0x04,               //HID_USAGE_SENSOR_DATA_MOTION_ACCELERATION_Y_AXIS
0x16, 0x01, 0x80,               //LOGICAL_MINIMUM (-32767)
0x26, 0xff, 0x7f,               //LOGICAL_MAXIMUM (32767)
0x75, 0x10,                     //HID_REPORT_SIZE(16)
0x95, 0x01,                     //HID_REPORT_COUNT(1)
0x65, 0x1a,                     //HID_USAGE_SENSOR_UNITS_G
0x55, 0x0e,                     //HID_UNIT_EXPONENT(14)
0x81, 0x02,                     //HID_INPUT(DATA_VAR_ABS)
0x0a, 0x55, 0x04,               //HID_USAGE_SENSOR_DATA_MOTION_ACCELERATION_Z_AXIS
0x16, 0x01, 0x80,               //LOGICAL_MINIMUM (-32767)
0x26, 0xff, 0x7f,               //LOGICAL_MAXIMUM (32767)
0x75, 0x10,                     //HID_REPORT_SIZE(16)
0x95, 0x01,                     //HID_REPORT_COUNT(1)
0x65, 0x1a,                     //HID_USAGE_SENSOR_UNITS_G
0x55, 0x0e,                     //HID_UNIT_EXPONENT(14)
0x81, 0x02,                     //HID_INPUT(DATA_VAR_ABS)
//Shake Event Notification
0x0a, 0x51, 0x04,               // USAGE (Data: Motion Intensity)
0x15, 0x00,                     //LOGICAL_MINIMUM (0)
0x25, 0x40,                     //LOGICAL_MAXIMUM (64)
0x75, 0x08,                     //HID_REPORT_SIZE(8)
0x95, 0x01,                     //HID_REPORT_COUNT(1)
0x81, 0x02,                     //HID_INPUT(DATA_VAR_ABS)
//</INPUT>
0xc0                            //HID_END_COLLECTION
```


11.4 REPORTS

The Following table summarizes the report format for each of the three reports for this device.

Input Report [9 Bytes]	Output Report [0 Bytes]	Feature Report [13 Bytes]
Sensor State (1 Byte) **Sensor Event (1Byte)** **Acceleration Axis X (2 Bytes)** **Acceleration Axis Y (2 Bytes)** **Acceleration Axis Z (2 Bytes)** **Data: Motion Intensity (1 Byte)**	N/A	Property: Reporting State (1 Byte) Property: Sensor Reporting State (1 Byte) Property: Connection Type (1Byte) Property: Change Sensitivity Abs (2 Bytes) Property: Report Interval (4 Bytes) Property: Acc Mod Max (2 Byte) Property: Acc Mod Min (2 Byte)