AN14250

Implement LVGL GUI Voice Recognition on Framework

Rev. 1 — 26 March 2024 Application note

Document information

Information	Content
Keywords	AN14250, smart HMI, smart TLHMI, framework
Abstract	This application note describes how to use DSMT or VIT model to enable voice recognition feature on framework by a simple LVGL GUI example on SLN-TLHMI-IOT board.

1 Overview

NXP has launched a solution development kit named as SLN-TLHMI-IOT, which focuses on smart HMI applications. It enables smart HMI with ML vision, voice, and graphics UI implemented on one NXP i.MX RT117H MCU. Based on the SDK, the solution software is constructed on a design called framework, which supports flexible designs and customization of vision and voice functions. To help the users to use the software platform better, some basic documents are provided, for example, the software development user guide. The guide introduces the basic software design and architecture of the applications covering all components of the solution including the framework to help the developers more easily and efficiently implement their applications using the SLN-TLHMI-IOT.

For more details about the solution and relevant documents, visit:

NXP EdgeReady Smart HMI Solution based on i.MX RT117H with ML Vision, Voice and Graphical UI | NXP Semiconductors

However, it is still not so easy for the developers to implement their smart HMI applications referring to these basic guides. A series of application notes are planned to help study the development of the framework step by step from basics. This application note is based on the application note that shows how to enable LVGL GUI on framework with a simple GUI camera preview example.

This application note describes how to use the DSMT or VIT model with English and Chinese languages to enable the voice recognition feature on the framework by a simple LVGL GUI example of the SLN-TLHMI-IOT board.

In the application note, the example presents an LVGL GUI screen with a camera preview and some buttons, which can be triggered by voice or touch.

At a high level, the application note contains the below contents:

- Enable the voice recognition feature on the framework.
- Implement an LVGL GUI application.

Through the above introductions, this document helps the developers be able to:

- Understand the framework and the smart HMI solution software more deeply.
- Develop their voice recognition on framework with LVGL GUI application.

1.1 Framework overview

The solution software is primarily designed around the use of a **framework** architecture that is composed of several different parts:

- Device managers – Core part
- Hardware Abstraction Layer (HAL) Devices
- Messages/Events

As shown in Figure 1, the overview of the mechanism of the framework is:

Device managers are responsible for "managing" devices used by the system. Each device type (input, output, and so on) has its own type-specific device manager. After registering the devices, a device manager initializes and starts them, then waits for a message to transfer data to other managers and devices.

The HAL devices are written **on top of** lower-level driver code, helping to increase code understandability by abstracting many of the underlying details.

Events are a means by which information is communicated between different devices via their managers. When an event is triggered, the device that first receives the event sends it to its manager who then notifies other designated managers.

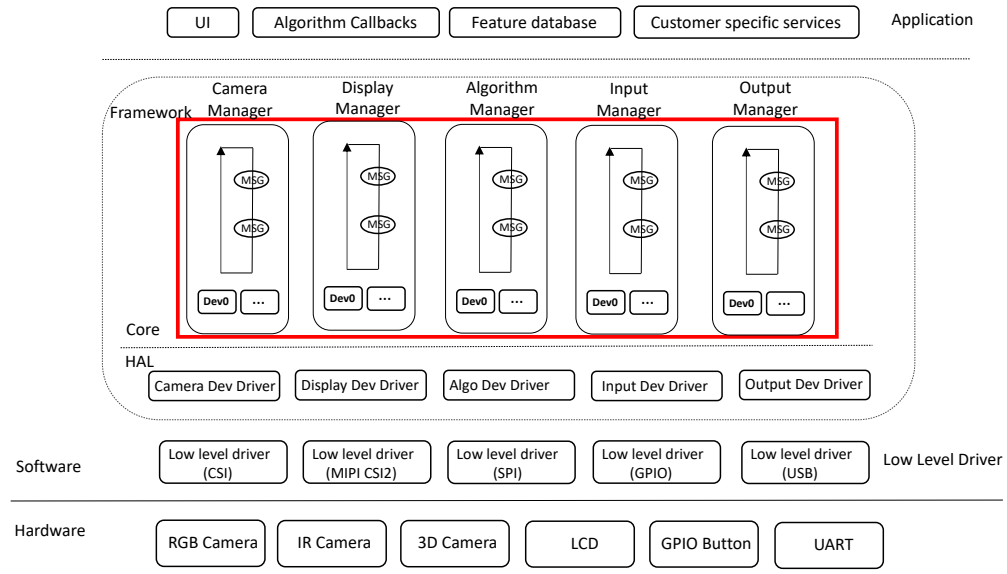

Figure 1. Smart HMI software architecture on framework

The architectural design of the framework is centered on three primary goals:

- Ease-of-use
- Flexibility/Portability
- Performance

The framework is designed with the goal of speeding up the time to market for vision and other machine-learning applications. To ensure a speedy time to the market, it is critical that the software itself is easy to understand and modify. Keeping this goal in mind, the architecture of the framework is easy to modify without being restrictive, and without coming at the cost of performance.

For more details about the framework, see the *Smart HMI Software Development User Guide* (document MCU-SMHMI-SDUG)

1.2 Light and Versatile Graphics Library (LVGL)

LVGL is a free and open-source graphics library. It provides everything that you require to create an embedded GUI with easy-to-use graphical elements, beautiful visual effects, and a low memory footprint.

1.3 GUI Guider

GUI Guider is a user-friendly graphical user interface development tool from NXP that enables the rapid development of high quality displays with the LVGL Open-Source Graphics Library. The drag-and-drop editor of GUI Guider makes it easy to use the many features of LVGL such as widgets, animations, and styles to create a GUI with minimal or no coding at all.

With the click of a button, you can run your application in a simulated environment or export it to a target project. Generated code from GUI Guider can easily be added to your project, accelerating the development process and allowing you to seamlessly add an embedded user interface to your application.

GUI Guider is free to use with general purpose of NXP and crossover MCUs, including built-in project templates for several supported platforms.

To learn more about LVGL and GUI development on GUI Guider, see Light and Versatile Graphics Library and GUI Guider

2 Development environment

Prepare and set up the hardware and software environment for implementing the example on the framework.

- **Hardware environment**
  The hardware environment is set up for verifying the example.
  - The smart HMI development kit based on NXP i.MX RT117H (SLN_TLHMI_IOT kit)
  - SEGGER J-Link with a 9-pin Cortex-M adapter and V7.84a or newer of driver
- **Software environment**
  The software environment is set up for developing the example:
  - MCUXpresso IDE V11.7.0
  - GUI Guider V1.6.1 — GA
  - `lvgl_gui_camera_preview_cm7` — example code of implementing LVGL GUI on framework. Visit: https://mcuxpresso.nxp.com/appcodehub
  - RT1170 SDK V2.13.0 — SDK as the code resource for the development.
  - SLN-TLHMI-IOT software V1.1.2 — smart HMI source codes released on NXP GitHub repository as the code resource for the development. Visit: GitHub - NXP/mcu-smhmi at v1.1.2

For details about the acquirement and setup of the software environment, refer to: Getting Started with the SLN-TLHMI-IOT.

3 Voice recognition architecture on framework

Figure 2 shows the voice architecture on framework in the current application note.

The process in the framework HALs is:

- The input PDM MIC HAL sends the raw streams of the voice recorded from the microphones to the audio processing AFE HAL.
- The audio pressing AFE HAL triggers the Voiceseeker algo to preprocess the voice data streams, then send the clean streams to the voice algo DMST ASR HAL or voice algo VIT ASR HAL.
- The voice algo DMST or VIT ASR HAL triggers the corresponding speech recognition algo and model to recognize the voice and sends the result to the output UI HAL.
- The output UI HAL acts per the result.

Note: *No speaker is supported in the application note. It means that no echo stream is required to be processed in the application note.*

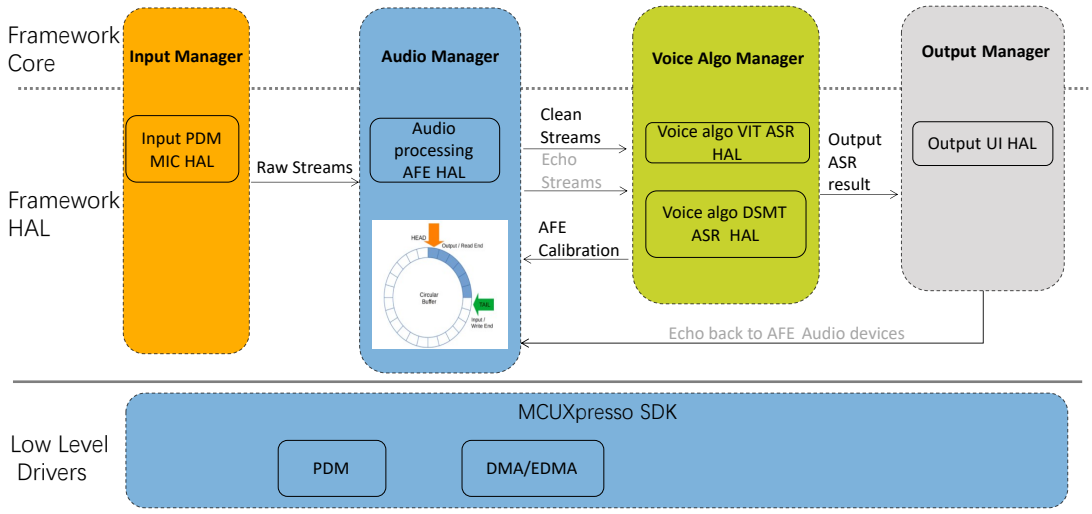

Figure 2. Voice architecture

4 Voice recognition example design on framework

The LVGL GUI voice recognition example (the example for short hereinafter) on framework is implemented based on the example code `lvgl_gui_camera_preview_cm7` (Visit https://mcuxpresso.nxp.com/appcodehub). This example enables DMST and VIT ASR models with English and Chinese languages for voice recognition feature.

For demonstrating the feature, design the GUI application as below:

- Add a standby screen presented after startup. A dropdown list widget is presented for English and Chinese languages selection. The texts on the screen are presented with the selected language and it is required to speak the wake words in the selected language for voice recognition. After recognizing the wake words or touching the screen, it enters the home screen. See Figure 3.

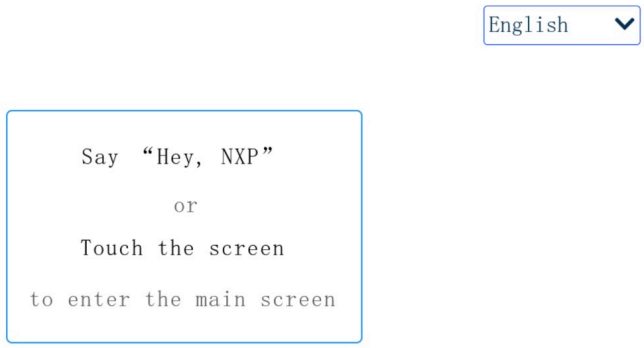

Figure 3. LVGL GUI standby screen

- Similarly, add a dropdown list widget for language selection on the home screen for the same functions with the standby screen. The hints for the supported voice commands are presented on the screen as well. See Figure 4. The voice commands have the same functions with the touch:
  - Speak **registration** or touch the button **Registration**, showingthe hint **Registration…** on the status label.
  - Speak **recognition** or touch the button **Recognition**, showing the hint **Recognition…** on the status label.
  - Speak **preview** or touch other areas out of buttons, showing**previewing…** on the status label.
- There is a timeout function for the home screen. That is, it goes back to the standby screen when the time (60 s) is up on the home screen.

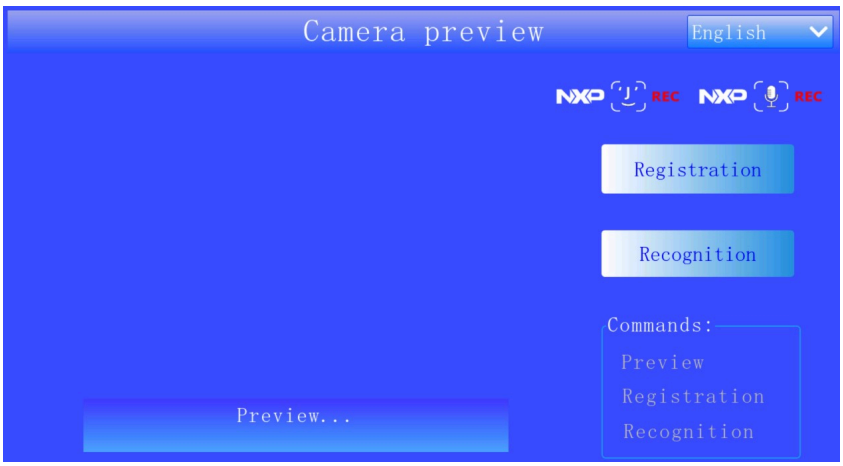

Figure 4. LVGL GUI home screen

SW Preparations

Prepare the software package for the implementation of the example.

- Clone the base software l*vgl_gui_camera_preview_cm7*. Change the project name and the main file name to *lvgl_gui_voice_rec_cm7*.
- The framework is needed to be updated in the software as the source codes for the framework core have started to be public on GitHub from the version 1.1.2.
  - Replace the framework folder with the copy of V1.1.2 from Github except for the files *fwk_log.h* and *fwk_common.h* under *inc*\ as they have been modified for the series of application note.

– Delete the folder *framework_cm7* under the `libs` group and remove the library *framework_cm7* and its search path configured in **Project** > **Properties** > **C/C++ Build** > **settings** > **Tool Settings** > **MCU C++ Linker** > **Libraries** since the source codes of core are provided.

5 Add hardware support

The hardware involved in the voice recognition example is the microphone.

5.1 Add the drivers for microphone

Add the PDM microphone interface and DMA support drivers.

1. Copy *fsl_dmamux.c* and *fsl_dmamux.h*, *fsl_edma.c* and *fsl_edma.h*, *fsl_pdm.c* and *fsl_pdm.h*, *fsl_pdm_edma.c* and *fsl_pdm_edma.h* from *SDK_2_13_0_MIMXRT1170-EVK\devices\MIMXRT1176\drivers* \ to the folder *drivers* of the example.

5.2 Add board support for microphone

1. Copy the function `BOARD_InitMicPins()` from the *[smart HMI]\coffee_machine\cm7\board\pin_mux.c* to *pin_mux.c* of the example and call it with the macro definition `ENABLE_INPUT_DEV_PdmMic` in `BOARD_InitBootPins()` in *pin_mux.c* for microphone pins settings.
2. Add the code line `#include "board_define.h"` as the above macro definition `ENABLE_INPUT_DEV_PdmMic` is defined in the file *pin_mux.c*.
3. Copy the function `BOARD_InitEDMA()` from the *[smart HMI]\coffee_machine\cm4\board\board.c* to *board\board.c* of the example for EDMA initialization.
4. Add the below code lines in the *board.c*:

```
#include "fsl_edma.h"
#include "fsl_dmamux.h"
```

5. Declare the function `BOARD_InitEDMA()` in *board.h*.

6 Enable voice recognition feature on framework

6.1 Add voice algo libraries and engines

The smart HMI solution supports Far-Field voice recognition enabled by phoneme-based Automatic Speech Recognition (ASR) engine, digital signal processing (DSP), and audio front end (AFE) which are provided by static libraries. They are used for the example as well. Below are the steps to add the libs and the engines.

1. Copy the folder *local_voice* containing the libraries for the ASR for DMST and VIT, DSP, and AFE from smart *HMI\coffee_machine\cm7\libs* into the folder *libs* of the example software.
   Note: *The VIT speech model resources with different languages are also included in the header files in the cloned folder local_voice. They are used for coffee machine application and required to be modified for the example (to be introduced in Section 6.2.2).*
   Do the configurations for the new folder:
   • Add the libs and their search paths on **Project** > **Properties** > **C/C++ Build** > **settings** > **MCU C++ Linker** > **Libraries**.

```
VoiceSeekerLight
VIT_CM7_v04_07_07
sln_asr
arm_cortexM7lfdp_math
"${workspace_loc:/${ProjName}/libs/local_voice}"
```

```
"${workspace_loc:/${ProjName}/libs/local_voice/vit/RT1170_CortexM7/Lib}"
```

Note: *The `VoiceSeekerLight` lib is for AFE. The `VIT_CM7_v04_07_07` lib is for VIT model. The `sln_asr` lib is for the DSMT model. The `arm_cortexM7lfdp_math` lib is for DSP.*
Add the search path for the header file of the libs in **Project** > **Properties** > **C/C++ Build** > **settings** > **Tool Settings** > **MCU C compiler** > **Includes** and **MCU C++ compiler** > **Includes**.

```
"${workspace_loc:/${ProjName}/libs/local_voice/vit/RT1170_CortexM7/Lib}"
"${workspace_loc:/${ProjName}/libs/local_voice/vit/RT1170_CortexM7/Lib/Inc}"
```

2. Copy the folder *audio* containing the header files for the above algo libs in *libs\local_voice*, the APIs to drive the AFE algo `VoiceSeeker` and the Common public utilities for `VoiceSeeker` from *[smart HMI]\coffee_machine\cm7* to the example. And do the configurations for the new folder:
   - Uncheck **Exclude resource from build** to enable the folder for being built into the project by right-clicking it and choosing the **Properties** on the pop-up menu.
   - Add the search path for the folder in the project settings as Step 1.

```
"${workspace_loc:/${ProjName}/audio}"
"${workspace_loc:/${ProjName}/audio/RDSP_Includes}"
"${workspace_loc:/${ProjName}/audio/rdsp_utilities_public/include}"
 "${workspace_loc:/${ProjName}/audio/rdsp_utilities_public/
rdsp_memory_utils_public}"
```

3. Copy the folder *local_voice* containing the bin files of DSMT model resources with different languages, and the engines to convert the voices to action commands with the related definitions for DSMT and VIT models from *[smart HMI]\coffee_machine\cm7* to the example. And do the configurations for the new folder:
   - Uncheck **Exclude resource from build** to enable the folder for being built into the project as the above step.
   - Add the search path for the folder in the project settings as the above step.

```
"${workspace_loc:/${ProjName}/local_voice}"
```

Note: *The DSMT model resources are used for the coffee machine application that requires to be modified for the example (to be introduced in Section 6.2.2).*

6.2 Implement speech models

The DSMT and VIT speech models are supported in the example. Both support English and Chinese languages. But note that both models are unable to be supported in one application at the same time. So, it is required to rebuild the project with enabling the model for switching to it. The default language is English on the system startup. Below is the process of implementing both models.

6.2.1 Set up the models

The models resources are designed to contain the wake word **hey NXP** and three pieces of voice commands `registration`, `recognition`, and `preview` in English, and accordingly, the wake word 你好 恩智浦 and voice commands 用户注册, 人脸识别, and 预览 in Chinese.

To set up the DMST and VIT models.

- To use the third party of Tool to generate DSMT model, see the *Smart HMI Software Development User Guide* (document MCU-SMHMI-SDUG).
- To use the NXP online tool to generate VIT model see the *Smart HMI Software Development User Guide* (document MCU-SMHMI-SDUG).

6.2.2 Integrate the DMST and VIT models to the example

As introduced, the folder *local_voice* contains the bin files of DSMT model resources with different languages, and the engines to convert the voices to action commands depending on the app. So, the modifications for the files under the folder depend on the current example.

1. Update the DSMT and VIT model resources in the example with the generated ones by tools:
   - For DSMT, clone and rename the generated Chinese model bin file to *oob_demo_cn_pack_WithMapID.bin* and replace the one under *local_voice\oob_demo_cn* of the example. Same to the generated English model bin file.
     Considering the compatibility and simplification, clear the contents of the files *oob_demo_fr_pack_WithMapID.bin* for French and *oob_demo_de_pack_WithMapID.bin* for Germany used originally in the coffee machine application and are unsupported in the example.
   - For VIT, replace the files *VIT_Model_cn.h* and *VIT_Model_en.h* under *\libs\local_voice\vit\RT1170_Cortex M7\Lib* of the example with the ones generated by VIT online tool.
     Considering the compatibility and simplification, declare an empty array for Germany used in the coffee machine application and comment the old declaration in *VIT_Model_de.h*.
2. Per the example, modify the types of the various actions are defined and the related functions to convert the voice commands to actions for both models in the file *IndexCommands.h* dependent with the coffee machine application:
   - Rename the enum `_coffee_machine_action` and redefine the action types for the voice commands as below:

```
enum _voice_rec_demo_action
{
    kVoiceRecDemoActionReg = 0,
    kVoiceRecDemoActionRec = 1,
    kVoiceRecDemoActionPre = 2,
    kVoiceRecDemoActionInvalid
};
```

   - Delete all the code lines related to the case `ASR_CMD_USER_REGISTER` unsupported in the example in the functions `get_cmd_number()` and `get_action_index_from_keyword()`.
   - Replace the string `COFFEE_MACHINE` (case sensitive) with `VOICE_REC_DEMO` and the `coffee_machine` (case sensitive) with `voice_rec_demo` for the example.
   - Change the `kCoffeeMachineActionInvalid` to `kVoiceRecDemoActionInvalid` in the function `get_action_index_from_keyword()`.
3. Modify the functions about getting the VIT model in *local_voice_model_vit.h* per the example:
   - Replace the string `COFFEE_MACHINE` (case sensitive) with `VOICE_REC_DEMO` for the example.
   - Delete all the code lines related to the case `ASR_CMD_USER_REGISTER` unsupported in the example.
4. Redefine the specific actions for various languages in *IndexCommands_vit.h* using the actions type defined in *IndexCommands.h*:
   - Replace the string `coffee_machine` (case sensitive) with `voice_rec_demo`, the `COFFEE_MACHINE` (case sensitive) with `VOICE_REC_DEMO` and the `CoffeeMachine` with `VoiceRecDemo`.
   - Refine the arrays *action_voice_rec_demo_en* for English and *action_voice_rec_demo_cn* for Chinese, *action_voice_rec_demo_de* for German (unsupported) and *action_voice_rec_demo_fr* for French (unsupported). For example:

```
unsigned int action_voice_rec_demo_en[] = {
  kVoiceRecDemoActionInvalid,    // unknown
  kVoiceRecDemoActionReg,   // "registration"
  kVoiceRecDemoActionRec,   // "recognition"
  kVoiceRecDemoActionPre,   // "preview"
};
unsigned int action_voice_rec_demo_de[] = {
```

```
  kVoiceRecDemoActionInvalid,    // unknown
};
```

Note: *Same modifications to the file IndexCommands_dsmt.h for the DSMT model.*

5. Modify the files *IndexToCommand_cn.h* for Chinese and *IndexToCommand_en.h* for English, *IndexToCommand_de.h* for German and **IndexToCommand_fr.h** for French used for the DSMT model:
   - Redefine the wake word and voice commands per the designed commands (may refer to the commands in the files *ww.txt* and *cmd_voice_rec_example.txt* generated by DMST tool) for English and Chinese, for example about the definition of the wake word for English.

     ```
     char *ww_en[] = {"Hey NXP"};
     ```

     And remove the definitions of all commands for German and French languages unsupported in the example. For example, the definition of the wake word for French is as below.

     ```
     char *ww_fr[] = {};
     ```

   - Replace the string `coffee_machine` with `voice_rec_demo` in all the files.
6. Add the below definition of the command type while reserving the previous in the `enum _asr_inference` in the file *hal_voice_algo_asr_local.h* considering the compatibility.

   ```
   ASR_CMD_VOICE_REC_DEMO = (1U << 1U)
   ```

6.3 Update and enable voice related HALs

To drive the voice recognition, the below HAL drivers are involved: PDM microphone HAL implemented in *hal_input_pdm_mic.c*, audio process AFE HAL in *hal_audio_processing_afe.c*, DSMT voice algo HAL in *hal_voice_algo_dsmt_asr.c* and VIT voice algo HAL in *hal_voice_algo_vit_asr.c*.

The functions implemented in the HALs are common to different applications. Few modifications are required but do some configurations for the example:

1. No modification is required to the file *hal_input_pdm_mic.c* for PDM microphone HAL driver but add the definition to enable it in the *board_define.h*.

   ```
   #define ENABLE_INPUT_DEV_PdmMic
   ```

2. Do updates for audio process AFE HAL driver:
   - Declare the global variable `g_MQSPlaying` in the file *hal_audio_processing_afe.c* as it is declared in audio playing related HALs, which is unsupported in the example and set the variable to false as default. See below.

     ```
     #ifdef ENABLE_OUTPUT_DEV_MqsAudio || ENABLE_OUTPUT_DEV_MqsStreamerAudio
     extern volatile bool g_MQSPlaying;
     #else
     volatile bool g_MQSPlaying = false;
     #endif
     ```

   - Add the definition of the memory section for the buffer used in the file *hal_audio_processing_afe.c* in the file *board_define.h*.

     ```
     #define AT_NONCACHEABLE_SECTION_ALIGN_DTC(var, alignbytes) \
     __attribute__((section(".bss.$SRAM_DTC_cm7,\"aw\",%nobits @"))) var
      __attribute__((aligned(alignbytes)))
     ```

- Add the definition to enable audio processing HAL driver in the *board_define.h*.

```
#define ENABLE_AUDIO_PROCESSING_DEV_Afe
```

3. Update for DSMT and VIT voice algo HAL drivers.
   - Add the definition to use the current supported application as below.

```
#if ENABLE_VOICE_REC_DEMO
#define CURRENT_DEMO ASR_CMD_VOICE_REC_DEMO
```

   - Add the setup for the current application in the function *voice_algo_dev_input_notify()* in the file *hal_voice_algo_dsmt_asr.c*.

```
#if ENABLE_VOICE_REC_DEMO
s_AsrEngine.voiceConfig.demo = ASR_CMD_VOICE_REC_DEMO;
```

   - Redefine the active languages for the DSMT model in the file *hal_voice_algo_asr_local.h* as only two languages are supported while four are set for the DSMT model in the example.

```
#ifdef ENABLE_DSMT_ASR
#define DEFAULT_ACTIVE_LANGUAGE (ASR_ENGLISH | ASR_CHINESE)
```

   - To configure DSMT and VIT ASR and AFE HALs, add some definitions in the file *board_define.h*.
     - Add the definitions to enable both algos HALs (currently enable DSMT ASR as default):

```
#define ENABLE_DSMT_ASR
//#define ENABLE_VIT_AS
```

     - Add the definitions to configure some features for DSMT, VIT, and AFE under the control of the above enablement. For example, max wake word length.

```
#elif defined(ENABLE_DSMT_ASR)
/* "Hey NXP" and its corresponding translations in other languages may
 take up to 3s to be spoken. */
#define WAKE_WORD_MAX_LENGTH_MS 3000
```

     - Configure the SRAM memory section used for DSMT and VIT ASR algo HAL.

```
#define AT_CACHEABLE_SECTION_ALIGN_OCRAM(var, alignbytes) \
__attribute__((section(".bss.$SRAM_OC1,\"aw\",%nobits @"))) var
 __attribute__((aligned(alignbytes)))
```

   - Update the memory setting on **Project** > **Properties** > **C/C++ Build** > **MCU Settings** to enlarge the size to `0x100000` for **SRAM_OC1** used above and delete the setting for **SRAM_OC2** as its space is merged to the **SRAM_OC1**.
   - Add the search path for the above voice HALs in the project settings.

```
"${workspace_loc:/${ProjName}/framework/hal/voice}"
```

6.4 Add and update output UI HAL

The output UI HAL depends on the LVGL GUI app. It notifies the events to the voice algo HAL and responds to the inference results from the voice algo HAL. With GUI app, the events are generally triggered by the GUI and the results are shown on the GUI.

To enable the HAL, clone the existed similar HAL driver file where generally the below functions are implemented:

1. Face algo trigger and result handling with the features: progress bar, face indicator (preview mode).
2. Audio playing with some languages.
3. Coffee machine application related including GUI.

4. Voice algo trigger and result handling with some languages.
5. Standby and wake-up mechanism with a session timer.
6. The callbacks are the APIs for the GUI application callings to communicate to the output UI HAL.
7. The calls of the APIs from the LVGL GUI application to communicate to the LVGL GUI application.

The major works to implement the HAL for the example are:

- Clone the existed similar HAL driver file and change the related names.
- Remove the codes related to 1, 2, and 3.
- Keep and update the above 4, 5, 6, and 7.

The main steps are as below:

1. Clone *hal_output_ui_coffee_machine.c*. Change the file name to *hal_output_ui_ voice_rec.c* (The below updates are all for the file).
2. Replace all strings `CoffeeMachine` to `VoiceRecDemo` in the file.
3. Remove the codes related to vision (face rec), audio containing the string `prompt` and face indicators including icons and progress related. For example:

```
#include "hal_vision_algo.h", #include "hal_event_descriptor_face_rec.h"
```

4. Remove the codes containing GUI related to the coffee machine app. For example, the variables involved with the coffee type registration: `s_IsWaitingAnotherSelection` and `s_IsWaitingRegisterSelection`.
5. Change the setting for the member `.attr.pSurfac` from `&s_UiSurface` to `NULL` in the structure `s_OutputDev_UiVoiceRecDemo` as it is used for face recognition.
6. Update the voice command type in the `enum` type for the example.

```
enum
{
    VOICE_CMD_REG = 0,
    VOICE_CMD_REC = 1,
    VOICE_CMD_PRE = 2,
    VOICE_CMD_INVALID
};
```

7. Change the setting for the member **eventInfo** from `kEventInfo_Remote` to `kEventInfo_Local` in the output event structure in the function `_SetVoiceModel()` and `_StopVoiceCmd()` since only the single core – cm7 is used.
8. Update the function `_InferComplete_Voice()` to take the corresponding actions for the inference results from voice recognition:
   - Keep the cases of the screen IDs `kScreen_Home` and `kScreen_Standby` for home and standby screens while removing others.
   - Add three voice command types of `VOICE_CMD_REG`, `VOICE_CMD_REC` and `VOICE_CMD_PRE` defined in the above step as three cases of voice inference results while removing the old ones in the case `kScreen_Home`.
   - Add the calls of the API `gui_show_voice_rec_action()` to show the voice interference results on the GUI screen. The API provided by the LVGL GUI application is implemented in *custom.c* (introduced in Section 7).
9. Change the macro definition `ASR_CMD_COFFEE_MACHINE` to `ASR_CMD_VOICE_REC_DEMO` in the function `WakeUp()`.
10. Update UI callback functions to handle the events triggered from GUI.
    - Update the function `UI_EnterScreen_Callback()` used for initializations, such as session timer setup, voice model setup when entering a screen as following points.
      – Keep the cases of the screen IDs `kScreen_Home` and `kScreen_Standby` for home and standby screens while removing others.

- Remove the unneeded function `UI_Finished_Callback()`.
- Add a function `UI_GetLanguage_Callback()` to convert the current recognized language index to the language index, which is provided for GUI application.

11. Continue to use the APIs from GUI application: `get_current_screen()`, `gui_set_home()`, and `gui_set_standby()` (required to be implemented in *custom.c* and introduced in Section 7) while removing others.
12. Add the code line `#include "custom.h"` to use the GUI related APIs.
13. Add the below definitions to enable UI output HAL in *board_define.h*.

```
#define ENABLE_OUTPUT_DEV_UiVoiceRecDemo
```

Note: *May do some optimizations for the output UI HAL per your application design as the HAL is much dependent with the application.*

6.5 Update display HAL

The camera preview is enabled anytime in the file *hal_display_lvgl_camerapreview.c* for display HAL. However, the camera preview is required on the home screen but not for the standby screen in the example. It is required to disable camera preview on the standby screen in the function `HAL_DisplayDev_LVGLCameraPreview_Blit()`.

7 Implement an LVGL GUI application

The development of an LVGL GUI application based on framework is mainly calling the APIs from output UI HAL and providing the APIs to output UI HAL.

However, the detailed implementation of an LVGL GUI application depends on the requirements and design of the application. The GUI application in this example is designed as described at the beginning of Section 4.

The below is the implementation introduction for reference.

1. The customized codes are implemented in the files *custom.c* and *custom.h* given by GUI Guider as the interface between the GUI Guider project and the embedded system project.
   - Implement the functions `get_current_screen()`, `gui_set_standby()` and `gui_set_home()` and related sub functions, such as `gui_setup_screen()` to set up the home and standby screen in the file *custom.c* (May refer to the ones in coffee machine application).
   - Implement the function `gui_show_voice_rec_action()` in *custom.c* to show the results on GUI screen as the actions of response to the events of the buttons and the voice commands.
   - Implement multi-language support for GUI application in the file *custom.c*:
     – Add the functions `gui_standby_set_language_UI()` and `gui_home_set_language_UI()` to display all the texts on the standby and home screens with the selected language. Accordingly, define all the strings of texts in English and Chinese languages. For example, the title on the home screen is defined as below.

```
static const char *s_HomeTitleStr[kLanguage_Ids][1] = {
     {"Camera preview",},
     {"相机预览",},
};
```

     – Add the functions `gui_standby_language_changed_cb()` and `gui_home_language_changed_cb()` to respond to the triggered events of the language selection by clicking the dropdown widget on the standby and home screen. They are called in the file *event.c* for events handling.
   - The UI callback functions and the function `WakeUp()` implemented in output UI HAL are dependent with the embedded platform. Meanwhile, they are also required to be called in GUI Guider project to run on

the simulator, which is independent with the embedded platform. So, it is required to do some updates for being compatible with the embedded platform and the GUI Guider simulator.

- Implement another set of the functions (may be simpler) with the same prototypes under the control of the macro definition `#ifdef LV_USE_GUIDER_SIMULATOR` in *custom.c* for the compatibility. For example, the function `UI_EnterScreen_Callback()` can be implemented simply as eblow.

```
#if LV_USE_GUIDER_SIMULATOR
uint8_t UI_EnterScreen_Callback(screen_t screenId)
{
  return 1;
}
```

- The above macro definition `LV_USE_GUIDER_SIMULATOR` is originally defined and enabled to 1 in the file *lv_conf.h* in the folder lvgl-simulator of GUI Guider. Copy the file to the folder source of the example software and set the definition to 0 for disabling it on the embedded platform.

- Update the file *custom.h* to declare the global types and functions:
  - Define the enum types about voice recognition action index, language ID, and screen ID.
  - Move the enum type `_wake_up_source` from output UI HAL to *custom.h* as the button type defined in the enum is used by the file *event.c* as well.
  - Declare all the global functions like `UI_xxx_Callback()` and `gui_xxx()`.

2. Develop the GUI on GUI Guider:
   - Clone the folder camera preview containing the GUI Guider project software in the folder `gui_guider` in the base software package `lvgl_gui_camera_preview_cm7`. Change the related name `camera_preview` to `voice_rec` for the new example.
   - Copy the above updated files custom.c and custom.h to the new GUI Guider project software.
   - Open the new `voice_rec` project on GUI Guider. Update per the design introduced at the beginning of Section 4. And add the various events handling as below:
     - Implement custom C codes for the various events handling on GUI Guider using the APIs, such as `WakeUp()` to enter the home screen when clicking on the standby screen.
3. Update the generated codes from GUI Guider to MCUXpresso project.
   - Replace the *.c* and *.h* files in the folder generated except for the ones in the folder images with the corresponding ones in the folder generated of GUI Guider project software.
4. Add the GUI image support.
   The GUI images in the example are unchanged with the cloned GUI app. However, the name and descriptor of the images are changed with the new version of GUI Guider used in the example. So, some updates are required based on the cloned one:
   - Clone the image resource folder *resource_lvgl_gui_camera_preview* and the tool folder *resource_build* from the base software package *lvgl_gui_camera_preview_cm7* to the example software. Then delete the generated files *resource_information_table.txt* and *camera_preview_resource.bin*.
   - Change all the string *camera_preview* in the name and the contents of the remaining files to *voice_rec* for the new example.
   - Copy the image files *_NxpVoiceRec_alpha_185x55.c* and *_NxpFaceRec_alpha_185x55.c* from the path of *generated\images* in GUI Guider project software to the path of *resource_lvgl_gui_voice_rec\images*. Remove the old files *_NxpVoiceRec_185x55.c* and *_NxpFaceRec_185x55.c*.
   - Accordingly, change the names to *_NxpVoiceRec_alpha_185x55.c* and *_NxpFaceRec_alpha_185x55.c* in the file *voice_rec_resource.txt*.
   - Build the image resource by double clicking to execute the script file *voice_rec_resource_build.bat* in windows. The binary file *voice_rec_resource.bin* and the information text file *resource_information_table.txt* are generated.
   - The size and address information of the image resource are not required to be updated to the example software since the image data is unchanged. So, only update the image descriptor arrays

_NxpFaceRec_alpha_185x55 and *_NxpVoiceRec_alpha_185x55* in the file *setup_images.c* using the ones in the files *_NxpVoiceRec_alpha_185x55.c* and *_NxpFaceRec_alpha_185x55.c*.

8 Add application level support

The codes and configurations at the application level are in the folder *source*. The main updates are usually in the main file **lvgl_gui_voice_rec_cm7.cpp** containing the hardware board initializations and the framework setup.

8.1 Update the board initialization

To update the board initialication, perform the following steps:

1. Add the call of the function `BOARD_ConfigMPU()` in `APP_BoardInit()` to configure memory MPU.

8.2 Update the framework setup

To set up the enabled voice algo HALs and UI output HAL and their managers on framework, following the conversions of development as below:

1. Include the header file related to the managers of the enabled HALs as below:

```
#include "fwk_input_manager.h"
#include "fwk_audio_processing.h"
#include "fwk_voice_algo_manager.h"
#include "fwk_output_manager.h"
```

2. Declare the enabled HAL devices:

```
HAL_INPUT_DEV_DECLARE(PdmMic);
HAL_AUDIO_PROCESSING_DEV_DECLARE(Afe);
HAL_VOICEALGO_DEV_DECLARE(Asr);
HAL_VOICEALGO_DEV_DECLARE(Asr_VIT);
HAL_OUTPUT_DEV_DECLARE(UiVoiceRecDemo);
```

3. Register the enabled HAL devices in the function `APP_RegisterHalDevices()`:

```
  HAL_INPUT_DEV_REGISTER(PdmMic, ret);
  HAL_AUDIO_PROCESSING_DEV_REGISTER(Afe, ret);
#ifdef ENABLE_DSMT_ASR
  HAL_VOICEALGO_DEV_REGISTER(Asr, ret);
#elif defined(ENABLE_VIT_ASR)
  HAL_VOICEALGO_DEV_REGISTER(Asr_VIT, ret)
#endif /* ENABLE_DSMT_ASR */
  HAL_OUTPUT_DEV_REGISTER(UiVoiceRecDemo, ret);
```

4. Initialize the related managers in the function `APP_InitFramework()`:

```
FWK_MANAGER_INIT(InputManager, ret);
FWK_MANAGER_INIT(AudioProcessing, ret);
FWK_MANAGER_INIT(VoiceAlgoManager, ret);
FWK_MANAGER_INIT(OutputManager, ret);
```

5. Start the related managers in the function **APP_StartFramework()**:

```
FWK_MANAGER_START(InputManager, INPUT_MANAGER_TASK_PRIORITY, ret);
FWK_MANAGER_START(AudioProcessing, AUDIO_PROCESSING_TASK_PRIORITY, ret);
FWK_MANAGER_START(VoiceAlgoManager, VOICE_ALGO_MANAGER_TASK_PRIORITY, ret);
FWK_MANAGER_START(OutputManager, OUTPUT_MANAGER_TASK_PRIORITY, ret);
```

6. Define the priorities of the manager tasks in the above step.

```
#define INPUT_MANAGER_TASK_PRIORITY          2
#define AUDIO_PROCESSING_TASK_PRIORITY       2
#define VOICE_ALGO_MANAGER_TASK_PRIORITY     3
#define OUTPUT_MANAGER_TASK_PRIORITY         1
```

Note: *For more details about all the modifications introduced above, see the attached example software at https://mcuxpresso.nxp.com/appcodehub.*

9 Verifications with the example project

Visit https://mcuxpresso.nxp.com/appcodehub and get the example software package containing the resources and tools for this application note.

Open the example project on MCUXpresso IDE. Build and program the *.axf* file to the address `0x30000000` and program the resource bin file *voice_rec_resource.bin* to the address `0x30800000`.

The LVGL GUI voice recognition example works normally as below:

- **Standby screen**
  With power up, the standby screen is displayed as Figure 3. Say the wake word or touch the screen to enter the home screen per the hints displayed in the selected language, which can be changed via the dropdown widget on the screen.
- **Home screen**
  After entering the home screen, the video streams captured by camera shows on the specific area of camera preview on the GUI screen. As the initial state, the preview state is displayed on the status label and the supported voice commands are hinted. They are shown in the current language, which can be changed via the dropdown widget. See Figure 4.
  - **Preview**: Every time clicking the area outside the buttons and images， or say the voice command `preview` or 预览 depending on the current selected language, the corresponding text **Preview…** or 预览… displays on the status label.
  - **Registration**: Every time clicking the button **Registration**, or say the voice command `registration` or 用户注册 depending on the current selected language, the corresponding text **Registration…** or 注册… displays on the status label.
  - **Recognition**: Every time clicking the button **Recognition**，or say the voice command `Recognition` or 人脸识别 depending on the current selected language, the corresponding text **Recognition…** or 识别… displays on the status label.

  When the time (60 s) is up on the home screen, it returns to the standby screen automatically.

10 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

11 Revision history

Table 1 summarizes the revisions to this document.

Table 1. Revision history

Document ID	Release date	Description
AN14250 v.1	26 March 2024	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

For more information, please visit: https://www.nxp.com

Date of release: 26 March 2024
Document identifier: AN14250