AN14263

Implement LVGL GUI Face Recognition on Framework

Rev. 1 — 19 April 2024 Application note

Document information

Information	Content
Keywords	Face recognition, LVGL GUI, Framework
Abstract	This application note describes how to enable AI&ML vision algorithm model for face recognition on framework to implement face recognition function with a simple LVGL GUI example on SLN-TLHMI-IOT board.

1 Overview

NXP has launched a solution development kit named SLN-TLHMI-IOT that 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 that supports flexible designs and customization of vision and voice functions. To help 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 the web page of the NXP EdgeReady Smart HMI Solution Based on i.MX RT117H with ML Vision, Voice, and Graphical UI.

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 on the framework step by step. This application note is based on *Implement LVGL GUI Camera Preview on Framework* (document AN14147).

This application note describes how to enable the AI&ML vision algorithm model for face recognition on the framework to implement the face recognition function via camera preview on the GUI screen with a simple LVGL GUI example on the SLN-TLHMI-IOT board.

In the application note, the example presents an LVGL GUI screen with a camera preview and some buttons to trigger face registration, recognition, and removal. The registered face data is stored on Flash via a little file system.

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

- Enable the face recognition feature on the framework.
- Add face database support on the framework via file system on Flash.
- Implement the LVGL GUI app.

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

- Understand the framework and the smart HMI solution software more deeply.
- Develop their AI&ML face recognition on framework with the LVGL GUI app.

1.1 Framework overview

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

- Device managers – the 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. With a device manager starting after the devices being registered to it, it waits and checks a message to transfer data to the devices and other managers after initializing and starting the registered devices.

The HAL devices are written on top of the 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 received the event communicates that event to its manager, then in turn it notifies other managers designated to receive the event.

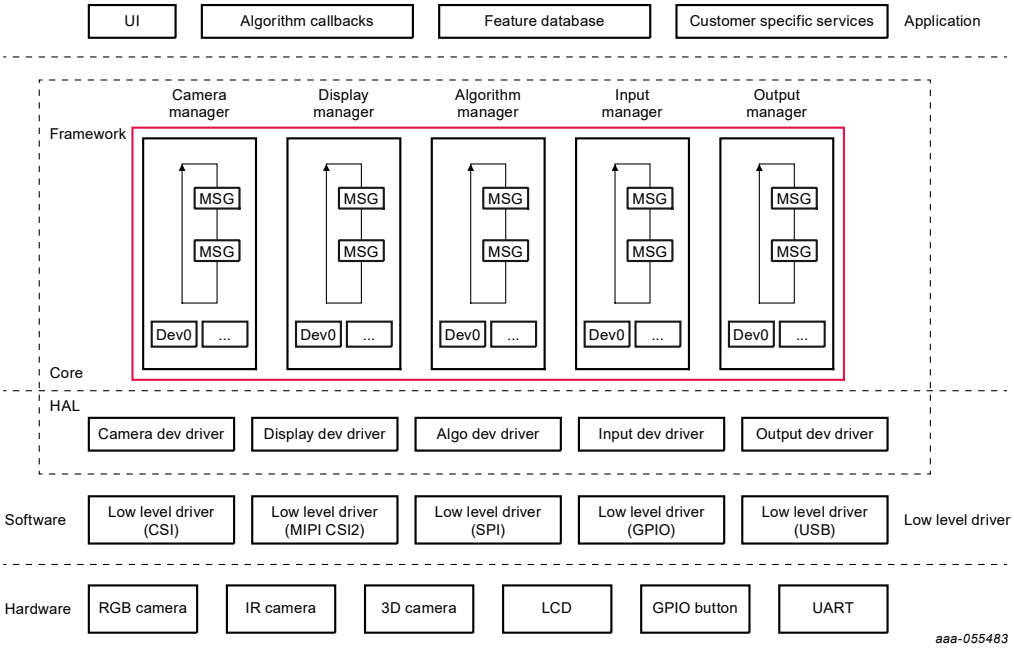


Figure 1. Smart HMI SW architecture on framework

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

1. Ease-of-use
2. Flexibility/Portability
3. 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 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 *Smart HMI Software Development User Guide* (document MCU-SMHMI-SDUG).

1.2 Light and Versatile Graphics Library (LVGL)

LVGL (Light and Versatile Graphics Library) is a free and open-source graphics library providing everything that you need to create an embedded GUI with easy-to-use graphical elements, beautiful visual effects and low memory footprint.

1.3 GUI Guider

GUI Guider is a user-friendly graphical user interface development tool from NXP that enables rapid development of high quality displays with the open-source LVGL graphics library. GUI Guider's drag-and-drop editor makes it easy to use 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 NXP's general purpose and crossover MCUs and includes built-in project templates for several supported platforms.

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

2 Development environment

First, 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 (the SLN_TLHMI_IOT kit)
- SEGGER J-Link with a 9-pin Cortex-M adapter and V7.84a or a newer version of the 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 the second application note as the basis software of the development. For details, see: https://mcuxpresso.nxp.com/appcodehub.
- RT1170 SDK V2.13.0 – as the code resource for the development.
- SLN-TLHMI-IOT software V1.1.2 – smart HMI source code released on the NXP GitHub repository as the code resource for the development. For details, see: GitHub - NXP/mcu-smhmi at v1.1.2

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

3 Vision architecture on framework

The vision architecture on the framework is shown in Figure 2. The vision algo HAL (`OASIS_HAL`) has the below processes:

- Do face registration and recognition through the AI&ML vision algorithm model after receiving the related events from the output UI HAL. Notify the inference results from the algorithm model to the output UI HAL.
- Accesses (add, delete…) the face feature database based on the little file system by calling the APIs of FaceDB HAL after receiving the related events from the output UI HAL.
- Request the camera video frame from the camera HAL when doing face registration and recognition.

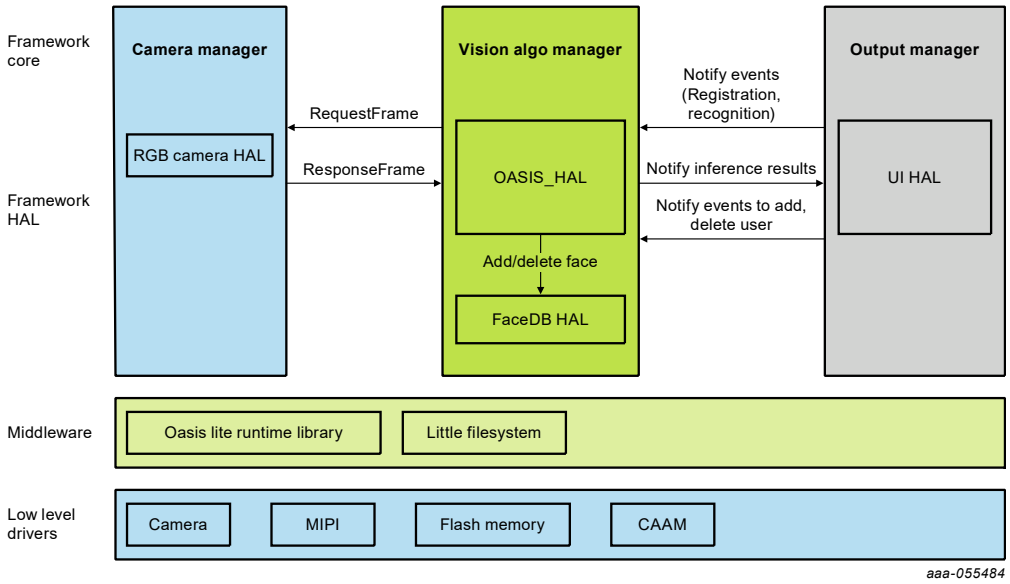


Figure 2. Vision architecture

4 Implement face recognition on framework

The LVGL GUI face recognition example (the example is provided later) on the framework is implemented based on the example codes of *Implement LVGL GUI Camera Preview on Framework* (document AN14147).

For demonstrating the face recognition in the example, the basic function of the GUI app (see the main screen in Figure 3) is designed as described below:

- The GUI app triggers the face registration or recognition event to the output UI HAL when clicking the button **Registration** or **Recognition**. And the output UI HAL notifies the event of adding a user to the vision algo HAL after the face registration is successful.
- The GUI app triggers the event of deleting a user to the output UI HAL when clicking the button **Delete User** after the face of the user is recognized.
- The GUI app triggers the event of stopping the oasis algo running to the output UI HAL when clicking the screen outside the buttons and images.

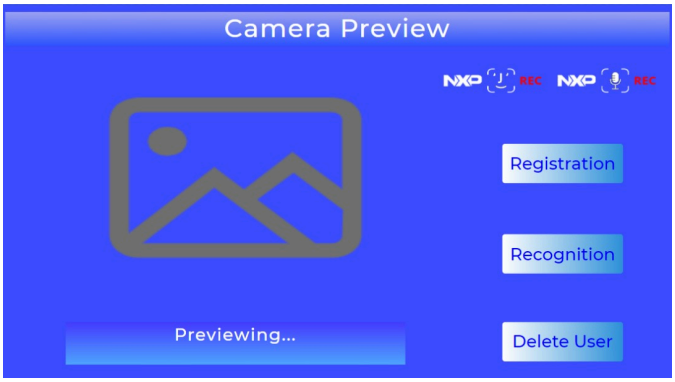


Figure 3. LVGL GUI face recognition app main screen

SW Preparations

Prepare the software package for the implementation of the example.

- Clone the base software `lvgl_gui_camera_preview_cm7`. Change the project name and the main filename to `lvgl_gui_face_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. The operations are shown in Figure 4:

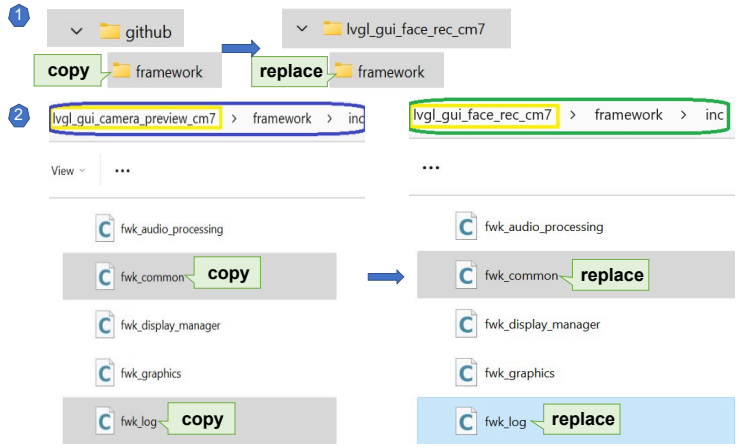


Figure 4. Update framework

- 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 code of the core is provided.

4.1 Enable face recognition feature on framework

The face recognition feature is built on the ML vision algorithm model provided as a static library – an oasis lite runtime library by NXP. The library is a tiny, highly efficient, customized, and optimized AI library. The model includes face detection, face recognition, glass detection, and liveness detection.

It mainly provides the `API OASISLT_run_extended()` to run the face recognition pipeline while updating results to the caller through event callbacks, and add/update/delete faces in the database through the face database callbacks after specifying the source frame information, callbacks, and memory pool used by the library by calling another `API OASISLT_init()` on initialization.

The calling of the APIs and the callback functions are implemented in the vision algo HAL of the framework.

4.1.1 Add vision algo model library

1. Copy folder `oasis` containing the library and the related header file from smart `HMI\coffee_machine\cm7\libs\` into the folder `libs` of the example SW.
2. Add the search path of the header file in `Project > Properties > C/C++ Build > settings > Tool Settings > MCU C compiler > Includes` and `MCU C++ compiler > Includes`:

```
"${workspace_loc:/${ProjName}/libs/oasis/include}"
```

3. Add the lib and its search path on `Project > Properties > C/C++ Build > settings > MCU C++ Linker > Libraries:`

```
liboasis_lite2D_DEFAULT_117f_ae.a
"${workspace_loc:/${ProjName}/libs/oasis}"
```

and the macro definition to enable the feature on `Project > Properties > C/C++ Build > settings > Tool Settings > MCU C compiler > Preprocessor` and `MCU C++ compiler > Preprocessor:`

```
SMART_TLHMI_2D
```

4.1.2 Enable vision algo HAL

The vision algo HAL drives the vision algo model to work and responds the results to the UI output HAL after receiving the events from it.

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

- Implement the callbacks of face database operations and events handling.
- Drive the vision algo to work by calling the APIs of the oasis library.
- Access user face database and app database (it is not needed in the example).
- Receive events from and send results to output UI HAL.

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 the app data operations.
- Update the definitions and functions for handling the events from the output UI HAL per the example design.
- Add the configurations required in oasis initialization.

The detailed steps are as below:

1. Clone `hal_vision_algo_oasis_coffeemachine.c`. Change the filename to `hal_vision_algo_oasis_guifacerec.c`. And replace all strings `CoffeeMachine` with `GUIFaceRec` in the file.
2. Remove the codes containing the string `coffeedb` (not case sensitive) related to the app database, for example, `#include hal_sln_coffeedb.h`.
3. Modify the function `HAL_VisionAlgoDev_OasisGUIFaceRec_InputNotify()` for handling the events from the output UI HAL.
   - Change the event definition `kEventFaceRecId_RegisterCoffeeSelection` to `kEventFaceRecId_RegisterUserFace` and the structure string `regCoffeeSelection` to `regGUIFaceRec` for the event handling to add new face feature data to the database.
   - To show the standard process of the face recognition actions in the example, modify the handling in the case of `kEventFaceRecID_OasisSetState` with the definitions of the states:

```
kOASISLiteState_Registration
kOASISLiteState_Recognition
kOASISLiteState_Stopped
```

4. Add and modify the definitions of the events mentioned in the above step.
   - Copy the header file `smart_tlhmi_event_descriptor.h` from `smart HMI\coffee_machine\cm7\source\event_handlers\` into the folder `source` of the example SW. Update the file as below:
   - Change the event definition `kEventFaceRecId_RegisterCoffeeSelection` to `kEventFaceRecId_RegisterUserFace` in the enum type `_event_smart_tlhmi_id` and the structure string `regCoffeeSelection` to `regGUIFaceRec` in the struct `_event_smart_tlhmi`.

So, change the struct `register_coffee_selection_event_t` for `regCoffeeSelection` to `register_gui_facerec_event_t`.

- Delete the else contents used for the coffee machine app, for example, the code line about voice: `#include "hal_event_descriptor_voice.h"`.
- Add the types `kOASISLiteState_Stopped` and `kOASISLiteState_Running` to the enum type `oasis_lite_state_t` in `hal_vision_algo.h` under `framework>hal>vision` in the project as below:

```
typedef enum _oasis_lite_state
{
    kOASISLiteState_Running,
    kOASISLiteState_Stopped,
    kOASISLiteState_Recognition,
    kOASISLiteState_Registration,
    kOASISLiteState_DeRegistration,
    kOASISLiteState_RemoteRegistration,
    kOASISLiteState_Count
} oasis_lite_state_t;
```

- Use the above updated struct `oasis_lite_state_t` to refine the struct `oasis_state_event_t` in `hal_event_descriptor_face_rec.h` under `framework>hal>vision` in the project as below:

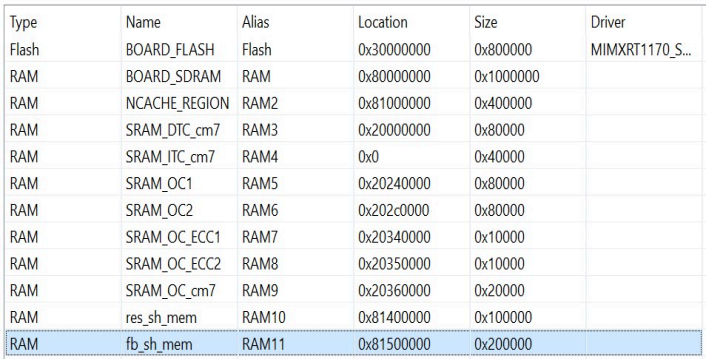
```
typedef struct _oasis_state_event_t
{
    oasis_lite_state_t state;
} oasis_state_event_t;
```

5. Change all `kEventInfo_Remote` to `kEventInfo_Local` for sending events from the vision algo HAL to other HALs running on the same core as single core instead of dual-core is used in the example.
6. Add and modify the below configurations for oasis initialization in `OASISLT_init()`:
   - Add the macro definitions and memory sections for the video frame in `board_define.h`:

```
#define OASIS_RGB_FRAME_WIDTH      800
#define OASIS_RGB_FRAME_HEIGHT    600
#define OASIS_RGB_FRAME_SRC_FORMAT kPixelFormat_YUV1P444_RGB
#define OASIS_RGB_FRAME_BYTE_PER_PIXEL 3

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

   - Config the memory assignment to the above memory section `fb_sh_mem` on `Project > Properties > C/C++ Build > MCU Settings` shown in Figure 5:



Type	Name	Alias	Location	Size	Driver
Flash	BOARD_FLASH	Flash	0x30000000	0x800000	MIMXRT1170_S...
RAM	BOARD_SDRAM	RAM	0x80000000	0x1000000	
RAM	NCACHE_REGION	RAM2	0x81000000	0x400000	
RAM	SRAM_DTC_cm7	RAM3	0x20000000	0x80000	
RAM	SRAM_ITC_cm7	RAM4	0x0	0x40000	
RAM	SRAM_OC1	RAM5	0x20240000	0x80000	
RAM	SRAM_OC2	RAM6	0x202c0000	0x80000	
RAM	SRAM_OC_ECC1	RAM7	0x20340000	0x10000	
RAM	SRAM_OC_ECC2	RAM8	0x20350000	0x10000	
RAM	SRAM_OC_cm7	RAM9	0x20360000	0x20000	
RAM	res_sh_mem	RAM10	0x81400000	0x100000	
RAM	fb_sh_mem	RAM11	0x81500000	0x200000	

Figure 5. Memory assignment to the section `fb_sh_mem`

- Declare the global variable `g_DTCOPBuf` in `lvgl_gui_face_rec_cm7.cpp`:

```
AT_NONCACHEABLE_SECTION_ALIGN_DTC
(uint8_t g_DTCOPBuf[DTC_OPTIMIZE_BUFFER_SIZE], 4);
```

- Continue to add the definitions used in the above variable:
- Define the above section in `board_define.h`:

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

- Include the header file `hal_vision_algo.h` containing the macro definition `DTC_OPTIMIZE_BUFFER_SIZE` in `app_config.h` included in `lvgl_gui_face_rec_cm7.cpp`.

7. Set the variable `s_debugOption` to true for showing the progress status on face recognition.
8. Add the search path of the header files of the vision HAL on `Project > Properties >C/C++ Build > settings > Tool Settings > MCU C compiler > Includes` and `MCU C++ compiler > Includes`:

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

9. Add the below definition to enable vision algo HAL in `board_define.h`:

```
#define ENABLE_VISIONALGO_DEV_Oasis_GUIFaceRec
```

4.1.3 Enable output UI HAL

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

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

- Notify the events for face recognition and database access.
- Implement the callbacks for the GUI app to trigger the events.
- Handle the inference results from the vision algo module.
- Show the process and results of the events handling on the UI by the progress bar controlled with the timers and face guide rectangle.

The major works to implement the HAL for the example used in this document are:

- Clone the existed similar HAL driver file and change the related names.
- Remove the codes related to the app.

- Update the functions for the events notification and results response per the example design.
- Add the callbacks for the GUI app to trigger the events.

The detailed steps are as below:

1. Clone `hal_output_ui_coffee_machine.c`. Change the filename to `hal_output_ui_guifacerec.c`.
2. Replace all strings `CoffeeMachine` with `GUIFaceRec` in the file.
3. Remove the codes related to the app – coffee machine.
   - Remove the functions `WakeUp()` and `_StandBy()` and the related codes (may search the string `wake_up` and `standby` for them).
   - Remove preview mode events handling related codes in `HAL_OutputDev_UiGUIFaceRec_InputNotify()`.
   - Remove the functions `UI_xxx_Callback()` and the codes containing the string `gui_` and screen related to the GUI of the coffee machine except for `gui_set_virtual_face()` for the preview mode feature.
   - Remove all codes involved with the variables `s_IsWaitingAnotherSelection` and `s_IsWaitingRegisterSelection` related to the coffee machine app.
   - Remove the codes related to voice, audio, and language. For example:

```
#include "hal_voice_algo_asr_local.h",
#include "hal_event_descriptor_voice.h"
```

4. For the various events notification, implement the new functions `_OutputManagerNotify()`, `_SetFaceRec()`, `_RegisterGUIFaceRec()`, and `DeregisterGUIFaceRec()` referring to the functions `_StopFaceRec()`, `_RegisterCoffeeSelection()`, and `DeregisterCoffeeSelection()` before deleting them.
   - The `_OutputManagerNotify()` implements the basic event output function to send an event to the vision algo HAL. The below functions call it to send their own events.
   - The `_SetFaceRec()` sends the event `kEventFaceRecID_OasisSetState` to trigger the vision algo for face registration, recognition, and stop the algo.
   - The `_RegisterGUIFaceRec()` sends the event `kEventFaceRecId_RegisterGUIFaceRec` that is defined in `smart_tlhmi_event_descriptor.h` to add face feature data to the database when registration OK.
   - The `DeregisterGUIFaceRec()` sends the event `kEventFaceRecID_DelUser` to delete the face feature data from the database when passing the face recognition.
5. Update the codes to take the corresponding actions including refresh the GUI by calling the APIs from the LVGL GUI app for the inference results of face registration and recognition in the function `_InferComplete_Vision()` per the example's design. For example, when face registration is successful,
   - Stop showing the progress by calling `_FaceRecProcess_Stop();`
   - Stop the face registration by calling `_SetFaceRec(kOASISLiteState_Stopped);`
   - Show the successful result on the GUI: `gui_show_face_rec_result(kFaceRecResult_OK, s_UserId);`
   - Register the face data to the database: `_RegisterUserFace(s_UserId);`
6. Add UI callback functions to handle the events: preview, face registration, recognition, and deleting user triggered from the GUI. For example, the face registration callback:

```
void UI_Registration_Callback()
{
 _SetFaceRec(kOASISLiteState_Registration);
 _FaceRecProcess_Start();
```

```
}
```

- And add the functions `_FaceRecProcess_Start()` and `_FaceRecProcess_Stop()` to show the progress and status in the different events and results.
- Update the timer ISR callback function `_SessionTimer_Callback()` to handle the case of time-out by calling:

```
gui_show_face_rec_result(kFaceRecResult_TimeOut,
                         s_UserId);
```

7. Add the below definitions to enable UI output HAL in `board_define.h`:

```
#define ENABLE_OUTPUT_DEV_UiGUIFaceRec
```

Notice:

To present the face recognition feature better, keep the function to show the process and results of face recognition in the output UI HAL. The function is described as below:

- *The face guide rectangle shows blue, and the progress bar shows the progress when starting the face registration or recognition.*
- *The face guide rectangle shows red when face registration is successful.*
- *The face guide rectangle shows green when face recognition is successful.*
- *The face guide rectangle keeps blue, and the progress bar shows full progress when the action is unsuccessful after the timer expiration. At that point, stop the face registration or recognition.*

The progress bar and face guide rectangle are presented as the icons that are built into the resource binary file to be programmed into Flash. The pointers to the icons data on SDRAM are set up in the function `LoadIcons(APP_ICONS_BASE)` called on the output UI HAL device initialization in the output UI HAL. It must implement the icons support for the function.

4.1.4 Implement the icons support

1. Build the resource combining the icons with the images used in the LVGL GUI app:
   - Clone the four icon header files `process_bar_240x14.h`, `virtual_face_blue_420x426.h`, `virtual_face_green_420x426.h`, and `virtual_face_red_420x426.h` from `smart HMI \coffee machine\resource\icons\` to the new folder `icons` under the `resource` folder of the example SW.
   - Add the search path for the four icon files in the `camera_preview_resource.txt` file in the `resource` folder, for example: `icon ../resource/icons/process_bar_240x14.h`
   - Execute `camera_preview_resource_build.bat` to build the images and icons resources to generate the bin file `camera_preview_resource.bin` and the info file `resource_information_table.txt` (See Figure 6).

```
_NxpFaceRec_alpha_185x55.data = (base + 0);
_NxpVoiceRec_alpha_185x55.data = (base + 30528);
s_Icons[process_bar_240x14] = (base + 0);
s_Icons[virtual_face_blue_420x426] = (base + 6720);
s_Icons[virtual_face_green_420x426] = (base + 364608);
s_Icons[virtual_face_red_420x426] = (base + 722496);

Images Total: 0x00ee80, 61056

Icons Total: 0x107c40, 1080384
```

Figure 6. The info generated in resource_information_table.txt

2. Define the start address on SDRAM and the size of the icons in `app_config.h`. The address starts next to the images of the GUI app. The size is generated in the info file.

```
#define APP_ICONS_BASE (APP_RES_SHMEM_BASE +
                        APP_LVGL_IMGS_SIZE)
#define APP_ICONS_SIZE 0x107c40
```

3. Update the assigned size of the memory section named `res_sh_mem` to 0x200000 by redefining it in `app_config.h`:

```
#define RES_SHMEM_TOTAL_SIZE 0x200000
```

and the corresponding setting in `Project > Properties > C/C++ Build > MCU settings`.

4. Add the icon size to the total size of the resource loaded from Flash to SDRAM in the function `APP_LoadResource()` in the main file `lvgl_gui_face_rec_cm7.cpp`:

```
memcpy((void *)APP_LVGL_IMGS_BASE, pLvglImages, APP_LVGL_IMGS_SIZE +
                    APP_ICONS_SIZE);
```

Notice: *To complete the face recognition feature, the LVGL GUI app support is needed. The UI callback functions in the output UI HAL are called by the LVGL GUI app for handling the events from the UI screen. On the other hand, the output UI HAL calls the APIs from the LVGL GUI app to update the UI for showing the result and status. The development of the LVGL GUI app is relatively independent and introduced in Section 4.3.*

4.1.5 Start HAL devices and managers for face recognition

The enabled vision algo HAL and UI output HAL and their managers are started in the main file `lvgl_gui_face_rec_cm7.cpp` following the conversions of development on the framework as below:

1. Include the header file related to the two HAL managers by adding the code line:

```
#include " fwk_output_manager.h "
#include "fwk_vision_algo_manager.h"
```

2. Declare the HAL devices:

```
HAL_VALGO_DEV_DECLARE(OasisGUIFaceRec);
HAL_OUTPUT_DEV_DECLARE(UiGUIFaceRec);
```

3. Register the HAL devices:

```
HAL_VALGO_DEV_REGISTER(OasisGUIFaceRec, ret);
HAL_OUTPUT_DEV_REGISTER(UiGUIFaceRec, ret);
```

4. Initialize the managers:

```
FWK_MANAGER_INIT(VisionAlgoManager, ret);
FWK_MANAGER_INIT(OutputManager, ret);
```

5. Start the managers:

```
FWK_MANAGER_START(VisionAlgoManager, VISION_ALGO_MANAGER_TASK_PRIORITY, ret);
FWK_MANAGER_START(OutputManager, OUTPUT_MANAGER_TASK_PRIORITY, ret);
```

6. Define the priority of the manager tasks:

```
#define VISION_ALGO_MANAGER_TASK_PRIORITY 3
#define OUTPUT_MANAGER_TASK_PRIORITY 1
```

4.2 Add face database support on framework

The registered face feature data is accessed in the face database stored on Flash via a little file system. The steps to add the face database support are described below.

4.2.1 Add drivers for Flash storage

Copy the Flash interface FlexSPI driver files `fsl_flexspi.c` and `fsl_flexspi.h`, and the data encryption driver files `fsl_caam.c` and `fsl_caam.h` from the path `SDK_2_13_0_MIMXRT1170-EVK\devices\MIMRX1176\drivers\` to the `drivers` folder of the example SW.

4.2.2 Add board level support

1. Add the definitions of FlexSPI used for the Flash device on board in `board.h`:

```
#define BOARD_FLEXSPI    FLEXSPI1
#define BOARD_FLEXSPI_CLOCK  kCLOCK_FlexSpi1
#define BOARD_FLEXSPI_AMBA_BASE FlexSPI1_AMBA_BASE
```

2. Copy the operators and configurations files of the Flash device `flexspi_nor_flash_ops.c`, `flexspi_nor_flash_ops.h`, `sln_flash_config.c`, `sln_flash_config_w25q256jvs.h`, and`sln_flash_ops.h` under the path `smart HMI\coffee_machine\cm7\source\flash_config\` to the folder `board` of the example SW.
   - Uncheck “Exclude resource from build” in `C/C++ Build > Settings` after right-clicking on the files’ name and opening the Properties for enabling them to be built into the project.
3. Change the included header filename `sln_flash_config.h` to `sln_flash_config_w25q256jvs.h` in `sln_flash_config.c` and `flexspi_nor_flash_ops.h`.
4. Set the **FlexSPI1** clock source in the file `clock_config.c` referring to the coffee machine app.

4.2.3 Add adapter and middle level support

1. Copy the files `sln_flash.c`, `sln_flash.h`, `sln_encrypt.c`, and `sln_encrypt.h` as adapter drivers for the file system and app from the path of `smart HMI\coffee_machine\cm7\source\` to the folder `source` of the example. Update the new files:
   - Uncheck “Exclude resource from build” on them for building.
   - Change all the included header file name `sln_flash_config.h` to `sln_flash_config_w25q256jvs.h`.
2. Copy the folder `filesystem` containing the APIs for the little filesystem and HAL driver from `smart HMI\coffee_machine\cm7\source\` to the example SW. And update for the new folder:
   - Uncheck “Exclude resource from build” on it for building.

- Add the include path for it in project settings: "${workspace_loc:/${ProjName}/filesystem}"
- Change the included header file name sln_flash_config.h to sln_flash_config_w25q256jvs.h and fica_definition.h to app_config.h in the file sln_flash_littlefs.h.

3. Copy the folder littlefs containing the middle ware – little filesystem from the path SDK_2_13_0_MIMXRT1170-EVK\middleware\ to the example SW. And update the new folder:
   - Uncheck "Exclude resource from build" on it for building.
   - Add the include path for it in project settings:

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

4.2.4 Add HAL drivers

There are two HAL devices - file system and face database HAL supported for the database access feature and they are already implemented in the framework without any change. Enable them by adding the below definitions in board_define.h:

```
#define ENABLE_FLASH_DEV_Littlefs
#define ENABLE_FACEDB
```

And change the face database name for the example:

```
#define OASIS_FACE_DB_DIR "oasis_gui_face_rec"
```

4.2.5 Add app level support

1. Update the main file lvgl_gui_face_rec_cm7.cpp:
   - Include the header file related to the Flash file system HAL manager by adding the code line: #include "fwk_flash.h"
   - Declare and register file system HAL device:

```
HAL_FLASH_DEV_DECLARE(Littlefs);
HAL_FLASH_DEV_REGISTER(Littlefs, ret);
```

   Note: *The file system HAL device must be registered before all device managers are initialized in the function* *APP_InitFramework().*
   - Call the function BOARD_ConfigMPU() in APP_BoardInit() to config MPU.
2. Set the file system assignment on Flash in the file app_config.h by defining the macro definitions used in the file sln_flash_littlefs.h:

```
#define FICA_IMG_FILE_SYS_ADDR (FLASH_IMG_SIZE + RES_SHMEM_TOTAL_SIZE)
#define FICA_FILE_SYS_SIZE (0x280000)
```

4.2.6 Configurations

Some Flash-related codes are executed in the SRAM ITC area for enough performance. Copy the folder linkscripts containing the linker configurations from the path smart HMI\coffee_machine\cm7\ to the example SW.

4.3 Implement a LVGL GUI app

The development of a LVGL GUI app based on framework calls the APIs from output UI HAL and provides the APIs to output UI HAL (See Section 4.1.3 for the implementation of output UI HAL).

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

Below is the implementation introductions:

1. The customized codes are implemented in the `custom.c` and `custom.h` given by GUI Guider as the interface between the GUI Guider project and the embedded system project.
   - Add the new functions named `gui_xxx()` in `custom.c` to achieve the below functions:
     - For output UI HAL and GUI app to update UI.
     - For GUI app to trigger events by calling UI callback functions from output UI HAL.

   For example, the new function `gui_event_face_rec_action()` calls UI callback functions to handle one of the events of face registration, face recognition and deleting user triggered from the GUI app when the related button is clicked.

   Note: *The function `gui_set_virtual_face()` called in output UI HAL for preview mode needs to be implemented in `custom.c`:*
   - *Clone function `gui_set_virtual_face()` from `smart HMI\coffee_machine\cm4\custom \custom.c`.*
   - *Change the widget's name `home_img_cameraPreview` to `screen_img_camera_preview` in the function.*
   - *Implement the UI callback functions with the same prototype to all the ones in output UI HAL under the control of the macro definition `#ifndef RT_PLATFORM` in `custom.c` for being compatible with the GUI Guider project because these functions in output UI HAL are dependent with the embedded platform. In `custom.c`, they depend on the simulator on GUI guider and are independent to the embedded platform. For example, the face registration callback is implemented as below for the GUI Guider simulator running:*

```
#ifndef RT_PLATFORM
void UI_Registration_Callback()
{
  gui_hide_del_user_btn(true);
  s_InAction = false;
  return;
}
```

   Note: *Refer to the same prototype of the function introduced in step 6 of Section 4.1.3*

   The macro definition `RT_PLATFORM` is set on the project settings of MCUXpresso as shown in Figure 7:

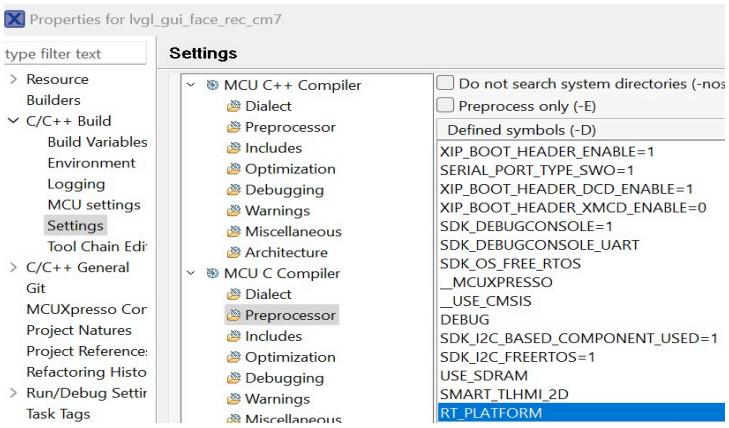


Figure 7. The setup of macro definition "RT_PLATFORM" on project settings

- Declare all the functions named `UI_xxx_Callback()` and `gui_xxx()` in `custom.h` and add `custom.h` included in `smart_tlhmi_event_descriptor.h` to share the GUI APIs to UI output HAL.

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 `face_rec` for the new example.
   - Copy the above updated `custom.c` and `custom.h` to the new GUI Guider project software.
   - Open the new `face_rec` project on GUI Guider. Update as below:
     – Add the new button labeled **Delete User**. Add the flag **Hidden** to it so that the button will be hidden when the GUI app starts up.
     – Add the code line of calling the API `gui_event_face_rec_action()` with different event ID parameter on the "Released" trigger in the **Event Setting** of all the buttons **Registration, Recognition** and **Delete User** for triggering the events of face registration, face recognition and deleting user. Figure 8 shows the code for the event of the button **Registriation:**

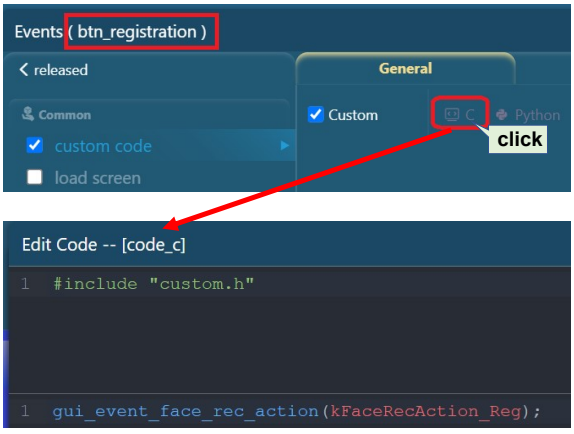


Figure 8. Coding for the event of the button "Registration"

3. Update the generated code from GUI Guider to the MCUXpresso project.
   - Replace the contents except for the folder `images` in the folder `generated` of the MCUXpresso project SW with the corresponding ones in the folder `generated` of GUI Guider project SW.

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

5 Verifications with the example project

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

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

The LVGL GUI face recognition example works normally as below:

- **Preview:** With power up, the video streams captured by the camera shows on the specific area of camera preview on the GUI screen. The status label displays “Preview…”. For details, see Figure 3. The button **Delete User** is hidden. When clicking the area outside the buttons and images, it shows the preview state as the above after the face registration or recognition action ends.
- **Registration**:
  - **Startup**: When the **Registration** button is clicked, the face registration starts. The status label changes to display “Registration…”, the face guide rectangle shows blue, and the progress bar starts showing the progress. Make sure the user’s face showing into the blue face guide rectangle for registration.
  - **Success**: The status label shows “Registration…OK” and the registered user ID number, the face guide rectangle becomes red if the face registration is successful before the progress shows full on the bar.
  - **Failure** -> Time out: The status label shows “Registration…Time out” if the face registration is still failed when the progress shows full on the bar.
  - **Failure** -> Duplication: The status label shows “Registration…Failed”, the face guide rectangle becomes green if the registered face is recognized before the progress shows full on the bar.
- **Recognition**:
  - **Startup**: When the **Recognition** button is clicked, the face recognition starts. The status label changes to display “Recognition…”, the face guide rectangle shows blue, and the progress bar starts showing the progress. Make sure the user’s face is shown into the blue face guide rectangle for registration.
  - **Success**: The status label shows “Recognition…OK” and the recognized user ID number, the face guide rectangle becomes green if the face recognition is successful before the progress shows full on the bar. At the point, the button **Delete User** appears. It means that the user is allowed to be deleted only when it is recognized.
  - **Failure**: The status label shows “Recognition…Time out” if the face recognition is still failed when the progress shows full on the bar.
- **Delete User:** When the button “Delete User” is clicked, after the face recognition is successful, the status label changes to display “Delete User…OK” with the face guide rectangle becoming blue and the progress showing full on the bar. The button **Delete User** is hidden again. The recognized face/user is deleted from the database. It means this face/user cannot be recognized until is registered again.

6 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 must be 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.

7 Revision history

Table 1. Revision history

Document ID	Release date	Description
AN14263 v.1	19 April 2024	Initial version

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

1 Overview 2
1.1 Framework overview 2
1.2 Light and Versatile Graphics Library (LVGL) 3
1.3 GUI Guider 3
2 Development environment 4
3 Vision architecture on framework 4
4 Implement face recognition on framework 5
4.1 Enable face recognition feature on framework 6
4.1.1 Add vision algo model library 6
4.1.2 Enable vision algo HAL 7
4.1.3 Enable output UI HAL 9
4.1.4 Implement the icons support 11
4.1.5 Start HAL devices and managers for face recognition 12
4.2 Add face database support on framework 13
4.2.1 Add drivers for Flash storage 13
4.2.2 Add board level support 13
4.2.3 Add adapter and middle level support 13
4.2.4 Add HAL drivers 14
4.2.5 Add app level support 14
4.2.6 Configurations 14
4.3 Implement a LVGL GUI app 14
5 Verifications with the example project 17
6 Note about the source code in the document 17
7 Revision history 18
Legal information 19

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

Date of release: 19 April 2024
Document identifier: AN14263