<Product Area>

<Device>

Timberwolf Host Boot

Draft Version 0.1

June 2015

# Table of Contents

[1 Introduction 5](#_Toc441831733)

[1.2 Purpose of the Document 5](#_Toc441831734)

[1.3 Scope 5](#_Toc441831735)

[1.4 Abbreviations 5](#_Toc441831736)

[1.5 References 5](#_Toc441831737)

[1.6 Assumptions 5](#_Toc441831738)

[2 S-Record 5](#_Toc441831739)

[3 Proposed Image Format 7](#_Toc441831740)

[3.1 HBI Header 7](#_Toc441831741)

[3.2 Command Based Boot Image 7](#_Toc441831742)

[3.2.1 HBI Header parsing 9](#_Toc441831743)

[3.2.2 Length of each HBI command entry 9](#_Toc441831744)

[3.2.3 Partial data handling 9](#_Toc441831745)

[3.2.4 Convertor Tool Requirement 9](#_Toc441831746)

[3.2.5 Command line input options to Convertor tool 9](#_Toc441831747)

[5 Advantages 10](#_Toc441831748)

# List of Tables

[Table 1 Abbreviations used in this document 5](#_Toc441831446)

# List of Figures

[Figure 1 HBI Header 7](#_Toc441831685)

[Figure 2 output Image Format 8](#_Toc441831686)

[Figure 3 Convertor input/output 9](#_Toc441831687)

|  |  |
| --- | --- |
| **Document Owner** | Shally Verma |
| **Version** | Draft |
| **Date** | 24 June 2015 |

| Document Change History | | | |
| --- | --- | --- | --- |
| Date | Version | Changed by | Change Description |
| 7 July 2015 | 0.1 | Shally Verma | Update document as per review comments. Add a header description to an output file |
| Oct 2015 | 0.1 | Shally Verma | Reworked on HBI header to add “firmware length” field. Added one more image format type |
| Jan 2016 | 1.0 | Shally Verma | Rework after review. Image now generated into fixed size chunk. Size is as selected by user during conversion and is in unit of words.Block size now goes as 1-time parameter into Header field. |

# Introduction

Microsemi VPROC device supports 3 different booting options:

* Booting from host (over I2C and SPI HBI)
* Booting from flash
* Booting from UART

This document specifically focused on “Booting from host” option. Current Microsemi VPROC device HBI driver in its present form support S-record based firmware image loading to device RAM which reads every record, gets the data length, make up an HBI command and then push it over HBI. This approach works way too slow and takes appx 1 sec over SPI running at 25Mhz and even much slower on I2C.

Intention behind this document is to come up with a proposal on reformatted boot image to enable faster booting of device via host.



## Purpose of the Document

This is proposal document for boot image format to enable fast booting over SPI and I2C HBI.

This document will cover

* Basic overview of S-Record based file
* Proposed format of optimized boot image
* Necessary tool to generate optimized boot image

## Scope

The content covered in this document in its present state is only specific to TW VPROC device family.

## Abbreviations

| Table 1 Abbreviations used in this document | |
| --- | --- |
| Abbreviation | Explanation |
| HBI | Host Bus Interface |
| VPROC | Voice Processor |
| TW | Timberwolf |

## References

[1] ZL38040/50/60/80/51 firmware Manual

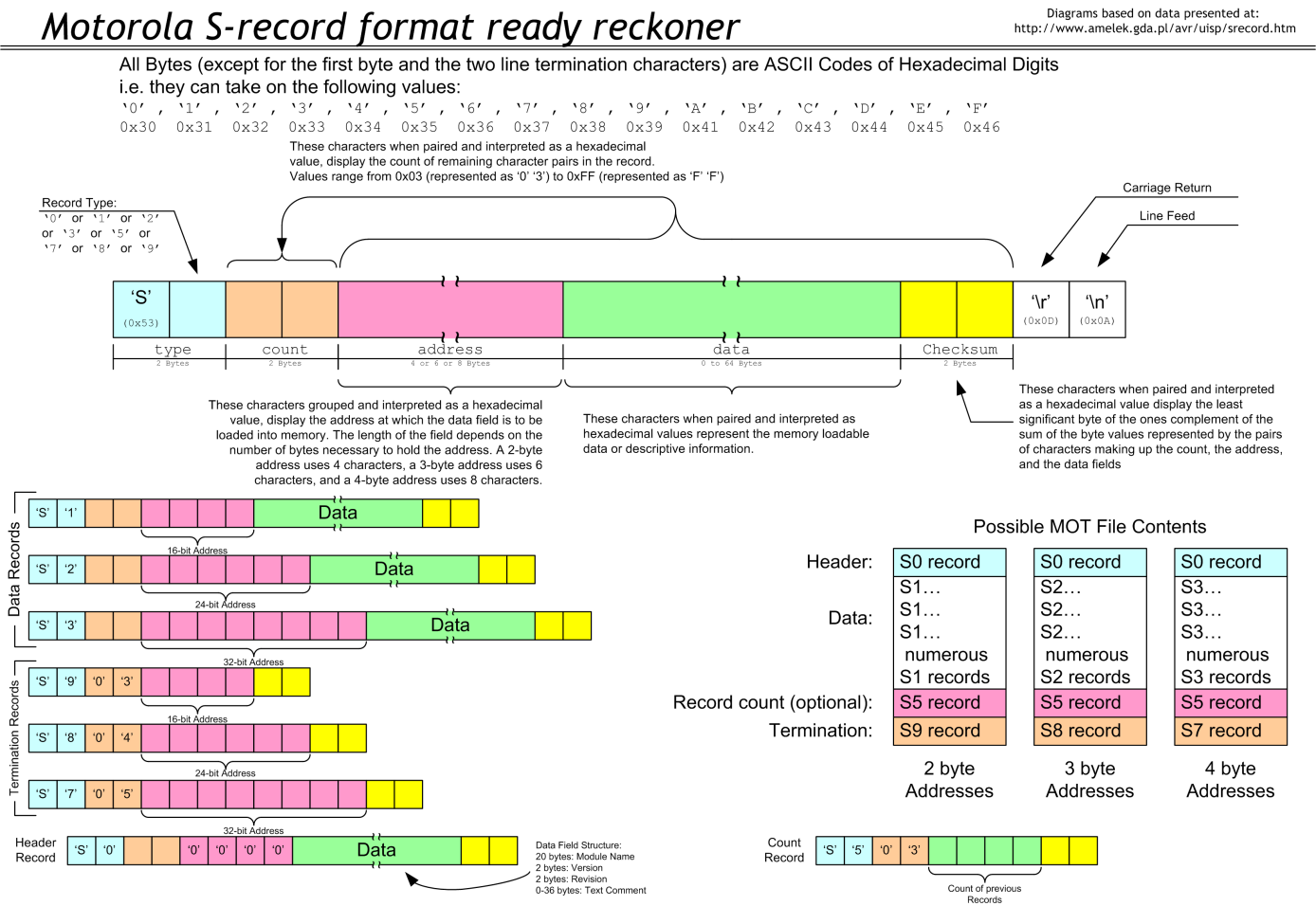
[2] https://en.wikipedia.org/wiki/SREC\_(file\_format)

## Assumptions

This document assumes that user is aware of voice processor device HBI.

# S-Record

Below diagram gives an overview of complete S-record based entries and how to interpret them.



As we can see, every entry in S-record based format starts with character ‘S’ followed by record\_type (0,1,2,3,5,7,8,9) , length and data information.

The current TW firmware image consists of following record types:

0 - Entry with record type 0 is a header record and presumably consists of firmware version/revision information (??) and is normally ignored.

7 - Entry with record type 7 is termination record and consists of execution address of firmware. Content of this record needs to be programmed in Execution Address Register (ex. reg 0x12C of TW ZL38040) device.

3 – Entries with record type 3 are the actual firmware image along with Device RAM load address. The data format of these record types consist of:

<Device RAM addresses to be written><data><last 2 chars of checksum>. Example, below entry consists of total 37 bytes (shown as 25 as hex) of data inclusive 1-byte checksum. Actual characters in entries will be 37\*2 (every char is a hex value)

S325000BEF40849A92C3C0FFFCC190C053947DC2CB4390C0EB1A539325E9CC4394C0CC4E8033B4

# Proposed Image Format

Each of the above entry with record value 3 and 7 are parsed and translated to HBI write command and Data words.

Proposal here is we can devise a tool which can input this S-record file and output a buffer containing different image types.

## HBI Header

The reformatted boot image will begin with HBI header. This header will contain following image information.

Proposed HBI header will look like as in Figure 1



Figure HBI Header

* Header version number.
  + It can be combination of Major.Minor where

2-bit = Major and

2–bit = Minor,

4-bit can be kept reserved for future extension for more granular information on version

* 1 byte to keep miscellaneous firmware image information:
  + Image\_type
    - 2-bit = boot image or device configuration record
  + Endianness
    - 1 bit = Little / Big
* Firmware OPN number
  + To match target device OPN for which image is generated
  + 16-bit = keeping future extension
  + To be written Most Significant Byte 1st
* Firmware Block Size
  + Block Size image divided to with Most Significant Byte 1st (Given at the time of image generation)
* Total PayLoad Length
  + Length of the data following header in Bytes
* Reserved
  + 16-bit reserved for future use. Added to maintain 32-bit alignment in header field

## Command Based Boot Image

Whole image would consist of a Header followed by HBI command-data pairs.Image either can be stored as binary OR a ‘C’ style output so that driver can dynamically or statically input it and without any parsing overhead writes it to device over Host Bus Interface (i2c/spi).

Pictorial representation shows how input firmware image will be parsed and data will be converted



Figure output Image Format

HBI Command will always be coded in Big endian. Data can be big/little endian depending upon user choice. User can give endian as a runtime parameter to Convertor tool.

### HBI Header parsing

Application should parse HBI header OR call driver HBI\_get\_header() to get header fields values and perform sanity check and error condition check. Example, OPN match, endian match, checksum check (if present) and parse whole chunk in entirety to driver. A partial chunk write will result in load error at the end.

### Length of each HBI command entry

Whole image Excluding payload will be divided into fixed block size. Block size will be selected by user and can be given as runtime parameter to Image conversion tool. Block size unit will be in word for Timberwolf device firmware image. Application is expected to start reading payload in that block size and pass down to driver

### Partial data handling

TW device accepts data, associated with each HBI command, to be sent in one shot i.e. there should be no STOP condition (in case of i2c) or CS deactivation(for SPI) until whole data associated with HBI command-data pair is sent to TW. Otherwise device would reset and will generate checksum error condition after “Boot Complete” event. Thus it is necessary that application passes one complete chunk to driver to further write to device RAM. Driver doesn’t do any check on expected or actual length of data or data itself. When writing binary image format it acts as a carrier which takes input from supplier and writes to PHY without any processing.

### Convertor Tool Requirement

Here’s block diagram showing input and output of tool



Figure Convertor input/output

### Command line input options to Convertor tool

* Block Size – whole image will be divided in to size as given by this parameter. Unit is in words where each word length is 16-bit.

Default value for generating firmware image is 16words

* Endiannes - HBI command will still be in Big endian however data can be put in either big or little endian. Useful if TW device is configured in little endian
* Output format type.
  + .bin ( a raw bin file consisting series of HBI command)
  + .c ( dumped as a array of chars into ‘C’ style file)
* Firmware – tells the firmware code ex. 38050/38040/38051 etc.

# 5 Advantages

* Low Software processing.
* Flexibility to generate images according to TW device configuration in big endian/little endian
* Easier to evaluate performance by generating images with different chunk length.