The following document contains information on Cypress products. Although the document is marked with the name "Broadcom", the company that originally developed the specification, Cypress will continue to offer these products to new and existing customers.

CONTINUITY OF SPECIFICATIONS

There is no change to this document as a result of offering the device as a Cypress product. Any changes that have been made are the result of normal document improvements and are noted in the document history page, where supported. Future revisions will occur when appropriate, and changes will be noted in a document history page.

CONTINUITY OF ORDERING PART NUMBERS

Cypress continues to support existing part numbers. To order these products, please use only the Ordering Part Numbers listed in this document.

FOR MORE INFORMATION

Please visit our website at www.cypress.com or contact your local sales office for additional information about Cypress products and services.

OUR CUSTOMERS

Cypress is for true innovators – in companies both large and small.

Our customers are smart, aggressive, out-of-the-box thinkers who design and develop game-changing products that revolutionize their industries or create new industries with products and solutions that nobody ever thought of before.

ABOUT CYPRESS

Founded in 1982, Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, home automation and appliances, consumer electronics and medical products. Cypress's programmable systems-on-chip, general-purpose microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first.

Cypress is committed to providing customers with the best support and engineering resources on the planet enabling innovators and out-of-the-box thinkers to disrupt markets and create new product categories in record time. To learn more, go to www.cypress.com.

Application Note

BCM2073X

Upgrading Human Interface Device Firmware

Example Protocol, Commands, and Scripts

2073X-AN100-R

5300 California Avenue • Irvine, CA 92617 • Phone: 949-926-5000 • Fax: 949-926-5203

July 21, 2014

Revision History

Revision	*Date*	*Change Description*
2073X-AN100-R	07/21/14	Initial release

Broadcom Corporation
5300 California Avenue
Irvine, CA 92617

Table of Contents

List of Tables

About This Document

Purpose and Audience

This document describes the human interface device (HID) layer protocol that the Broadcom® BCM2073X uses to support Bluetooth host-to-HID over-the-air upgrades. In addition, the document provides some sample commands and scripts.

This document is primarily intended for software developers writing Bluetooth host-device applications that will be used to communicate with BCM2073X-based HIDs.

Acronyms and Abbreviations

In most cases, acronyms and abbreviations are defined on first use.

For a comprehensive list of acronyms and other terms used in Broadcom documents, go to: http://www.broadcom.com/press/glossary.php.

Document Conventions

The following conventions may be used in this document:

Convention	*Description*
Bold	User input and actions: for example, type **exit**, click **OK,** press **Alt+C**
`Monospace`	Code: `#include <iostream>` HTML: `<td rowspan = 3>` Command line commands and parameters: `wl [-l] <command>`
< >	Placeholders for *required* elements: enter your <username> or `wl <command>`
[ ]	Indicates *optional* command-line parameters: `wl [-l]` Indicates bit and byte ranges (inclusive): [0:3] or [7:0]

References

The references in this section may be used in conjunction with this document.

Note: Broadcom provides customer access to technical documentation and software through its Customer Support Portal (CSP) and Downloads and Support site (see Technical Support).

For Broadcom documents, replace the “xx” in the document number with the largest number available in the repository to ensure that you have the most current version of the document.

Document (or Item) Name	Number	Source
Broadcom Items		
[1] Single-Chip Bluetooth Transceiver for Wireless Input Devices, Data Sheet	20730-DS1xx-R	Broadcom CSP
[2] Single-Chip Bluetooth Transceiver for Wireless Input Devices, Data Sheet	20733-DS0xx-R	Broadcom CSP
[3] EEPROM Format Information and Firmware Upgrading, Application Note	20730_20733-AN3xx-R	Broadcom CSP
[4] Application Development Kit (ADK)	–	Broadcom representative
[5] BlueTool™ Software	–	Broadcom representative
Other Items		
[6] Human Interface Device Profile 1.1	HID_SPEC_V11.pdf	www.bluetooth.org

Technical Support

Broadcom provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates through its customer support portal (https://support.broadcom.com). For a CSP account, contact your Sales or Engineering support representative.

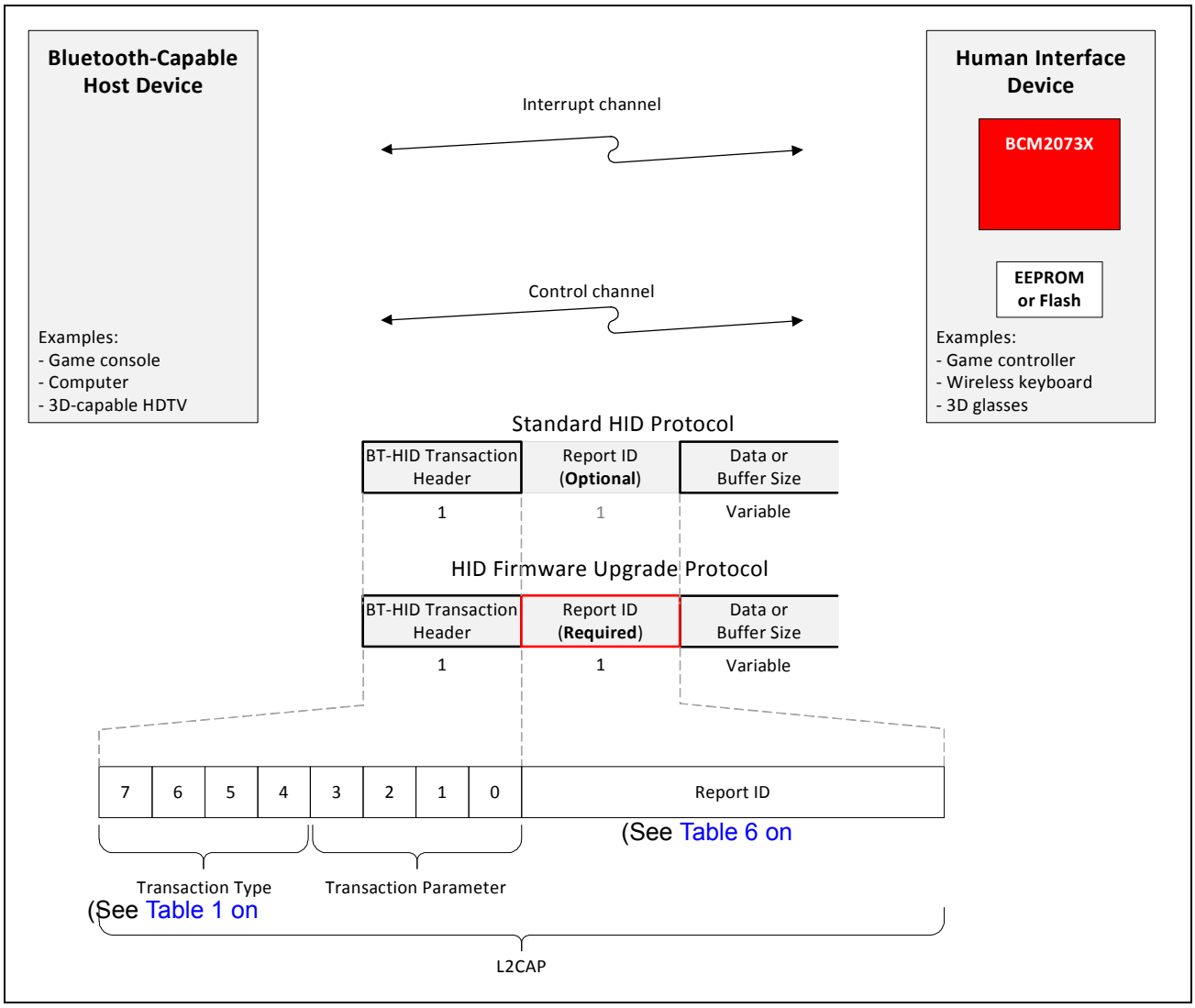
In addition, Broadcom provides other product support through its Downloads and Support site (http://www.broadcom.com/support/).

Introduction

A human interface device (HID) provides a means for human data input and output to and from a remote host.

Figure 1 shows an application diagram where a BCM2073X is used in a HID.

Figure 1: BCM2073X Application Diagram



The standard Bluetooth HID protocol has an optional report ID byte that follows the transaction header byte when it is used.

The extensions to the standard HID protocol in this document require using the report ID byte.

Standard HID Protocol Transaction Types

As shown in Figure 1 on page 7, all interrupt/control channel messages between a host and a HID begin with a BT-HID transaction header. The transaction header has two 4-bit fields, the transaction type and an associated transaction parameter.

Table 1 provides the complete set of transaction types, and the hexadecimal value and transaction length of each type.

Table 1: Transaction Type Information

Transaction Type	*BT-HID Transaction Header Value*	*Transaction Length (Bytes)*[a]	*Description*
HANDSHAKE	0x0	1	A HID response to all set requests and all get requests where an error is detected.
HID_CONTROL	0x1	1	A control used to request a state change in the Bluetooth HID.
Reserved	0x2–0x3	–	–
GET_REPORT	0x4	1–4	A host request for a report from the HID.
SET_REPORT	0x5	1 + report-data payload	A host report to the HID.
GET_PROTOCOL	0x6	1	A host request for the current protocol.
SET_PROTOCOL	0x7	1	A host set of the HID protocol.
GET_IDLE	0x8	1	A host request for the current idle state.
SET_IDLE	0x9	2	A host set of the idle state.
DATA	0xA	1 + report-data payload	The first (and possibly only) data payload of report-data payloads.
DATC	0xB	1 + continuation of report-data	Subsequent payloads associated with an overall report-data payload that exceeds a negotiated MTU.
Reserved	0xC–0xF	–	–

a. All transaction lengths include the 1-byte BT-HID transaction header.

Note: Of the transaction types in Table 1, GET_REPORT, SET_REPORT, DATA, and HANDSHAKE are the only types that pertain to the content in the remainder of this document.

Transaction Types Supporting a Firmware Upgrade

Note: The transaction format information in this section is from a Bluetooth specification. For complete transaction format information, refer to *Human Interface Device Profile 1.1* (available from www.bluetooth.org).

GET_REPORT Transaction Type

This transaction type is used by the host to request a report from the HID. If the request is received without errors, then the HID responds by returning a payload of transaction type DATA on the control channel that contained the requested report. If there are errors, then the HID returns a HANDSHAKE with an error message.

Table 2 describes the byte and bit fields in a GET_REPORT transaction.

Table 2: GET_REPORT Transaction Type Description

Field	*Size (Bytes)*	*Bits*	*Description*
Request	1		Request characteristics.
		7:4	Transaction type. Set to 4 to indicate GET_REPORT.
		3	Size 0: The host has allocated a buffer equal to the size of the report. 1: A 2-byte BufferSize field follows the Report ID. This field indicates the size of the buffer allocated by the host. HIDs must limit the returned payload size to BufferSize. **Note:** The BufferSize must be increased by 1 byte for Boot mode reports in order to include the Report ID imposed by BT-HID.
		2	Reserved. Set to 0.
		1:0	Report type. 0: Reserved 1: Input 2: Output 3: Feature
Report ID	1	7:0	Optional field for standard HID protocol. Required for a HID FW upgrade. Report ID of the requested report (see Table 6 on page 12).
BufferSize	2	15:0	Optional field. Maximum number of bytes to transfer during the data phase. This field does not exist if the Size field (bit 3 in the request byte) is 0. BufferSize is little endian.

SET_REPORT Transaction Type

The Bluetooth host uses this transaction type to send a report to a HID. A single report follows either the request or the report ID if a report ID is present. Only one report can be sent per SET_REPORT transaction.

Table 3 describes the byte and bit fields in a SET_REPORT transaction.

Table 3: SET_REPORT Transaction Type Description

Field	*Size (Bytes)*	*Bits*	*Description*
Request	1		Request characteristics.
		7:4	Transaction type. Set to 5 to indicate SET_REPORT.
		3:2	Reserved
		1:0	Report type. 0: Reserved. 1: Input. 2: Output. 3: Feature.
Report ID	1	7:0	Optional field for the standard protocol. Required for a HID FW upgrade. Report ID of the payload (see Table 6 on page 12).
Report Data Payload	n	–	Report data. See "SETUP_READ" on page 16 for the payload used to set up a data read from the HID during a firmware upgrade.

DATA Transaction Type

The Bluetooth HID uses this transaction type to send a report to the Bluetooth host.

Table 3 describes the byte and bit fields in a DATA transaction.

Table 4: DATA Transaction Type Description

Field	*Size (Bytes)*	*Bits*	*Description*
Request	1		Request characteristics.
		7:4	Transaction type. Set to 10 (0xA) to indicate a DATA transaction.
		3:2	Reserved (0)
		1:0	Report type. 0: Other. 1: Input. 2: Output. 3: Feature. **Note:** The report type returned from the HID during a firmware upgrade is 0.
Report ID	1	7:0	Optional field for the standard protocol. Required for a HID FW upgrade. Report ID of the payload (see Table 6 on page 12).
Report Data Payload	n	–	Report data. See "READ" on page 16 for the payload used while reading data from the HID during a firmware upgrade.

HID Firmware Upgrade Protocol

The protocol in this section describes a HID layer upgrade method for upgrading HID firmware (FW).

Note: The host must open one control channel and one interrupt channel before starting a HID firmware upgrade.

In order to support a HID FW upgrade, some previously unused report IDs are assigned. Table 5 shows the assigned report IDs and those that are unused.

Table 5: Bluetooth Report IDs from HID Profile Specification Version 1.0

Report ID	*Device*	*Report Size (Bytes)*
0	Reserved	Not applicable
1	Keyboard	9
2	Mouse	4
3–255 (0x03–0xFF)	Reserved (Unused)	Not applicable

The Broadcom approach to supporting a HID FW upgrade uses some of the many unused report IDs.

Report IDs and Associated Commands

Table 6 describes extended report command types that can be assigned to six of the previously unused report IDs.

Table 6: Report IDs to Support a HID FW Upgrade

Report ID	*Report Command Type*	*Report Size (Bytes)*[a]	*Transaction Type*	*Description*
0x70	ENABLE_FWU	1	SET_REPORT	Command to enable a FW upgrade.
0x71	SETUP_READ	8	SET_REPORT	Command to set the read address and length for a subsequent READ transaction.
0x72	READ	1 + n	GET_REPORT	Command to read data from the HID.
0x73	ERASE	8	SET_REPORT	Command to erase HID memory.
0x74	WRITE	8 + n	SET_REPORT	Command to write to HID memory.
0x75	LAUNCH	8 + n	SET_REPORT	Command to jump execution to a specific HID memory location.

a. The variable n represents the number of payload data bytes in the report. It does not include the report ID, address, length, and checksum bytes. The overall size reported in this column includes all bytes except the 1-byte transaction header.

For more information and software examples for each report command in Table 6, see “HID Firmware Upgrade Commands” on page 14 and “HID Firmware Upgrade Sample Scripts” on page 19.

Typical HID Firmware-Upgrade Packet Structure

Most of the report commands (see Table 6 on page 12) use the following basic packet structure.

```
struct HIDFWUReportHeader
{
    UINT8  reportID;  // Report ID (1 byte)
    UINT32 address;   // Address (4 bytes, little endian)
    UINT16 length;    // Length in bytes (2 bytes, little endian
    UINT8  data[];    // Data bytes (if present, their count must match "length")
    UINT8  checksum;  // Checksum (1 byte)
};
```

Table 7 describes the fields of the basic HID firmware-upgrade report header.

Table 7: Basic HID Firmware-Upgrade Report Header

Parameter	*Size (Bytes)*	*Description*
`reportID`	1	The Report ID of the packet. See Table 6 on page 12 for a complete list of report IDs.
`address`	4	The start address of a READ, WRITE, ERASE, or LAUNCH command.
`length`	2	The number of bytes to be read, written, or erased.
`checksum`	1	The 2's complement checksum of `HIDFWUReportHeader`, ignoring carry bits. The checksum calculation starts with the reportID. See "Calculating a Checksum" on page 23 for sample Perl code.

HID Firmware Upgrade Commands

The packet formats and descriptions for the following commands are covered in this section:

- ENABLE_FWU
- SETUP_READ
- READ
- ERASE
- WRITE
- LAUNCH

For reference Perl scripts of the commands in this section, see “HID Firmware Upgrade Sample Scripts” on page 19.

Note: BCM2073X firmware will verify the total transaction sizes of all commands. In the verification, all transaction bytes will be accounted for, including the transaction header. If the size of a command does not match the expected size, then a HANDSHAKE error with error code ERR_INVALID_PARAMETER will be returned. This same error code will be returned in a HANDSHAKE transaction when the HID detects any error.

ENABLE_FWU

This command is used to enable a HID firmware upgrade. Upon receiving this command, a watchdog timer will be enabled in the HID.

The ENABLE_FWU command must be sent before sending any ERASE, WRITE, and LAUNCH commands.

Note: The firmware upgrade process cannot be disabled after sending the ENABLE_FWU command. For a successful firmware upgrade, the HID will be restarted before watchdog expiration. For an unsuccessful upgrade, the HID will be restarted when the watchdog timer expires.

The following packet format is used for this command:

```
struct HIDFWUReportHeader
{
    UINT8  reportID;// 0x70, but changeable.
    UINT8  checksum;
};
```

Report Data Payload (The checksum covers the reportID.)

SETUP_READ and READ—Reading Memory

The SETUP_READ and READ commands are used to read memory. The SETUP_READ command can be sent at anytime, but must be sent at least once before issuing any READ commands.

To read HID memory, the Bluetooth host must do the following:

1. Send a SETUP_READ command, which is a SET_REPORT transaction, to indicate the start *address* and *length* in bytes of the subsequent read.
2. Send a READ command, which is a GET_REPORT transaction, to read the data block defined by the SETUP_READ command.

Note: Additional READ commands can be sent without first sending another SETUP_READ command if reading is to take place where the last READ command left off. In other words, HID firmware will advance the start address to the *address* + *length* defined in the SETUP_READ command.

Both of these read commands will work, regardless of the storage type from which data is being read. So the commands apply whether the storage type is RAM, parallel flash, serial flash, or EEPROM. However, an indirect memory map must be used for memory peripherals, such as EEPROM and flash, that are not in the addressable space of the BCM2073X.

Use the following enumeration to indirectly map EEPROM and flash memory:

```
enum
    {
        INDIRECT_MEM_MAP_MASK   = 0xFF000000,

        /// indirect memory map for EEPROM Read/Write/erase access
        INDIRECT_MEM_MAP_EEPROM = 0xFF000000,

        /// indirect memory map for parallel flash (write/erase access),
        //// read is in CPU addressable space
        INDIRECT_MEM_MAP_PF     = 0xFC000000,

        /// indirect memory map for serial flash Read/Write/Erase access
        INDIRECT_MEM_MAP_SF     = 0xF8000000,

    };
```

SETUP_READ

The SETUP_READ command sets the start address and length (in bytes) of a subsequent or multiple subsequent READ commands. This command can be issued by the host at anytime.

The following packet format is used with this command (using the SET_REPORT transaction):

```
struct HIDFWUReportHeader
{
    UINT8  reportID;  // report=0x71 , changeable.
    UINT32 address;
    UINT16 length;
    UINT8  checksum;
};
```

Report Data Payload (The checksum covers the payload and the reportID.)

For more information on the parameters in the `HIDFWUReportHeader` header, see Table 7: “Basic HID Firmware-Upgrade Report Header,” on page 13.

For a sample script of the SETUP_READ command, see “SetReport” on page 20.

READ

The READ command is used to read data from HID memory (RAM, flash, or EEPROM). Multiple READ commands can be issued after a single SETUP_READ command is sent to the HID. Each READ command sent by the host will result in a block of data being returned by the HID. The length of each block of data returned is the `length` parameter sent in the most recently received SETUP_READ command.

The following packet format is used by the host to send a read command (using the GET_REPORT transaction) to the HID:

```
struct HIDFWUReportHeader
{
    UINT8  reportID;//DEFAULT_ID_READ
    UINT8  checksum;
};
```

Report Data Payload (The checksum covers the reportID.)

The following packet format is used by the HID to send data packets (using DATA transactions) back to the host:

```
struct HIDFWUReportHeader
{
    UINT8  reportID;     //DEFAULT_ID_READ
    UINT32 address;
    UINT16 length;
    UINT8  data[length];
    UINT8  checksum;
};
```

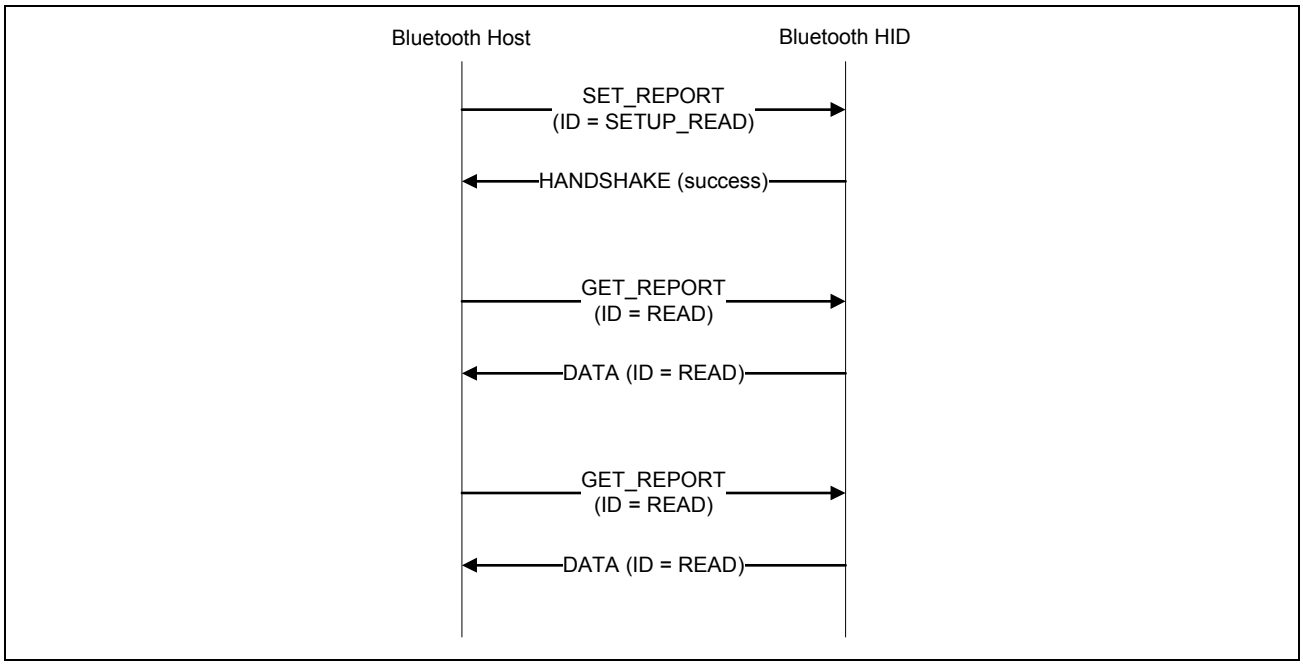
Report Data Payload (The checksum covers the payload and the reportID.)

For more information on the parameters in the `HIDFWUReportHeader` header, see Table 7: “Basic HID Firmware-Upgrade Report Header,” on page 13.

For a sample script of the READ command, see “GetReport” on page 21.

Figure 2 shows an example read sequence using some of the transaction types provided in Table 1 on page 8.

Figure 2: Read Sequence



ERASE

The erase command is used to erase memory. The ENABLE_FWU command needs to be sent before erasing memory.

Note: The ERASE command is only valid for serial flash. It does not apply to EEPROM, which does not need to be erased before overwriting. The ERASE command also does not apply to erasing parallel flash because the BCM2073X only supports reading from parallel flash.

The following packet format is used by the host to send this command:

```
struct HIDFWUReportHeader
{
    UINT8  reportID;     //DEFAULT_ID_ERASE
    UINT32 address;
    UINT16 length;
    UINT8  checksum;
};
```

Report Data Payload (The checksum covers the payload and the reportID.)

For more information on the parameters in the `HIDFWUReportHeader` header, see Table 7: “Basic HID Firmware-Upgrade Report Header,” on page 13.

For a sample script of the ERASE command, see “Erasing HID Memory” on page 22.

WRITE

The write command is used to write to HID memory. The ENABLE_FWU command must be sent before writing to memory.

The following packet format is used by the host to send this command:

```
struct HIDFWUReportHeader
{
    UINT8  reportID;     //DEFAULT_ID_WRITE
    UINT32 address;
    UINT16 length;
    UINT8  data[length];
    UINT8  checksum;
};
```

Report Data Payload (The checksum covers the payload and the reportID.)

For more information on the parameters in the `HIDFWUReportHeader` header, see Table 7: “Basic HID Firmware-Upgrade Report Header,” on page 13.

For a sample script of the WRITE command, see “Writing to HID Memory” on page 22.

LAUNCH

The LAUNCH command is used to indicate the address from which firmware will execute.

The following packet format is used by the host to send this command:

```
struct HIDFWUReportHeader
{
    UINT8  reportID;     //DEFAULT_ID_LAUNCH
    UINT32 address;
    UINT16 length;
    UINT8  data[length];
    UINT8  checksum;
};
```

Report Data Payload (The checksum covers the payload and the reportID.)

The following parameters in the above `HIDFWUReportHeader` header differ from the parameters defined in Table 7: “Basic HID Firmware-Upgrade Report Header,” on page 13:

- address: The entry address to which firmware jumps.

  If this value is 0x0000, then the host should assume that the HID will reboot.
- length: This parameter is passed to the above entry address as a parameter.
- data[]: These bytes will pass the entry address as a parameter.

HID Firmware Upgrade Sample Scripts

This section mostly contains reference scripts for developers of host software that is used to upgrade the firmware in a BCM2073X-based HID. All scripts are written in Perl.

Scripts to support the following actions are included in this section:

- Enabling a HID for a Firmware Upgrade
- Reading HID Memory.
- Erasing HID Memory
- Launching Execution at a Defined HID Memory Address
- Calculating a Checksum

The following scripts, which support the above list of actions, are provided in this section:

- EnableOtafu
- ReadMem
- SetReport
- GetReport
- RxReport
- PrintReport
- EraseMem
- WriteMem
- Launch
- Checksum

Enabling a HID for a Firmware Upgrade

The following Perl subroutine is an example script for enabling a HID for a firmware upgrade:

```
################################################################
# Enable OTAFU (send the magic 0x70 report)
################################################################
sub EnableOtafu
{
    print "Enable OTAFU \n";
    my %command = ('Transaction Type' => 'SET_REPORT',
                   'Report Type'      => 'Feature',
                   'Report ID'        => $ID_ENABLE_OTAFU,  #0x70
                   'Data'             => "");
    BTSP::SendHIDCommand( $dutaddr, \%command );
    BTSP::WaitForSpecificEvent( $hidht, "Number Of Completed Packets" );
}
```

Reading HID Memory

The following Perl subroutine is an example script for reading memory. This Perl subroutine calls the SetReport and GetReport subroutines.

```
################################################################
# Read memory (addr, len) -- READ SETUP set-rep then READ get-rep
################################################################
sub ReadMem
{
    my ($addr, $len) = @_;

    SetReport($ID_SETUP_READ, $addr, $len, [ ] );
    #setup read start address and the length

    my $report = GetReport($ID_READ);

    return $report;
}
```

SetReport

This Perl subroutine is used to issue the SETUP_READ command.

```
################################################################
# Send a set report( report-id, address, len, data)
################################################################
sub SetReport
{
    my ($repid, $addr, $len, $bytes) = @_;
    my $srepid = sprintf("%02x", $repid);
    my $saddr = sprintf("%02x %02x %02x %02x", $addr&0xFF, ($addr>>8)&0xFF, ($addr>>16)&0xFF,
($addr>>24)&0xFF);
    my $sbytes = "";

    if ( scalar(@{$bytes}) > 0 ) {
        $sbytes = BTSP::ByteArrayToDataString(@{$bytes}) . " ";
    }

    my $slen = sprintf("%02x %02x", $len&0xFF, $len>>8 );
    my $checksum = Checksum($repid, $addr, $len, $bytes);
    my $schecksum = sprintf("%02x", $checksum);
    my $sreportbytes = "$saddr $slen $sbytes" . "$schecksum";

    my %command = ('Transaction Type' => 'SET_REPORT',
                   'Report Type'      => 'Feature',
                   'Report ID'        => $repid,
                   'Data'             => $sreportbytes);

    BTSP::SendHIDCommand( $dutaddr, \%command );

    RxData(); #
}
```

GetReport

This Perl subroutine is used to read HID memory. This section also includes the RxReport subroutine, which the GetReport subroutine calls, and the PrintReport subroutine.

```
################################################################
# Send a GET-REPORT, return the data received
################################################################
sub GetReport
{
    my ($reptype) = @_;

    my %command = ('Transaction Type' => 'GET_REPORT',
                   'Report Type'      => 'Feature',
                   'Report ID'        => $reptype,
                   'Size'             => 'Equal to the size of the report');

    BTSP::SendHIDCommand( $dutaddr, \%command );

    return RxReport();
}
```

RxReport

This Perl subroutine is used to receive and decode a firmware upgrade report received from the HID.

```
################################################################
# Receive an OTAFU report, decode it into returned hash
################################################################
sub RxReport
{
    my @report  = RxData();

    my @payloadbytes = @report[7..$#report-1];
    my $payloadref = \@payloadbytes;
    my $ret;
    my $sumpresent = $report[$#report];

    $ret->{'id'} = $report[0];

    if (length @report > 1)
    {
    $ret->{'address' }  = ($report[1] | ($report[2]<<8) | ($report[3]<<16) | ($report[4]<<24));
    $ret->{'len'     }  = ($report[5] | ($report[6]<<8));
    $ret->{'data'    }  = $payloadref;
    $ret->{'checksum'}  = $report[$#report];

    my $expected = Checksum($ret->{'id'},$ret->{'address'},$ret->{'len'},$payloadref);
    if ($sumpresent != $expected)
        {
         print "Report is ", %{$ret}, "\n";
         die sprintf("Checksums don't match: Expected 0x%02x, Got 0x%02x", $expected, $sumpresent);
        }
    }
    return $ret;
}
```

PrintReport

This Perl subroutine is used to print a report of the hash returned by the RxReport and GetReport subroutines.

```
################################################################
# Print a report (the hash returned by RxReport/GetReport)
################################################################
sub PrintReport
{
    my ($report) = @_;
    my $bytes = $report->{'data'};

    printf("HID FWU REPORT:\n" .
           " id       = 0x%02x\n" .
           " address  = 0x%08x\n" .
           " len      = 0x%04x\n" .
           " data     = %s\n" .
           " checksum = 0x%02x\n",
           $report->{'id'},
           $report->{'address'},
           $report->{'len'},
           BTSP::ByteArrayToDataString(@{$bytes}),
           $report->{'checksum'});
}
```

Erasing HID Memory

The following Perl script is used to erase memory.

```
################################################################
# Erase memory (address, len)
################################################################
sub EraseMem
{
    my ($addr, $len) = @_;
    SetReport($ID_ERASE, $addr, $len, []);
    printf "\nErase 0x%x \n",$addr;
}
```

Writing to HID Memory

The following Perl script is used to write to memory.

```
################################################################
# Write memory (addr, bytes)
################################################################
sub WriteMem
{
    my ($addr, $bytes) = @_;
    printf "WriteMem: 0x%x, %d bytes ...\n", $addr, scalar @{$bytes};
    SetReport($ID_WRITE, $addr, scalar @{$bytes}, $bytes);
}
```

Launching Execution at a Defined HID Memory Address

The following Perl script is used to send a launch command to the HID.

```
#################################################################
# Launch(address)
#################################################################
sub Launch
{
    my ($addr) = @_;

    SetReport($ID_LAUNCH, $addr, 0, []);
}
```

Calculating a Checksum

The following Perl subroutine is used to compute a checksum.

```
#################################################################
# Compute and return a checksum
#################################################################
sub Checksum
{
    my ($id, $addr, $len, $bytes) = @_;
    my $sum = $id;

    $sum += $addr & 0xFF;
    $sum += ($addr>>8) & 0xFF;
    $sum += ($addr>>16) & 0xFF;
    $sum += ($addr>>24) & 0xFF;

    $sum += $len & 0xFF;
    $sum += ($len>>8) & 0xFF;

    my $i;
    for($i=0; $i <= $#{$bytes}; $i++)
    {
        $sum += $bytes->[$i];
    }

    return (-$sum)&0xFF; # 1-byte 2's complement of all the bytes
}
```

Programming an Image

The following Perl subroutine is used to program (burn) an image to a HID device.

```
################################################################
# BurnImage(addr): Burn a config at the DS address passed
################################################################
sub BurnImage
{
    my ($DSaddr) = @_;
    my $line;
    my $base = 0; #$EEPROM_base_addr; # + $DSaddr;


    #printf "\nBurn image= %s ds=%x $base=%x",$burnimg,$DSaddr,$base ;

    # Parse the intel hex, writing each record to the target
    open(IHEXIN , "<$burnimg") || die "Can't open $burnimg: $!";
    LINE: while(<IHEXIN>)
    {
        /^:(..)         # $1: Record length
           (....)       # $2: Load Offset
           (..)         # $3: Record Type
           (.*)         # $4: Data (maybe empty)
           (..)         # $5: Checksum
           \s*$/x || die "Malformed line: $_";

        my $reclen  = hex($1);
        my $offset  = hex($2);
        my $rectype = hex($3);
        my $data    = $4;
        my $sum     = hex($5);

        #printf "\n offset=%x, data=%s",$offset,$data;
        my @bytes = HexStringToBytes($data);
        my $expected = Checksum($rectype,$offset,$reclen,\@bytes);
        if ($expected != $sum)
        {
            die
                sprintf("Invalid checksum: Got 0x%02X, expected 0x%02X in line\n%s",
                        $sum, $expected, $_);
        }

        if ($rectype == 0x01)     # End-of-ffile record
        {
            last LINE;
        }
        elsif ($rectype == 0x04)  # Extended linear address record
        {
            $base = hex($data);
            $base <<=16;    # upper 16 bits
           # if ($base == 0)  # we mean EEPROM
           # {
           #     $base = 0xFF000000;
            printf "\n record type =4 change base to %x\n",$base;
           # }
        }
```

```
        elsif ($rectype == 0x00)  # Data record
        {

            #WriteMem( $base | $offset , \@bytes);
            WriteMem( $base + $offset-$ds_offset_GenBurnImage +$DSaddr  , \@bytes);
            printf ".";

        }
        else # We don't support other record types
        {
            die "Unexpected record type: $rectype, in line\n$_";
        }
    }
    close(IHEXIN);
}
```

Broadcom Corporation
5300 California Avenue
Irvine, CA 92617

Phone: 949-926-5000
Fax: 949-926-5203
E-mail: info@broadcom.com
Web: www.broadcom.com

2073X-AN100-R July 21, 2014