AN14147

Implement LVGL GUI Camera Preview on Framework

Rev. 1.0 — 3 January 2024 **Application note**

Document information

Information	Content
Keywords	AN14147, LVGL GUI, framework, camera preview
Abstract	This application note describes how to enable LVGL GUI on framework to implement camera video showing on GUI screen with a simple GUI app.

1 Overview

NXP has launched a solution development kit named SLN-TLHMI-IOT, which focuses on smart human machine interface (HMI) applications. It enables smart HMI with machine learning (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 document introduces the basic software design and architecture of the applications, covering all components of the solution containing framework to help developers implement their applications more easily and efficiently using 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 on the framework step by step from basics. *Implement Camera Preview with Framework Enabled on SDK* (document AN14015) shows how to enable the framework on the SDK with a simple example – camera preview. This application note is the second part based on the first one. The document shows how to enable LVGL GUI on the framework enabled in the first application note with a simple GUI app – camera preview.

This application note describes enabling LVGL GUI on the framework to implement camera video showing on a GUI screen with a simple GUI app on the SLN-TLHMI-IOT board. At a high level, implementing it contains the below steps:

- Develop an LVGL GUI app with images on GUI Guider.
- Enable LVGL GUI on framework.
- Implement the GUI app on the enabled LVGL.
- Build and deploy the image resources for the GUI app.

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

- Understand the framework and the solution software more deeply.
- Develop their LVGL GUI application on the framework.

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 – a core part
- Hardware abstraction layer (HAL) devices
- Messages/events

Figure 1 shows the overview of the framework mechanism.

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.

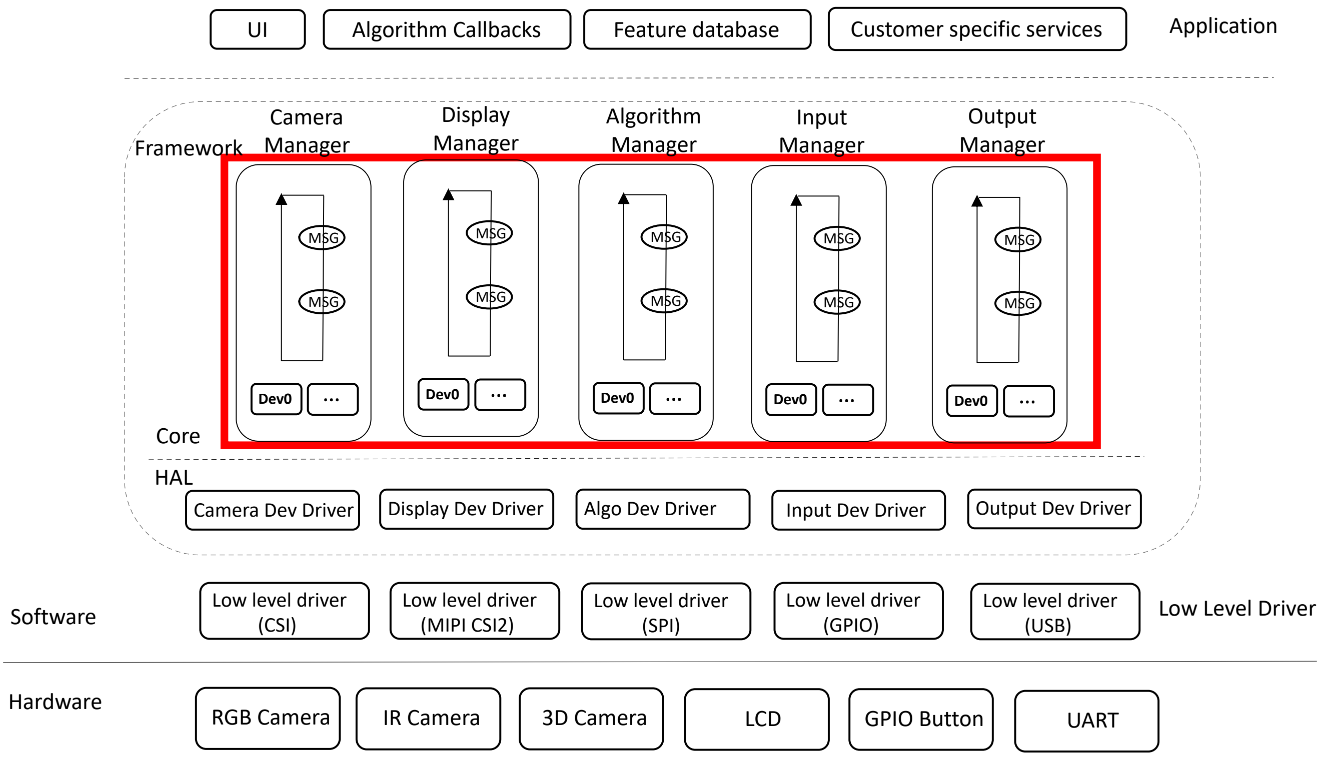

Figure 1. Smart HMI software architecture on framework

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 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, refer to 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 open-source LVGL graphics library. The drag-and-drop editor of GUI Guider makes it easy to use the many features of LVGL. These features include widgets, animations, and styles to create a GUI with minimal or no coding.

With the click of a button, you can run your application in a simulated environment or export it to a target project. Adding embedded user interfaces to your application is now easy and fast with the generated code from GUI Guider.

GUI Guider is free to use with NXP 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, visit https://lvgl.io/ and GUI Guider.

2 Development environment

First, prepare and set up the hardware and software environment for implementing the LVGL GUI camera preview example on the framework.

Hardware environment

The following hardware is required for the demonstration after development:

- 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

Software environment

The software tools and their versions used in this application note are introduced as below:

- MCUXpresso IDE V11.7.0
- GUI Guider V1.5.0 GA or greater
- `sln_tlhmi_iot_camera_preview_cm7` v1.0.0 – example code of application note is based on *Implement Camera Preview with Framework Enabled on SDK* (document AN14015).
- RT1170 SDK V2.13.0 – code resource for the development
- SLN-TLHMI-IOT software V1.1.1 – smart HMI source codes released on NXP GitHub repository as the codes resource for the development.

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

3 LVGL GUI design on smart HMI software with framework

To better understand the implementation process, first introduce the design of how an LVGL GUI app is integrated into the smart HMI software using the framework.

Figure 2 shows the LVGL GUI app codes and LVGL codes that are separately added to the application and middleware level of smart HMI software after the app is developed on GUI Guider.

There are two HAL devices implemented in the framework HAL level to support an LVGL GUI application. One is a display LVGL HAL device to handle the LVGL GUI display task periodically and refresh the camera video to the GUI screen. The other is an output UI HAL device to handle the data output between the GUI app and other modules. For example, handle the result of inference from the vision algorithm module and update the GUI display.

Remark: In this application note, only LVGL display HAL device is enabled as only the camera preview function is demonstrated in the example.

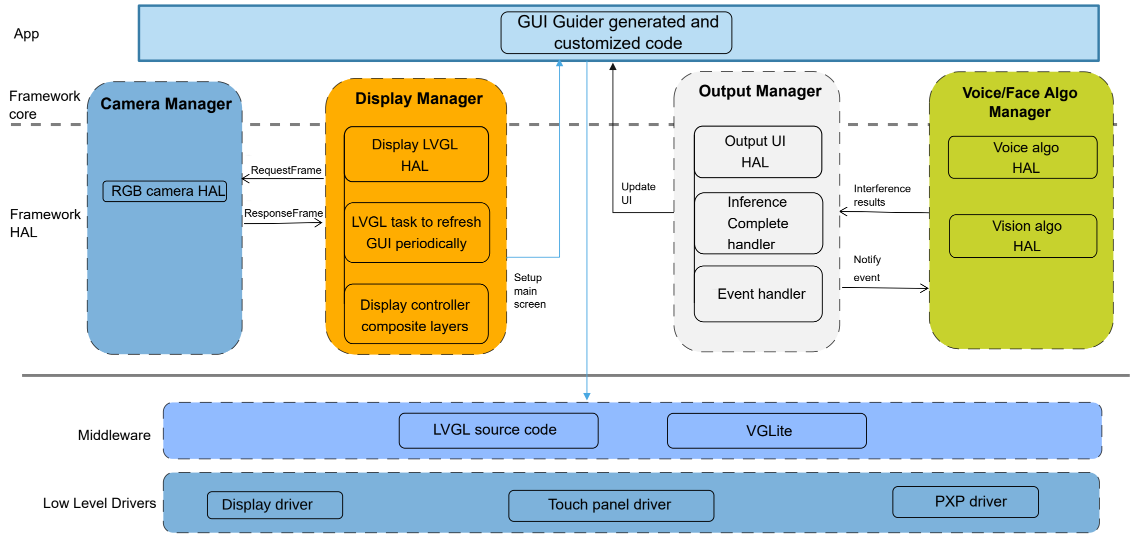

Figure 2. The architecture of smart HMI software with LVGL GUI app integrated

Generally, some images are used to make the display more beautiful in a GUI app. The image resources are stored on the flash in the embedded system. A mechanism is designed to deploy the image resources used on GUI Guider to the smart HMI platform for making the developers do it more easily. Figure 3 shows a simple self-made tool. It builds the image resources generated by GUI Guider to a binary file containing the image data. Meanwhile, the tool generates an information text file containing the addresses on the SDRAM and the total size of the images. The binary data file is programmed into the flash. The image data is then loaded from the flash to SDRAM during system startup using the information in the text file. Therefore, the GUI app can show the images on the GUI screen on the smart HMI platform.

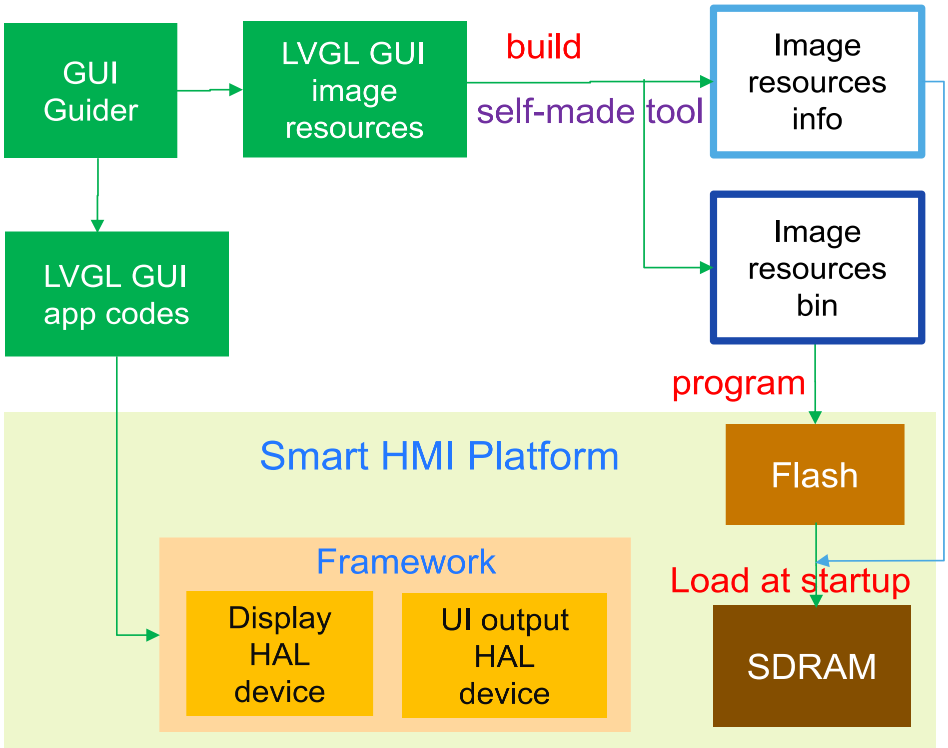

Figure 3. LVGL GUI images and codes integration block

4 Implement LVGL GUI camera preview on framework

The LVGL GUI camera preview example on the framework is implemented on the example code of *Implement Camera Preview with Framework Enabled on SDK* (document AN14015). It shows how the LVGL GUI is enabled on the framework and a camera preview GUI example is implemented on the enabled LVGL. First, the LVGL GUI example is developed by GUI Guider.

4.1 Develop LVGL GUI app with images on GUI Guider

How to develop a GUI app with GUI Guider is not introduced in the application note (To get more resources, see GUI Guider). A simple LVGL GUI example app with images on GUI Guider is developed for this application note. To create a project with the following selections, refer to *GUI Guider v1.6.1 User Guide* (document GUIGUIDERUG):

- The LVGL library V8.3.2
- The board template: MIMXRT1176xxxxx
- The application template: EmptyUI
- The panel type: 1280*720 – (RK055MHD091)

Figure 4 shows the main screen (only one) of the GUI app after developing.

Figure 4. LVGL GUI app main screen

In the screen, the big blank area (size is 640*480) on the left is used for the camera preview. The two buttons labeled "Registration" and "Recognition" are reserved for face registration and recognition for the coming application note. Currently, it only provides hints on the label by clicking them.

4.2 Enable LVGL GUI on framework

Clone the software `sln_tlhmi_iot_camera_preview_cm7` v1.0.0 and change all the name strings `*_camera_preview_*` to `*_lvgl_gui_camera_preview_*` as the basis of the LVGL GUI camera preview example for this application note. Add the features related to the LVGL GUI function to the software at driver, middleware, and board levels.

- **Add touch panel support**

Generally, button touch is a basic feature in a GUI app. The buttons are enabled in the GUI example of this application note, as mentioned above.

To add the support to the software, perform the following steps:

1. Copy the related driver files `fsl_rt911.c` and `fsl_rt911.h` from the `[SDK V2.13.0]\` components to a new folder "touchpanel" in the software.
2. Uncheck "Exclude resource from build" in C/C++ Build > Settings after right-clicking on the "touchpanel" group and opening the properties to enable the new group to be built into the project.
3. Add include path for touch panel driver on Project > Properties > C/C++ Build > settings > Tool Settings > MCU C compiler > Includes and MCU C++ compiler > Includes:

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

4. Modify below board level of definitions for the touch panel in `board.h`.

```
#define BOARD_MIPI_PANEL_TOUCH_I2C_CLOCK_DIVIDER (4U)
#define BOARD_MIPI_PANEL_TOUCH_RST_GPIO GPIO8
#define BOARD_MIPI_PANEL_TOUCH_RST_PIN 21
#define BOARD_MIPI_PANEL_TOUCH_INT_PIN 20
```

5. Initialize the I2C for touch panel control in `board\peripherals.c` by calling `BOARD_MIPIPanelTouch_I2C_Init()` in the function `BOARD_InitPeripherals()`.
6. Add `#include "board.h"` in `board\peripherals.h`.

- **Add middleware – LVGL support**

Besides LVGL, the VGLite graphics API based on the 2D GPU module of i.MX RT117H is used for the GUI display. To enable them in the software, add LVGL as below:

1. Copy the "lvgl" folder from `gui_guider\camera_preview\sdk\Core\` generated by GUI Guider to the software. Remove `lvgl.mk` since it is only used for GUI Guider.
2. Uncheck "Exclude resource from build" in C/C++ Build > Settings after right-clicking on the "lvgl" group and opening the properties to enable the new group to be built into the project.
3. Add include path for the LVGL on Project > Properties > C/C++ Build > settings > Tool Settings > MCU C compiler > Includes and MCU C++ compiler > Includes:

```
"${workspace_loc:/${ProjName}/lvgl}"
"${workspace_loc:/${ProjName}/lvgl/lvgl/src}"
"${workspace_loc:/${ProjName}/lvgl/lvgl/src/font}"
"${workspace_loc:/${ProjName}/lvgl/lvgl/src/hal}"
"${workspace_loc:/${ProjName}/lvgl/lvgl}"
```

4. Copy the LVGL GUI app configuration file `lv_conf.h` under `gui_guider/SDK/core/source/to source/` of the software.

- **Add middleware – VGLite support**

1. Copy the "vglite" folder from the `[SDK V2.13.0]\middleware` to the software and delete the unused folder "elementary" under the "vglite".
2. Uncheck "Exclude resource from build" in C/C++ Build > Settings after right-clicking on the "vglite" group and opening the properties to enable the new group to be built into the project.
3. Add include path for the VGLite on Project > Properties > C/C++ Build > settings > Tool Settings > MCU C compiler > Includes and MCU C++ compiler > Includes:

```
"${workspace_loc:/${ProjName}/vglite/inc}"
"${workspace_loc:/${ProjName}/vglite/font}"
"${workspace_loc:/${ProjName}/vglite/font/mcufont/decoder}"
"${workspace_loc:/${ProjName}/vglite/VGLite/rtos}"
```

```
"${workspace_loc:/${ProjName}/vglite/VGLiteKernel}"
"${workspace_loc:/${ProjName}/vglite/VGLiteKernel/rtos}"
```

4. Add the below definition in the file `lv_conf.h` for enabling VGLite:

```
#define LV_USE_GPU_NXP_VG_LITE 1
```

- **Add board support**

To add LVGL and VGLite support at board level, perform the following steps:

1. Copy `lvgl_display_support.c` and `lvgl_display_support.h` from `[SDK V2.13.0]\board\evkmimxrt1170\lvgl_examples`, and `vglite_support.c` and `vglite_support.h` from `[SDK 2.13.0]\boards\evkmimxrt1170\lvgl_examples\lvgl_demo_widgets\cm7` to the folder "board" of the software.
2. To support camera preview and GUI screen rotation, modify `lvgl_display_support.c`. It is done because of the display panel design on the board with a 270 degree rotation compared to the display on GUI Guider.
   Camera preview is on another layer different from the GUI display. It is enabled with the macro definition `ENABLE_DISPLAY_DEV_LVGLCameraPreview`.
   - Add a callback function `DEMO_CameraBufferSwitchOffCallback()` for the new camera preview layer referring to the callback function `DEMO_BufferSwitchOffCallback()` of the existing GUI display layer, declared as below:

   ```
   #ifdef ENABLE_DISPLAY_DEV_LVGLCameraPreview
   static void DEMO_CameraBufferSwitchOffCallback(void *param, void
    *switchOffBuffer);
   #endif
   ```

   - To enable or disable camera preview, add the function `lv_enable_camera_preview()`.
   - To merge the initialization of the camera preview layer with the GUI display layer, modify the initialization function `lv_port_disp_init()`. As shown below, configure the camera preview layer (Layer 1) and set the callback function to it:

   ```
   g_dc.ops->setLayerConfig(&g_dc, 1, &fbInfo);
   g_dc.ops->setCallback(&g_dc, 1, DEMO_CameraBufferSwitchOffCallback, NULL);
   ```

   For better performance, the PXP hardware handles the GUI screen rotation.
   But note that there are respective PXP APIs implemented in the LVGL and the framework. The APIs in the framework instead of LVGL are used to handle the rotation in the software. Therefore, the macro definition `LV_USE_GPU_NXP_PXP` for LVGL is not enabled. The related implementations are as below:
   - To use PXP for rotation, include the header files in the framework:

   ```
   #if ((LV_HOR_RES_MAX == DEMO_PANEL_HEIGHT) && (LV_VER_RES_MAX ==
    DEMO_PANEL_WIDTH))
   #include "fwk_graphics.h"
   #include "hal_graphics_dev.h"
   #endif
   ```

   - Set the macro definition to enable rotation:

   ```
   #define DEMO_USE_ROTATE 1
   ```

   - Add a function `_rotate_270()` to call the PXP API supported in the framework. Call it in `DEMO_FlushDisplay()` for rotation implementation when flushing the display.
3. Modify `lvgl_support.h` as below:
   - Add `#include "board_define.h"`. The macro definition `ENABLE_DISPLAY_DEV_LVGLCameraPreview` is defined in the header file.

- Add the declaration.

```
#ifdef ENABLE_DISPLAY_DEV_LVGLCameraPreview
void lv_enable_camera_preview(void* frameBuffer, bool enable);
#endif
```

- **Add display LVGL HAL device in framework**
  Display LVGL HAL device handles the LVGL GUI display task periodically and refreshes the camera video to the GUI screen, as mentioned (see Section 3). Here, the document introduces how to integrate the LVGL into the framework after adding and enabling the related source codes at driver and board level. It is easy to do for the developers by cloning the existing display LVGL HAL driver where the basic driver model has been built.

1. Clone the HAL driver file `hal_display_lvgl_coffeemachine.c`.
2. Replace all the strings "coffeemachine" with "camerapreview" for the file including the file name.
3. Delete the line: `#include "smart_tlhmi_event_descriptor.h"`.
4. Delete the unused variable extern `preview_mode_t g_PreviewMode`.
5. Modify `HAL_DisplayDev_LVGLCameraPreview_Blit()` to enable the camera preview layer to refresh camera video on a GUI screen by calling the following function:

```
lv_enable_camera_preview(lcdFrameAddr, true);
```

6. Remove the calling `FWK_LpmManager_RegisterRequestHandler()` and the related variable used for low-power mode in the function `HAL_DisplayDev_LVGLCameraPreview_Register()`.

```
static hal_lpm_request_t s_LpmReq = {.dev =
 &s_DisplayDev_LVGLCameraPreview, .name = "LVGLCameraPreview"};
```

7. Include the header file `guider_images.h` for supporting the setup of the images used in the LVGL GUI example since some images are used. If no images, remove the function `setup_imgs()` called in `_LvglTask()` without including.

Enable and config the display LVGL HAL for camera preview in `board_define.h`:

1. Comment the definition used for the previous example:

```
//#define ENABLE_DISPLAY_DEV_Lcdifv2Rk055mh
```

2. Remove the comment of the below macro definition used for the coffee machine app:

```
#define ENABLE_DISPLAY_DEV_LVGLCoffeeMachine
```

3. Replace all the strings "CoffeeMachine" with "CameraPreview" to transfer all the definitions about the enablement and the configurations (for example, height, width, and so on) of the coffee machine app to the camera preview example.
4. Modify the start point configurations of camera preview on the display panel according to the GUI example settings developed on GUI Guider:

```
#define DISPLAY_DEV_LVGLCameraPreview_StartX 93
#define DISPLAY_DEV_LVGLCameraPreview_StartY 127
```

5. Add the definition for enabling the camera preview layer:

```
#define ENABLE_LAYER_LVGLCameraPreview
```

Remark: As mentioned in Section 3, the output UI HAL driver is not needed for the LVGL camera preview example, so it is not introduced in the application note. It is easy to implement by cloning the existing one for other apps just like the display LVGL HAL driver.

4.3 Implement LVGL GUI app

After the LVGL is enabled, the next step is to implement a GUI app on the software. The work becomes simpler since many of the C codes of GUI app are generated by GUI Guider.

- **First, integrate the codes to the software**
  It is easy as the interface part is implemented in display LVGL HAL of the framework. It is required to add the codes to the software with a few modifications:

1. Copy the "custom" and "generated" folders from the GUI Guider project to the software.
2. Uncheck "Exclude resource from build" in C/C++ Build > Settings after right-clicking on the "custom" and "generated" groups and opening the properties for enabling the new groups to be built into the project.
3. Remove all *.mk and *.py files and the folder "mPythonImages" in both folders since they are unused in the software.
4. Add include files on Project > Properties > C/C++ Build > settings > Tool Settings > MCU C compiler > Includes and MCU C++ compiler > Includes:

```
"${workspace_loc:/${ProjName}/custom}"
"${workspace_loc:/${ProjName}/generated}"
"${workspace_loc:/${ProjName}/generated/guider_customer_fonts}"
"${workspace_loc:/${ProjName}/generated/guider_fonts}"
"${workspace_loc:/${ProjName}/generated/images}"
```

- **Enable system tick on FreeRTOS for the LVGL GUI app**
  The elapsed time is required for the LVGL to handle the task. The time is measured with the system tick. The FreeRTOS provides a hook function `vApplicationTickHook()` for using the system tick, which must be enabled for the LVGL GUI app running.

1. Implement `vApplicationTickHook()` in `source\os_hook.c`, as below:

```
void vApplicationTickHook(void)
{
if (g_LvglInitialized)
{
lv_tick_inc(portTICK_PERIOD_MS);
}
}
```

2. The `g_LvglInitialized` is a global variable declared in the display LVGL HAL file `hal_display_lvgl_camerapreview.c`. It is set to 1 when LVGL initialization is completed. So, it is also needed to add extern volatile `bool g_LvglInitialized;` and `#include <stdbool.h>` for the boot type support in the `os_hook.c`.
3. Define `configUSE_TICK_HOOK` to 1 for enabling the tick hook on FreeRTOS.

- **Register the display LVGL HAL device in the main file**

1. Rename the previous main file in the group "source" to `lvgl_gui_camera_preview_cm7.cpp` for the example of this application note.
2. Change the declaration and registration of display HAL device from `Lcdifv2Rk055mh` to `LVGLCameraPreview`:

```
HAL_DISPLAY_DEV_DECLARE(LVGLCameraPreview);
HAL_DISPLAY_DEV_REGISTER(LVGLCameraPreview, ret);
```

- **Configure the project on MCUXpresso**
  Set optimization level to Optimize most (-O3) in Project > Properties > C/C++ Build > settings > Tool Settings > MCU C compiler > Optimization and MCU C++ compiler > Optimization for a good performance of the LVGL GUI app.

Remark:

- For this example, the development of the app and the software codes are completed if no images are supported in the GUI app. More information about GUI image support is introduced in the below section.
- In practice, some complex customized codes are required for the GUI app. It can be more convenient to develop the codes on the MCUXpresso IDE (recommend implementing them in the folder "custom"). In this case, the development of the app needs more workload.

4.4 Implement GUI image support

As mentioned, images are used in an LVGL GUI app. The section shows how the images are implemented in the example software.

- **Build image resource**

1. A simple self-made tool is used to build the images for the example software (see Section 3). The tool is packaged in the folder "resource_build" under `[SLN-TLHMI-IOT software V1.1.1]\tools\`. Copy the folder to the same directory to the example software.
2. Create a folder "resource" in the same example software directory. Clone the following files from `[SLN-TLHMI-IOT software V1.1.1]\coffee_machine\resource\` to the new folder:
   - `coffee_machine_resource.txt` (each image *.c file with the relative path is recorded in it for the tool to search and build)
   - `coffee_machine_resource_build.bat` (Windows script file for executing the building automatically)
   - `coffee_machine_resource_build.sh` (Linux script file for executing the building automatically)

   Then, rename the three files to `camera_preview_resource.txt`, `camera_preview_resource_build.bat`, and `camera_preview_resource_build.sh`.
3. Copy the folder "images" including the *.c files of all the images under `generated\` to the folder "resource".
4. Open and modify the Windows script file `camera_preview_resource_build.bat` for the camera preview example, as below:

   ```
   ..\resource_build\resource_build.exe camera_preview_resource.txt
    camera_preview_resource.bin 0 0.
   ```

   Remark: Do similar modifications to `camera_preview_resource_build.sh`.
5. Remove all information of image, icon, and sound for the coffee machine app in `camera_preview_resource.txt`. For the tool to build the images correctly, add the path relative to the tool file `resource_build.exe` and the *.c files of the images in the folder "images":

   ```
   image ../resource/images/_NxpFaceRec_185x55.c
   image ../resource/images/_NxpVoiceRec_185x55.c
   ```

6. Change all strings "coffee_machine" to "camera_preview" in the *.txt file.
7. Execute the script file `camera_preview_resource_build.bat` by double-clicking on it. It generates the binary file `camera_preview_resource.bin` containing the image data to be programmed on the flash. It also generates the file `resource_information_table.txt` containing the relative addresses on the SDRAM and total size of the image for the image resources access in the example.

- **Add images support**
  - First, set up the access to the images data on the flash.
    1. Create a source file `setup_images.c` under `generated/images/` group.
    2. Copy all the image descriptors from the *.c files of the images to the `setup_images.c`. For example, the below descriptor array in the image *.c file `NxpFaceRec_185x55.c`:

       ```
       lv_img_dsc_t _NxpFaceRec_185x55 = {
       .header.always_zero = 0,
       .header.w = 185,
       .header.h = 55,
       ```

```
.data_size = 10175 * LV_COLOR_SIZE / 8,
.header.cf = LV_IMG_CF_TRUE_COLOR_ALPHA,
.data = _NxpFaceRec_185x55_map,
};
```

Then delete the `.data = _NxpFaceRec_185x55_map` used for the GUI Guider on PC. So, the data is accessed with another way as below function `setup_imgs()`.
Remark: Delete all the image *.c files after completing the copy of the descriptors from them.

3. Implement a function `setup_imgs(unsigned char *base)` to set up the pointers to each image data on the SDRAM. The setup codes are generated in the file `resource_information_table.txt` and can be copied to the new function.

```
_NxpFaceRec_185x55.data = (base + 0);
_NxpVoiceRec_185x55.data = (base + 30528);
```

4. Create a header file `guider_images.h` under `generated/images/` group for declaring the API `setup_imgs()`.
Remark: With the above modifications, the sub-folders and files in the folder "generated" except for the sub-folder "images" can be overwrite directly when the generated codes in GUI Guider are updated. This makes the update more simple.

– Next, implement loading the images from the flash to the SDRAM at system startup in the main file `lvgl_gui_camera_preview_cm7.cpp`.

1. Define an array variable `res_sh_mem[]` for the data buffer of images in the main file `lvgl_gui_camera_preview_cm7.cpp`.

```
AT_RES_SHMEM_SECTION_ALIGN(unsigned char res_sh_mem[RES_SHMEM_TOTAL_SIZE],
 64);
```

2. Implement the function `APP_LoadResource()` to load LVGL GUI images from the flash to the specific section of SDRAM defined by the buffer `res_sh_mem[]`.

```
void *pLvglImages = (void *)(FLEXSPI_FLASH_BASE + FLASH_IMG_SIZE);
memcpy((void *)APP_LVGL_IMGS_BASE, pLvglImages, APP_LVGL_IMGS_SIZE);
```

3. Call `APP_LoadResource()` in the function `APP_BoardInit()` executed on system startup.
4. Include the header file `app_config.h`.
Remark: The above involved macro definitions are defined in a new header file `app_config.h` (see below steps from 5) except the `AT_RES_SHMEM_SECTION_ALIGN` is already defined in `board_define.h`. It is a memory section assigned in SDRAM to store the LVGL image resources.
5. Clone `app_config.h` from `[SLN-TLHMI-IOT software V1.1.1]\coffee_machine\source\` to `source\` of the example software.
6. Delete all the definitions except for an LVGL image resource in the file.
7. Define the total byte size of `array res_sh_mem[]` in the file. It must be more than the size of the image data:

```
#define RES_SHMEM_TOTAL_SIZE 0x100000
```

8. Define the actual total size of all the images in the file. The value of the size is generated in the file `resource_information_table.txt`. So, copy it.

```
#define APP_LVGL_IMGS_SIZE 0x00ee80
```

9. Define the start address and size of the code on the flash according to the memory assignment on project settings. The value of the start address plus size is the start address of the image data on the flash. It determines the start address to program the image data to the flash.

```
#define FLASH_IMG_SIZE 0x800000
#define FLEXSPI_FLASH_BASE FlexSPI1_AMBA_BASE
```

– **Config the project**
  Add memory assignment on SDRAM for the image data section "res_sh_mem" on Project > Properties > C/C++ Build > MCU settings. The size is set to 0x100000 same to the definition of `RES_SHMEM_TOTAL_SIZE`.

Type	Name	Alias	Location	Size	Driver
Flash	BOARD_FLASH	Flash	0x30000000	0x800000	MIMXRT1170_...
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	

Figure 5. Add memory assignment to the GUI image data

Remark: For more details about the modifications introduced above, check the attached example software.

5 Verifications with the example project

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

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

The LVGL GUI camera preview example works normally, as below:

The video streams captured by the camera shows on the specific area of camera preview on the GUI screen. With clicking the button "Registration", the label displaying "Preview..." changes "Start registration...". With clicking the button "Recognition", the label changes to display "Start recognition...".

Remark: It demonstrates the hints on GUI screen without any actual action, as mentioned in Section 4.1.

The label returns to display "Preview…" when clicking any other area out of the buttons and images.

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 2023 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.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

7 Revision history

Table 1 summarizes the revisions done to this document.

Table 1. Revision history

Document ID	Release date	Description
AN14147 v1.0	3 January 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.

Amazon Web Services, AWS, the Powered by AWS logo, and FreeRTOS — are trademarks of Amazon.com, Inc. or its affiliates.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

i.MX — is a trademark of NXP B.V.

J-Link — is a trademark of SEGGER Microcontroller GmbH.

Contents

Date of release: 3 January 2024
Document identifier: AN14147