<Product Area>

<Device>

Generic Software Architecture

Draft Version

Mar 2015

# Table of Contents

[1. Introduction 6](#_Toc424724314)

[1.1 Purpose of the Document 6](#_Toc424724315)

[1.2 Scope 6](#_Toc424724316)

[1.3 Abbreviations 6](#_Toc424724317)

[1.4 Definitions 6](#_Toc424724318)

[1.5 References 7](#_Toc424724319)

[1.6 Assumptions 7](#_Toc424724320)

[2. Generic SW 8](#_Toc424724321)

[2.1 Generic Software Requirement 8](#_Toc424724322)

[2.2 Generic SW Life Cycle Development 8](#_Toc424724323)

[2.3 Generic Software Architecture Overview 8](#_Toc424724324)

[2.1.1 System Service 9](#_Toc424724325)

[2.1.2 Host Bus Interface Layer 10](#_Toc424724326)

[3 Code Organization 11](#_Toc424724327)

[4 Version control 14](#_Toc424724328)

[5 Release Milestones 15](#_Toc424724329)

[6 Development Platform 16](#_Toc424724330)

[6.1 Gen SDK Build System 17](#_Toc424724331)

[6.1.1 Requirements 17](#_Toc424724332)

[6.1.2 Main Components 17](#_Toc424724333)

[6.1.3 Compile-time Variables 17](#_Toc424724334)

[6. Test Specification 21](#_Toc424724335)

[7. Open Issues 22](#_Toc424724336)

[Appendix 23](#_Toc424724337)

# List of Tables

[Table 1 Abbreviations used in this document 6](#_Toc415645286)

# List of Figures

[Figure 1 Generic SW Architecture Block Diagram 8](#_Toc415645290)

[Figure 2 Generic SW Code Organisation 12](#_Toc415645291)

[Figure 3 Generic SW SVN Code Organisation 13](#_Toc415645292)

|  |  |
| --- | --- |
| **Document Owner** | Shally Verma |
| **Version** | Draft |
| **Date** | March 2015 |

| Document Change History | | | |
| --- | --- | --- | --- |
| Date | Version | Changed by | Change Description |
| 02 April 2015 | 0.1 | Shally Verma | Incorporated changes as an outcome of architecture document review. Major changes include Simplify Kernel Abstraction layer requirement. Consolidate both HAL (hardware abstraction) and KAL(kernel abstraction) in to a single SSL. Modify Code Organization to move HAL into platform. Development scope also consider Galileo device class as well. |
| July 2015 | 0.2 | Shally Verma | Added section on Gen SDK build system. Current status and proposed changes |
|  |  |  |  |

# Introduction

Microsemi Voice Processor family are group of devices which support Acoustic and Line Echo cancellation for hand-held and hands free communication devices. This field upgradeable platform, when combined with algorithmic firmware, enables beam forming, multi-channel Acoustic Echo Cancellation (AEC), direction of arrival and far field MIC capabilities.  These powerful new technologies are the foundation for automatic speech recognition, sound classification and other intelligent decision making functions based on sound and audio processing.

## Purpose of the Document

This document is an Architecture proposal document for Generic Software Development for voice processor devices. Intention is to exploit lot of commonality that exists between various variants of communication interfaces of these devices and avoid redundancy and various versions of software solutions that we provide to customers. Having a generic software platform would enable developer to port & customize solution with minimal time and lot of ease. There’s no separate requirement specification document for this project. The architecture document is believed to be considered as requirement specification as well.

## Scope

Scope of this document will only be limited to cover sections which deemed necessary for generic software development cycle. It will cover a description on proposed software layers, their organization, development platform and code version control and any other section as deemed necessary to be mentioned. This document only gives an overview. Specific details of specific layer will be covered in separate design/specification documents.

UART will not be a part of this development.

## Abbreviations

| Table Abbreviations used in this document | |
| --- | --- |
| Abbreviation | Explanation |
| SSL | System Service Layer |
| SW | Software |
| VPROC | Voice Processor Devices |
| HBI | Host Bus Interface |
| HAL | Hardware Abstraction Layer |

## Definitions

Class –Term used to refer to a product line consisting of different variants with same base characteristics. Example, Timberwolf is a class zl38040, 50, 60 are variants under this.

Timberwolf - Microsemi ZL38040/50 series of Voice Processor Platform

Galileo - Microsemi ZL3800x series of Voice Processor Platform

Host - 3rd party SoC vendor like NXP, Ambarella, Raspberry Pi etc

Target – Refers to Microsemi VPROC device or device class ex. Timberwolf, Galileo or more specific zl38040/50/51/60/80

HBI- refers to SPI and I2C bus interface of Microsemi VPROC device

## References

## Assumptions

# Generic SW

## 2.1 Generic Software Requirement

Generic Software System should meet following requirements:

* As far as possible, all of the software layer in gamut of Generic code must be deterministic in nature on their resource requirements thus avoiding dynamicity of the driver especially memory. Design must avoid dynamic allocations to favor the pure embedded systems
* Layers should be as thin as possible and simplistic in prototyping & implementation.

## Generic SW Life Cycle Development

Generic SW must follow a procedure to start where each SW module must be accompanied with design/specification document with minimal coverage on API and Public Interfaces and list of test cases. Every Module may end with test result/report covering test cases identified.

## Generic Software Architecture Overview

Different voice processor device class differs on their communication protocols. Current description as of now covers only Timberwolf class of devices.

Timberwolf class of devices supports two level of communication interface with host:

* Physical interface (bus) between host and timberwolf devices, and
* Transport Host Bus Interface which defines communication protocol between host and timberwolf. This makes up payload for data sent over physical interface.

While all of the timberwolf variants share common “Host Bus Interface (HBI)” protocol but there may be minor variations in type and number of physical interfaces[See Appendix Section for details on different variants].

Below block diagram (Figure 1) gives an overview Generic Software flow w.r.t communication layer described above:



Figure Generic SW Architecture Block Diagram

### System Service

This layer is combination of both Hardware Abstraction and Operating System Abstraction. It should provide basic operating system services such as memory allocations, locking/unlocking mechanism to user also hardware port access: initialization, open, read and write calls to physical bus.

The layer will be responsible to initialize and setup, send and receive user data and frame it according to bus in use. Specification should be generic in nature so as to make user agnostic of actual physical interface in use. HAL specification must cover minimum transaction and configuration requirement of all possible physical media and meet the requirement as outlined in Generic Software Requirement.

In theory, this layer defines API and public interfaces which will be platform-independent however practically, it will be a platform dependent code and implemented upon host platform base operating system layer and host bus driver.

Scope of development of this layer is only limited to specification with example reference code. In end scenario, this is expected from users to implement it according to their host environment in use. **every development platform using voice processor device is expected to port this layer according to their environment**.

This layer should be providing following minimal operations:

* Print APIs for error reporting and informational messages
* Memory operations: Clear and Copy
* Synchronisation primitives: locking/unlocking (preferably mutex locking)
* bytewide address read/write operations on bus
* initialization / termination
* opening / close

### Host Bus Interface Layer

HBI Driver implements “HBI Protocol” as described by HBI Chapter of Timberwolf device firmware manual which defines communication protocol between host and timberwolf device.

HBI will sit on top of SSL and supposed to using Only & Only interfaces exposed by this layer to configure itself and communicate to timberwolf device.

This layer will also contain utilities functions so as to support Boot image & configuration record download and such operation to facilitate user to setup device in particular mode. These utilities functions will be compile-able under the conditional compile time directives.

This layer must not call any platform dependent code and meet the requirement as outlined in “Generic Software Requirement Section”.

Specification of both layers will be covered in separate API specification document.

# Code Organization

Figure 2 describes current sdk organization as it will reside on trunk where all the development activity will go on. Once a release milestone is achieve, code will be branched to “Release” branch and tagged.

Please note this gives an overview and subject to revision.



Figure 2 Generic SW Code Organisation

**trunk** is master root folder containing whole sdk.

**drivers** would contain generic software by Microsemi. This software should be platform-independent code making reference to hal and ssl.

Each driver\* folder would have:

* **inc** – includes public header files only which are meant for external world use. No internal files should be in here.
* **doc** – contains all of the internal / external documentation as developed during development phase, and
* **driver** **source** **files**. single or combination of files depending upon code complexity.
* **Makefile**

\***Note** - SSL driver would contain an example implementation code for development and test purpose. However for release, only HBI driver will be provided with full-fledged implementation with source code. HAL and SSL only specification document will be given during release.

**docs** – apart from each driver holding their own “doc” folder. This is “external” doc folder. All the release documentation is supposed to reside in this.

**tst** – contains test files

**platform** – contains host platform environment on which generic sdk is tested. This is also be called as “host” containing host specific information.

**lnxdriver** – would contain linux specific driver for HBI, hal and ssl. Hal and SSL may or may not be released. This further would contain kernel and userspace version of driver. Driver structure here is also kept same as “generic driver” folder.

**environment setup file** - script that setup toolchain, kernel src and other build configs. This is also define target chip type host is integrated to. HBI driver will link to target as setup by this environment variable.

**Include** – all files that are globally included by all SDK element

There will be root makefile which should be supporting minimal rule to make release package and other targets.

# Version control

Figure 3 shows proposed Generic SDK organization in svn.main



Figure 3 Generic SW SVN Code Organisation

**vproc\_sdk** project will be created to keep generic software work here: http://aussvn01/svn/apps/vproc\_sdk/

**trunk** is main branch where all development activity will go on.

**branches** – contain specific branch. Release will come under it

**tags** – contains release tags.

# Release Milestones

This release as of now is divided into 3 milestones:

1. 1st milestone

This milestone would include Generic SDK tested on I2C and SPI interface of timberwolf device with Linux host platform

1. 2nd Milestone

This would have Generic SDK tested on Galileo devices with Linux host platform

1. 3rd Milestone

This would have Generic SDK tested with non-os host platform on either timberwolf or galileo.

# Development Platform

We will be using Raspberry Pi 1 Model B for this project development. All of the raspberry pi kernel source code, toolchain will go in “Platform/raspberry” folder within Gen SDK.

There’s a separate document covering “How To” on configuring RPI for Development purpose. Please refer to “RpiSetup.doc” for file sharing between RPI and windows PC, setting it up for native and cross compiling of modules.

# 6.1 Gen SDK Build System

### Requirements

Gen SDK Build system should meet basic two prime requirements:

* Should be configurable to compile for various target devices. Example, Timberwolf, Galileo. If more granular build is required, then Target can be defined as ZL38040, ZL38004 , ZL38050 etc
* Should be configurable to compile for any host including their hard and soft limits. Example of hard limits includes selection of VPROC device, VPROC HBI type (SPI/I2C), total number of VPROC devices in system etc. Soft limits include Driver name, maximum number of users of device in a system, maximum number of mutual exclusion locks in a system, debugging level etc.

### Main Components

Gen SDK build system consists of 3 main files present at Root

* Makefile – defines target rules
* Makefile.globals – placeholders to define all compile-time variables for Gen SDK
* config.mk – reads all Makefile.globals variables and define them as compile time build options

### Compile-time Variables

Below table summarize list of variables currently defined by Gen SDK build options in Makefile.globals :

|  |  |  |
| --- | --- | --- |
| Variable | Description | Values |
| TARGET | Identify Target VPROC device | TW=1 \* |
| GAL=2 |
| HBI | Identify Host Bus Interface to VPROC device | I2C=1 \* |
| SPI=2 |
| HBI\_BUF\_SIZE | Allocate a buffer size used by HBI driver transactions | 1024 or user defined |
| HOST\_ENDIAN | Identify host endianness | little , big |
| VPROC\_DEV\_ENDIAN | Identify device endianness | little,big |
| BOOT\_FROM\_HOST | Option to enable Booting of target device over HBI | yes, no |
| FLASH\_PRESENT | Option to indicate is target device has boot flash | yes,no |
| BUILD\_TYPE | Indicate the build type | DEBUG, RELEASE |
| HBI\_MAX\_INST\_PER\_DEVICE | Maximum users on one device | User defined value, by default 1 |
| HBI\_MAX\_INSTANCES | Maximum HBI Driver user | User defined value, by default 1 |
| VPROC\_MAX\_NUM\_DEVS | Maximum number of devices in system | User defined value, by default 1 |
| NUM\_MAX\_LOCKS | Maximum number of locks deployed for mutual exclusion | User defined value, by default 100 |
| DEBUG\_LEVEL | Set the debugging information from HBI /SSL driver on console | 0= none  0x1 = function Entry/Exit info,  0x2 = informational  0x4 = warning  0x8 = error  0x1F = All |

Table

\*currently we assume that all devices under Timberwolf or Galileo class shares same property over HBI thus we limited our assignment to TW or Gal. However if we need chip specific version then we can redefine Targets as:

|  |  |
| --- | --- |
| Target | Zl38040=1 |
| Zl38050=2 |
| Zl38060=3 and so on |

To build Gen SDK for specific target and bus, user can pass make command line arguments “TARGET” and “HBI”. For ex. to build Gen SDK for TW with I2C as HBI, invoke

make TARGET=TW HBI=I2C

If there’re no command line arguments then, by default, these parameters are set to TW and SPI respectively.

#### Defining target specific compile time variables

Currently, all build options are defined in Makefile.globals. Since the list of identified variables (at the time of writing this document) is small, thus we can live with them all present in Makefile.globals. However, if it grows bigger where each target defining their own configuration, then we can opt to define a target-specific-configuration file (for ex zl380\_tw.cfg for TW, zl380\_gal.cfg for galileo) which may override existing and/or define new compile-time variables specific to that target. In such implementation, Makefile.globals would keep all common and default values for each compile time option and include config file per user selected target. In such case, snapshot of makefile would be something like:

ifeq ($(TARGET),TW)

include zl380\_tw.cfg

else

ifeq ($(TARGET),GAL)

include zl380\_gal.cfg

endif

endif

#### Making HBI Driver configurable

HBI driver is specific to target device. To compile driver for any target type, each target is assigned a unique key so that driver can identify which device it is compiling for. These keys currently defined in makefile.globals or may be defined target-specific-file, if used.

#### Dynamic or Static linking

There are two possible approaches to import all configurable options down to drivers / source files

**Approach # 1 Static linking (Currently supported)**



Here, all options statically imported to each source file makefile which build the driver according to these options. This we are referring as static linking

**Approach # 2 Dynamic linking**



This means all of the configurable options are exported into dynamically generated include files. These files further included by each driver during compilation. This would require a utility to parse and read configuration file and dynamically generate a include file.

**Being low in scope, current SDK system is living with Approach 1. In future, if need be we may relook to consider dynamic linking approach**.

# Test Specification

Every modules need to be unit tested with bare test code that ensure basic exported features are working fine. Details of individual module test specification will be covered in respective design document.

# Open Issues

# Appendix

Below table summarizes physical interfaces as supported by timberwolf class of devices for host communication. This table may not currently cover all of the devices as fall under Timberwolf class however still cover the minimal set required to be supported by Generic Software.

|  |  |
| --- | --- |
| Device | Interfaces |
| ZL38040 | Slave I2C,SPI,UART |
| ZL38050 | Slave I2C,SPI,UART |
| ZL38051 | Slave I2C,SPI,UART |
| ZL38060 | Slave I2C, SPI,UART |
| ZL38080 | Slave I2C, SPI,UART |