AN214820

EEPROM Format Information and Firmware Upgrading

Associated Part Family: CYW20730/CYW20733

This document describes the memory structure of external EEPROM devices to be used in conjunction with Cypress CYW20733 and CYW20733 devices. It also provides sample Perl and C code for developing host software that updates human interface device (HID) firmware over-the-air.

The document is intended for those writing host application software to update HID firmware over-the-air.

Contents

1 Introduction

This document describes the memory structure of external EEPROM devices to be used in conjunction with Broadcom® CYW20733 and CYW20733 devices. It also provides sample Perl and C code for developing host software that updates human interface device (HID) firmware over-the-air.

The document is intended for those writing host application software to update HID firmware over-the-air.

1.1 Scope

Although the CYW20733 and BCM20733 devices also support external serial flash, this document only applies to external EEPROM devices.

This document does not supply complete information for updating HID firmware over-the-air. The underlying details of the HID-layer protocol are not supplied. For information on the HID-layer protocol and the Perl subroutines used, but not defined, in this document, refer to *Upgrading Human Interface Device Firmware —Example Protocol, Commands, and Scripts* (Broadcom application note 20730_20733-AN4xx-R).

1.2 Cypress Part Numbering Scheme

Cypress is converting the acquired IoT part numbers from Cypress to the Cypress part numbering scheme. Due to this conversion, there is no change in form, fit, or function as a result of offering the device with Cypress part number marking. The table provides Cypress ordering part number that matches an existing IoT part number

Table 1. Mapping Table for Part Number between Broadcom and Cypress

Broadcom Part Number	Cypress Part Number
BCM20730	CYW20730
BCM20733	CYW20733

2 IoT Resources

Cypress provides a wealth of data at http://www.cypress.com/internet-things-iot to help you to select the right IoT device for your design, and quickly and effectively integrate the device into your design. Cypress provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates. Customers can acquire technical documentation and software from the Cypress Support Community website (http://community.cypress.com/).

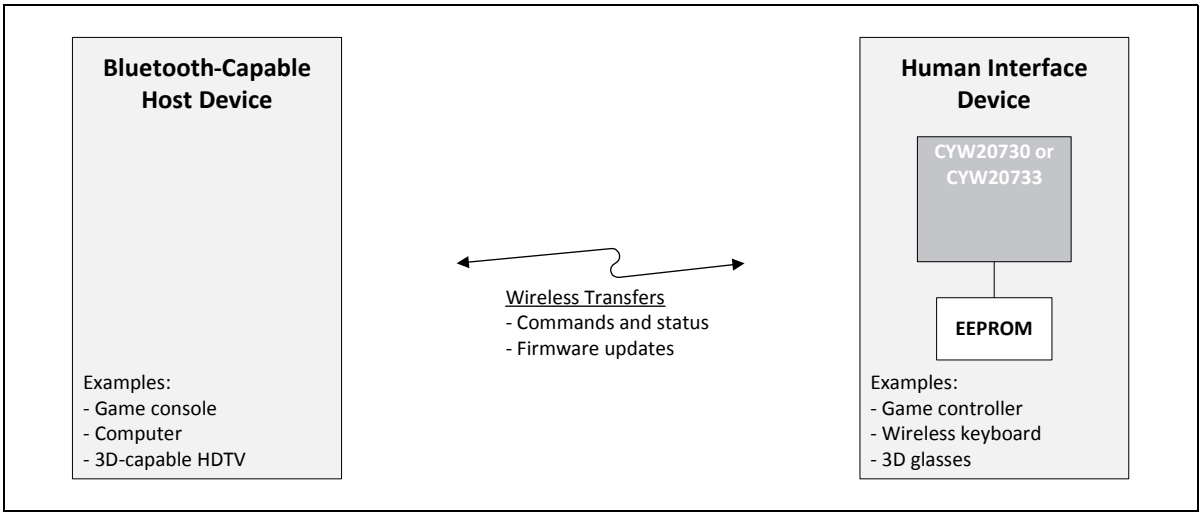
2.1 Acronyms and Abbreviations

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 www.cypress.com/glossary

1 Introduction

Figure 1 shows a simple application diagram where a CYW20730 or CYW20733 is used in a human interface device (HID).

Figure 1. CYW20730 and CYW20733 Application Diagram



For HID developers who choose to support dynamic over-the-air (OTA) configuration and software-application updating, Broadcom provides an interface to external EEPROM and details of how the EEPROM must be structured to work properly with the CYW20730 and CYW20733 devices.

The external EEPROM is used for application code, configuration data, software patches, pairing information, BD_ADDR storage, the operating baud rate, file system information, and more.

The interface between the CYW20730 or CYW20733 device and external EEPROM uses Broadcom Serial Control (BSC), which in this case is mostly compatible with a Philips® I^2C master/slave interface. The CYW20730 (or CYW20733) is the BSC master and the external EEPROM is the BSC slave. For this BSC interface, the CYW20730 and CYW20733 do not support master arbitration, so multiple I^2C masters cannot contend for the bus.

The CYW20730 and CYW20733 provide native support for the following EEPROM devices: Microchip® 24LC128, Microchip 24AA128, and STMicroelectronics® M24128-BR.

2 EEPROM Structure

CYW20730 and CYW20733 device software is setup to work with EEPROM that is divided into the following three sections: a static section (SS), a dynamic section (DS), and a volatile section (VS).

2.1 EEPROM Static Section

2.1.1 Static Section Description and Example

All data in the static section uses the type-length-value (TLV) paradigm. The following data is included in the static section:

- Intermediate frequency (IF) phase-locked loop (PLL) main oscillator settings

This value is usually derived from a crystal frequency of 24 MHz.

- Radio frequency (RF) placeholder setting for RF tuning
- Bluetooth device address (BD_ADDR)
- EEPROM dynamic section location
- EEPROM volatile section location

Figure 2 shows a memory configuration for a typical static section.

Figure 2. Typical Static Section Example

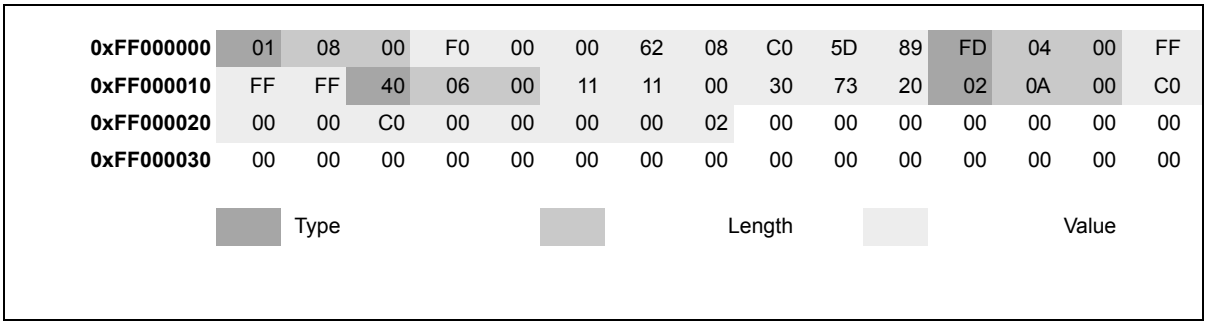


Table 1 shows a breakdown of the type, length, and value information for the example in Figure 2.

Table 1. Typical Static Section Memory Dump Defined

Parameter or Parameters	Type	Length (Little Endian)	Value	
			Data	Endianess
IF_PLL	0x01	0x08 00 (8 bytes of data)	0xF0 00 00 62 08 C0 5D 89	N/A
RF_PLL	0xFD	0x04 00 (4 bytes of data)	0xFF FF FF FF	N/A
BD_ADDR	0x40	0x06 00 (6 bytes of data)	0x11 11 00 30 73 20	Little
DS and VS locations, and VS length	0x02	0x0A 00 (10 bytes of data)	DS: 0xC0 02 00 00 VS: 0xC0 00 00 00 VS length: 0x00 02	Little Little Little

2.1.2 Bootloader Static-Section Searching

Device (CYW20730 and CYW20733) bootloader software searches the following EEPROM memory locations in the order shown for the static section:

0x0000
0x0100
0x0200
0x0400
0x0800
0x1000
0x2000

2.2 EEPROM Dynamic Section

The dynamic section is primarily for configuration data, application software, and software patches.

The location of the dynamic section is determined by TLV-defined data in the static section.

2.3 EEPROM Volatile Section

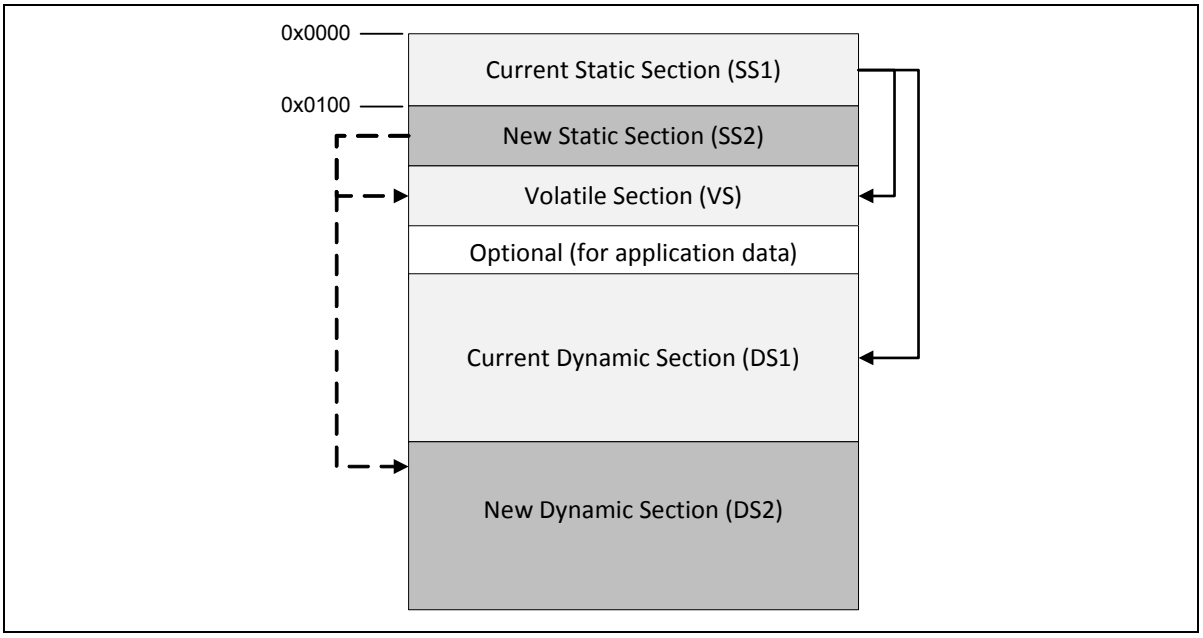
The volatile section is used to store paired host link-key information. A firmware upgrade should not change this section.

2.4 Typical EEPROM Content Structure

The structure and content of EEPROM memory is mostly static over time. Structure and/or content only changes for two reasons. The content of some EEPROM memory can change to update application data and the structure and content will change during and as a result of a firmware upgrade.

Figure 3 shows a typical EEPROM content structure during a firmware upgrade.

Figure 3. Typical EEPROM Structure During a Firmware Upgrade



In Figure 3, the current static section (SS1) starts at address 0x0000. Some content within SS1 points to the locations of the current dynamic section (DS1) and the volatile section (VS). For an example of static section memory content, see Figure 2 on page 4 and Table 1 on page 4. SS1, DS1, and VS represent the key EEPROM memory sections before a firmware upgrade.

After a firmware upgrade, SS2, DS2, and VS represent the key EEPROM memory sections.

3 Updating Firmware Over-the-Air to HID EEPROM

Host device application software initiates and performs over-the-air (OTA) firmware upgrades to EEPROM attached to CYW20730- and CYW20733-based HIDs. The host in this context is the device that communicates wirelessly to the HIDs (see Figure 1 on page 3).

The following steps comprise the high-level OTA upgrade sequence (see Figure 3 on page 5 for the SS1, SS2, DS1, DS2, and VS references):

1. Determine the location of the current static section (SS1) (see Bootloader Static-Section Searching on page 4).

Note: If a static section is not found, bootloader software will loop waiting for a firmware download via UART. There must be at least one valid SS for the device to boot properly.

2. Determine the location and length of the current dynamic section (DS1) (see Figure 2 on page 4 and Table 1 on page 4 for a typical example).
3. Determine the location and length of the volatile section (VS).
4. Find two EEPROM memory blocks, one for each of the new static and dynamic sections. Ensure that neither new block overlaps with the current static, dynamic, and volatile sections.

Note: For selecting the new static section location, opt for sections that will be found as quickly as possible by the bootloader (for example, 0x0 and 0x100). See Bootloader Static-Section Searching on page 4.

5. Download new dynamic section data to the dynamic section block (DS2) chosen in Step 4.
6. Copy the current static section (SS1) to the new static section block (SS2) chosen in Step 4.
7. Set the first byte of SS2 to 0 to make SS2 invalid.
8. Update the new static section (SS2) with the location of the new dynamic section.
9. Verify the new static section (SS2) is correct.
10. Set the first byte of SS2 to 1 to make the new static section (SS2) the valid static section.

11. Erase the old static section (SS1), but only if the Step 9 verification passed.

4 Sample Perl Script for Updating Firmware to HID EEPROM

This script is provided only as a reference for those using Perl to update HID EEPROM firmware over-the-air.

Note: For information on Perl subroutines used, but not defined, in this section, such as `EnableOtafu, WriteMem`, and Launch, refer to *Upgrading Human Interface Device Firmware —Example Protocol, Commands, and Scripts* (Broadcom application note 20730_20733-AN4xx-R).

Note: Step numbers provided in some of the Perl comments are not intended to match up with the step numbers in Updating Firmware Over-the-Air to HID EEPROM on page 5.

```
#!/usr/bin/perl

################################################################

#

# Perl script file for OTA firmware for 20730/20733 module only

# Need to double check the DS1/DS2 and SS1/SS2 for HID device

# before use this perl script file

################################################################

use warnings;

use strict;

use BTSP;

use Getopt::Long;

#

# memory lay out, please make sure memory layout is correct

# before use this perl script

#

my $DS1_OFFSET = 0x2c0;

my $DS2_OFFSET = 0x82c0;

my $SS1_OFFSET = 0x0;

my $SS2_OFFSET = 0x100;

#

# notes: this is offset that J9 use for generate burn image

# this offset will store in the burn image

# the OTA firmware updater need to compensate this offset

#
```

```
my $ds_offset_GenBurnImage=0x2c0;

my $hidht   = "usb0";

my $dutaddr = "207330002222";

#

# Firmware image for J9 module, this image should

# Be generated by BlueTool "BurnImage"

#

my $burnimg = "testimage2.hex";

#

# EEPROM base address for 20733

# fix value, don't change this.

#

my $EEPROM_base_addr=0xFF000000;

my $ss_block;

my $new_DS_offset = $DS2_OFFSET;

my $current_DS_offset;

my $new_SS_offset;

my $current_SS_offset;

#

# These are more or less constants of the HID OTAFU protocol

#

my $ID_ENABLE_OTAFU = 0x70;

my $ID_SETUP_READ   = 0x71;

my $ID_READ         = 0x72;

my $ID_ERASE        = 0x73;

my $ID_WRITE        = 0x74;

my $ID_LAUNCH       = 0x75;

my $CONFIG_FIRST_ITEM = 0x01;

############################################################
```

```
#################################################################
# START
#################################################################
#################################################################
BTSP::Open( $hidht );
BTSP::SetProtocol( $hidht, "HCI" );
BTSP::EnableACLReceipt();
 printf "Searching valid SS section.\n";
    $current_SS_offset = $SS1_OFFSET;
 # 1. seach the SS section block
    # will search 0x0 and 0x100
    # SS size is 0x28 bytes
    $ss_block = ReadMem($EEPROM_base_addr+$current_SS_offset, 0x28);
 #
    # if first byte is not 0x1 the for sure not the correct SS
    # try 0x100
    # Don't need to do detail checking for SS block here,
    # Since the device is "runing" which means SS block already
    # fully checked by bootloader.
    # And SS block is correct for sure. :)
    #
    # no redundant check here for the sample code
    #
    if ($ss_block->{'data'}->[0] != $CONFIG_FIRST_ITEM) {
        $current_SS_offset = $SS2_OFFSET;
        $ss_block = ReadMem($EEPROM_base_addr+$current_SS_offset, 0x28);
    }
 if ($ss_block->{'data'}->[0] != $CONFIG_FIRST_ITEM) {
        printf "\nNo SS found :( !!";
        die;
    }
```

```
printf "Valid SS secion offset =%x\n",$current_SS_offset;
#
   # 2. decided the new SS offset
   #
   if($current_SS_offset == $SS1_OFFSET)
   {
       $new_SS_offset = $SS2_OFFSET;
   }
   else
   {
       $new_SS_offset = $SS1_OFFSET;
   }
#
   # 3. get the current DS offset from SS block
   #
   $current_DS_offset = $ss_block->{'data'}->[0x1e] |
($ss_block->{'data'}->[0x1f] <<8) |
   ($ss_block->{'data'}->[0x20] << 16) |
($ss_block->{'data'}->[0x21] << 24);
printf "Current DS offset =%x\n",$current_DS_offset;
#
   # 4. decided new DS offset
   #
if($current_DS_offset == $DS1_OFFSET)
   {
       $new_DS_offset = $DS2_OFFSET
   }
   else
   {
       $new_DS_offset = $DS1_OFFSET
   }
```

```
    printf "New DS offset =%x\n",$new_DS_offset;
#
    # 5. enable OTA
    #
printf "Enable OTA.\n";
    EnableOtafu();
#
    # 6. Write new burn image into new DS offset
    #
    printf "Writing Firmware image.\n";
    BurnImage($new_DS_offset);
# 7. now write back the new SS
    # invalid this SS block first
    # for possible the first signature is correct
    # somehow the following data invalid
    $ss_block->{'data'}->[0]= 0;
printf "\nUpdate new SS to offset=%x\n",$new_SS_offset;
# update the new DS offse to SS
    $ss_block->{'data'}->[0x1e] = $current_DS_offset&0xff;
    $ss_block->{'data'}->[0x1f] = ($current_DS_offset >>8)&0xff;
    $ss_block->{'data'}->[0x20] = 0;
    $ss_block->{'data'}->[0x21] = 0;
WriteMem($EEPROM_base_addr+$new_SS_offset, $ss_block->{data});
# 8. make valid new SS
    # make sure above function call success before valid it
    WriteMem($EEPROM_base_addr+$new_SS_offset, [0x01]);
printf "Invalid Old SS at offset=%x\n",$current_SS_offset;
```

```
# 9. invalid old SS

   WriteMem($EEPROM_base_addr+$current_SS_offset, [0x0]);

print "Complete, resetting HID device (will lose connection).\n";

   Launch(0);

   exit 0;
```

5 Sample C Code for Updating Firmware to HID EEPROM

This code is provided only as a reference for those using C to update HID EEPROM firmware over-the-air.

```
//

// just temporary use this value

#define SIZE_PER_PACKET     0x30

#define DS1_OFFSET          0x2c0

#define DS2_OFFSET          0x82c0

#define SS1_OFFSET          0x0

#define SS2_OFFSET          0x100

#define BURN_IMAGE_OFFSET   0x2c0


void main(void)

{

    UINT32              current_DS_offset;

    UINT32              currest_SS_offset;

    UINT32              new_SS_offset;

    UINT32              new_DS_offset;

    UINT8               currest_SS_data[SS_BLOCK_SIZE];


    FILE                *cfg_file;

    UINT8               filename[]="SampleCgs.hex";

    UINT8               *p;

    UINT8               tempbuf[SIZE_PER_PACKET];


    if(searchForSS(&currest_SS_offset,currest_SS_data ) == FALSE)
```

```
    {
        printf("\n\nSomething wrong");
        return;
    }
 current_DS_offset = OTAFWU_GetDSOffset(currest_SS_data);

    // toggle the half size bit
    new_DS_offset = (current_DS_offset == DS1_OFFSET)?DS2_OFFSET:DS1_OFFSET;
 new_SS_offset =  (currest_SS_offset== SS1_OFFSET)?SS2_OFFSET :SS1_OFFSET;
 OTAFWU_SetDSOffset(currest_SS_data,new_DS_offset);

    // now start write data to HID device via OTA
if ((cfg_file = fopen(filename, "r")) == NULL)
    {
        printf("\nCannot open file %s", filename);
 return ;
    }
 do
    {
        // check each records
        // here just a sample code
        // and skip the "offset" inside the record
        p = fgets(tempbuf, SIZE_PER_PACKET, cfg_file);
 if(p == NULL)
            break;
 OTAFWU_write_mem(new_DS_offset -BURN_IMAGE_OFFSET, SIZE_PER_PACKET,tempbuf);
 new_DS_offset += SIZE_PER_PACKET;
 }while(p);
 // invalid SS2 first
    currest_SS_data[1]=0;
 // write the new SS2 to target
```

```
    OTAFWU_write_mem(new_SS_offset,SS_BLOCK_SIZE,currest_SS_data);

 // Valid SS2, only when above function call success

    currest_SS_data[1]=0;

 // write the new SS2 to target

    OTAFWU_write_mem(new_SS_offset,1,currest_SS_data);

 // erase SS1

    // just write the first byte to 0, this will currupt SS1

    tempbuf[0]=0;

    OTAFWU_write_mem(SS_Offset,1,tempbuf);

}
```

6 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 Cypress 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		Cypress Number	Broadcom Number	Source
[1]	Single-Chip Bluetooth Transceiver for Wireless Input Devices, Data Sheet	20730-DS1xx-R	002-014280	community.cypress.com
[2]	Single-Chip Bluetooth Transceiver for Wireless Input Devices, Data Sheet	20733-DS0xx-R		community.cypress.com
[3]	Upgrading Human Interface Device Firmware (Example Protocol, Commands, and Scripts), Application Note	20730_20733-AN4xx-R		
[4]	Application Development Kit (ADK)	–		community.cypress.com
[5]	BlueTool™	–		community.cypress.com

Document History

Document Title: AN214820 - EEPROM Format Information and Firmware Upgrading Document Number: 002-14820				
Rev.	**ECN No.**	**Orig. of Change**	**Submission Date**	**Description of Change**
**	–	–	04/08/2013	20730_20733-AN300-R: Initial release
*A	5475543	UTSV	10/14/2016	Updated to Cypress template
*B	5841867	AESATP12	08/02/2017	Updated logo and copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturers' representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Video | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709