AN13948

Integrating LVGL GUI Application into Smart HMI Platform

Rev. 1 — 16 June 2023 Application note

Document Information

Information	Content
Keywords	AN13948, LVGL, smart HMI, GUI Guider
Abstract	This application note describes how to integrate the LVGL GUI application developed by the user into the smart HMI software platform based on the framework.

1 Introduction

NXP has launched a solution development kit named SLN-TLHMI-IOT. It focuses on smart HMI applications containing two apps - coffee machine and elevator (smart panel app is coming soon). To provide information to the user, some basic documents are included, for example, the developer guide. The guide introduces the basic software design and architecture of the applications covering all solution components. These components include bootloader, framework, and HAL design to help the developers more easily and efficiently implement their applications using the SLN-TLHMI-IOT.

For more details about the documents and the solution, visit NXP EdgeReady Smart HMI Solution based on i.MX RT117H with ML Vision, Voice and Graphical UI.

However, the introduction focuses on the ideas and basic usage. Because of the compliance of the software based on the framework, it is still not easy for the developers to know how to implement their applications. To speed up the development, additional guides are required to introduce how to implement the major components (for example, LVGL GUI, vision, and voice recognition) step by step. For example, customers should have their own LVGL GUI application different from the present apps in the solution. After implementing their LVGL GUI with the GUI Guider provided by NXP, they must integrate it into the smart HMI software platform based on the framework.

This application note describes how to integrate the LVGL GUI application developed by the user into the smart HMI software platform based on the framework. The reference codes are also presented along with this application note.

Note: *This application note does not explain how to develop the GUI based on LVGL with the GUI Guider software tool.*

The overview of the LVGL and GUI Guider is described in Section 1.1 and Section 1.2.

1.1 Light and Versatile Graphics Library

Light and Versatile Graphics Library (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.

1.2 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 utilize 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. Generated code from GUI Guider can easily be added to your project, accelerating the development process and allowing you to add an embedded user interface to your application seamlessly.

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

Prepare and set up the development environment for developing and integrating a GUI app to the smart HMI platform.

Hardware environment

The following hardware is required for the demonstration after development:

- The smart HMI development kit based on NXP i.MX RT117H
- SEGGER J-Link with a 9-pin Cortex-M adapter

Software environment

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

- GUI Guider V1.5.0-GA
- MCUXpresso IDE V11.7.0
  Note: *A bug in versions before 11.7.0 does not allow proper build-in multicore projects. Therefore, version 11.7.0 or greater is required.*
- RT1170 SDK V2.12.1
- SLN-TLHMI-IOT software platform – smart HMI source codes released in our official GitHub repository

To learn more about how to set up and install the hardware and software environment, see *Getting Started with the SLN-TLHMI-IOT* (document MCU-SMHMI-GSG).

3 Integrate LVGL GUI application into smart HMI platform

The smart HMI software platform is built on framework architecture. Developers find it difficult to add their LVGL GUI application to the smart HMI software platform even if they read the developer guide and know about the framework. The next sections explain how to implement it step by step.

3.1 Develop LVGL GUI application on GUI Guider

As mentioned above, how to develop the LVGL GUI on GUI Guider is not the emphasis in this application note. But a GUI example is necessary. Therefore, one simple GUI template named SliderProgress provided in GUI Guider is selected as the GUI example for a quick setup. The SliderProgress GUI template is used because it contains an image that is required to demonstrate building image resources in the application. The GUI example is very easy to generate: To create a project with the updated LVGL library V8.3.2 and the board template as MIMXRT1176xxxxx, refer to *GUI Guider User's Guide* (document GUIGUIDERUG). Figure 1 shows the project settings.

Note: *The panel type must be selected, as shown in the red box in Figure 1, as it is used on the current development board.*

After creating the project, run the simulator to generate the related LVGL GUI codes and build the project as well. You may check the effect of the GUI example on the simulator.

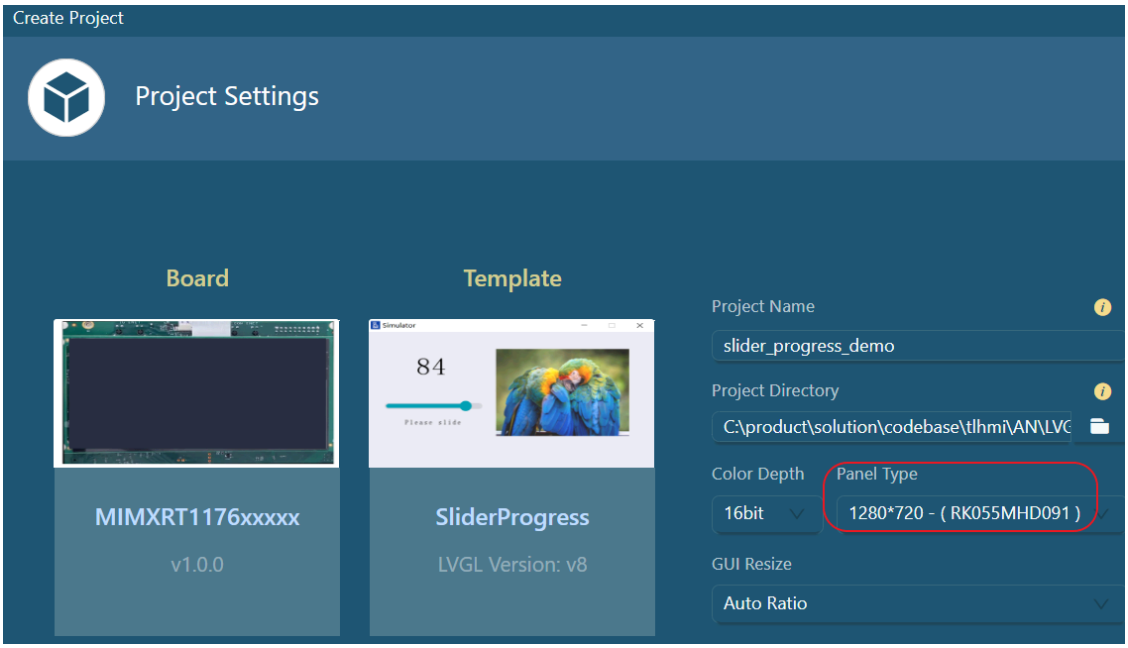

Figure 1. GUI project setup on GUI Guider

3.2 Create your project on smart HMI

Note: *First, create your project on MCUXpresso IDE.*

After the LVGL GUI example has been built, it can go to the main target to integrate it into the smart HMI software platform on the MCUXpresso project for implementing your GUI application.

The simple and quick method is to clone the current application project presented on the smart HMI platform. The elevator app is the better choice as the cloned source since it has a simple implementation.

To create your project, follow the steps below:

1. Copy and paste the "elevator" folder in the cloned smart HMI source code from GitHub. Rename it to yours. For this example, we have chosen "slider_progress", following the name of the GUI example.
2. In the "slider_progress" folder, enter the "lvgl_vglite_lib" folder containing the LVGL GUI project.
3. Open the project-related files `.cproject` and `.project` and replace all the string "elevator" with your project name string "slider_progress".
4. Do the similar replacement for both project files in the "cm4" and "cm7" folders.
   Set up your project by cloning the elevator project files.

As shown in Figure 2 your projects can now be opened in MCUXpresso IDE in the same manner as the elevator project.

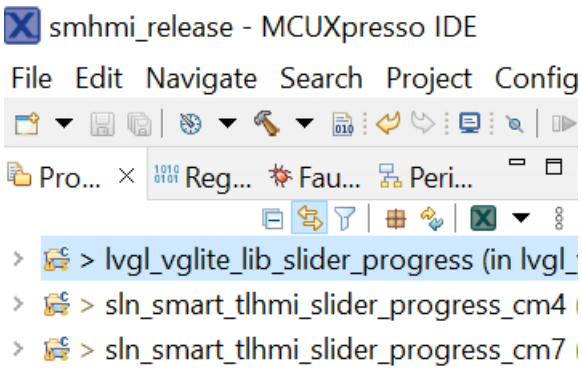

Figure 2. Projects setup on MCUXpresso

3.3 Build the resources for smart HMI

Generally, images are used in GUI (sounds used in voice prompts as well). The images and sounds are called resources, stored in a flash in sequence. Before programming them on flash, the resources should be built into a binary file. The main job is to replace the names of the reference app (elevator) with yours.

To do so, follow the steps below:

1. Delete the cloned "images" folder under `slider_progress/resource`.
2. Copy the "images" folder under `\generated` in your GUI Guider project.
3. Paste it under the `slider_progress/resource` (That is, use your own images rather than those from the elevator app.).
4. Delete the `*.mk` file used for GUI Guider in the "images" folder.
5. Rename the files `elevator_resource.txt`, `elevator_resource_build.bat`, and `elevator_resource_build.sh` in the "resource" folder to your project name `slider_progress_resource.txt`, `slider_progress_resource_build.bat`, and `slider_progress_resource_build.sh`.
   Remark:
   - `elevator_resource.txt`: containing the paths and names of all the resources (images and sounds) used in the app.
   - `elevator_resource_build.bat/elevator_resource_build.sh`: used for building the resources in Windows and Linux accordingly.
6. After opening the `slider_progress_resource.txt` file, replace all strings "elevator" with "slider_progress".
7. Remove all old images and add new ones with your image file names (here is "`_scan_example_597x460.c`"), such as `image ../../slider_progress/resource/images/_scan_example_597x460.c`.
8. Open the `slider_progress_resource.bat` file for Windows and replace all strings "elevator" with "slider_progress". Do the same to the file `slider_progress_resource.sh` for Linux.
9. Double-click the batch file `slider_progress_resource_build.bat` for Windows.
10. The command window appears and automatically runs to generate the image resource binary file containing the image data and resource access information containing C codes to set all image locations in flash and the total byte size of the images.
    After showing the message "Resource Generation Complete!", the image resource binary file named `slider_progress_resource.bin` and the resource access information file named `resource_information_table.txt` are generated in the folder "resource". The image resource binary

file is programed on flash, and the resource access information is used to access the resources on smart HMI (see Section 3.4.1).

3.4 Integrate LVGL GUI application into smart HMI

The LVGL GUI application codes (here is the SliderProgress GUI example) and the built image resources, including access information, can be added to the smart HMI. Additionally, to implement your LVGL GUI application on smart HMI, it is required to add the HAL devices related to LVGL GUI and the related configurations. The LVGL GUI application is running on the M4 core, and the related implementation is almost in the M4 project "sln_smart_tlhmi_slider_progress_cm4". The detailed steps are described in further sub sections.

3.4.1 Add LVGL GUI codes and resources

The LVGL GUI application codes used for smart HMI are in the folders "custom" and "generated" in the GUI Guider project.

To add the codes to smart HMI, follow the steps below:

1. Replace `custom.c` and `custom.h` under `slider_progress/cm4/custom/` with the ones in the folder "custom" in the GUI Guider project.
2. Remove the "generated" folders from `slider_progress/cm4/`. Then copy the "generated" folder from the GUI Guider project and paste it to `slider_progress/cm4/`.
3. Delete the folders "image" and "mPythonImages" and all the files `*.mk` and `*.py` in the "generated" folder. As mentioned above, the images in the "image" folder are built into a resource binary file, so the "image" folder is not required. The folder "mPythonImages" and all the files `*.mk` and `*.py` are unwanted for the smart HMI.
4. To add mutex control based on the smart HMI platform and set the image locations on flash, modify the file `custom.c` on MCUXpresso IDE. These are all defined by `RT_PLATFORM`.
5. Open elevator project on MCUXpresso IDE. Search the macro definition `RT_PLATFORM` in the `custom.c` under **sln_smart_tlhmi_elevator_cm4** > **custom** and copy all the code lines from `#if defined(RT_PLATFORM)` to `#endif`, and paste them in the file `custom.c` under **sln_smart_tlhmi_slider_progress_cm4** > **custom**.
6. Delete the code lines under `#else` containing `#else` since they are used for elevator GUI. The added code lines cover the following:
   - The include files are as follows:

```
#if defined(RT_PLATFORM) & LVGL_MULTITHREAD_LOCK
#include "FreeRTOS.h"
#include "semphr.h"
#include "lvgl_images_internal.h"
#endif
```

   - The variable declaration is as follows:

```
#ifdef RT_PLATFORM
/* LVGL should be single thread. Get this mutex for multithread access */
static SesaphoreHandle_t s_LVGLMutex;
#endif /*RT_PLATFORM*/
```

   - The C codes in the function `custom_init()` are as follows:

```
void custom_init(lv_ui *ui)
{
    /* Add your codes here */
    #ifdef RT_PLATFORM s_LVGLMutex = xSemaphoreCreateMutex();
    if (s_LVGLMutex == NULL)
```

```
    {
        while (1)
        {
        }
#endif
}
```

- The C codes for the functions `_takeLVGLMutex()`, `_giveLVGLMutex()`, and `setup_imgs()` where the locations of all the images are set.

7. Replace codes in the function `setup_imgs()` with the location setup codes for images in the `resource_information_table.txt` file (see Section 3.3). In this application note, there is only one image resource which is set up as: `_scan_example_597x460.data = (base + 0);`
   After doing it, the function `setup_imgs()` is shown as below:

```
void setup_imgs(unsigned char *base)
{
    _scan_example_597x460.data = (base 0);
}
#endif
```

8. To add the macro definition and function declaration related to `custom.c`, modify the `custom.h` file under **sln_smart_tlhmi_slider_progress_cm4** > **custom**, as shown below:

```
#define LVGL_MULTITHREAD_LOCK 1
void _takeLVGLMutex();
void _giveLVGLMutex();
void setup_imgs(unsigned char *base);
```

9. To define the images in your LVGL GUI application, modify the `lvgl_images_internal.h` file under **sln_smart_tlhmi_slider_progress_cm4** > **custom**.
   - Open one image `*.c` file (here is `_scan_example_597x460.c`) under `/generated/image/` in the GUI Guider project. Copy the image definition at the end of the file. Paste it to the `lvgl_images_internal.h` file after deleting all the original definitions about the images for the elevator app.
   - Delete `.data = _scan_example_597x460_map` in the array since the `.data` is set in the function `setup_imgs()`. The array is defined finally in the `lvgl_images_internal.h` file, as shown below:

```
lv_img_dsc_t _scan_example_597x460 =
{
    .header.always_zero = 0,
    .header.w = 597,
    .header.h = 460,
    .data_size = 274620 * LV_COLOR_SIZE / 8,
    .header.cf = LV_IMG_CF_TRUE_COLOR_ALPHA,
};
```

   Remark: Repeat the above operations for all image files one by one if there are multi-image files.

10. Configure the total size of the image resource by defining the macro definition `APP_LVGL_IMGS_SIZE` in the `app_config.h` file under **sln_smart_tlhmi_slider_progress_cm7** > **source** with the new size of the images. This new size is available in the built resource `resource_information_table.txt` file.

3.4.2 Add HAL devices and configurations

Based on the framework architecture, two HAL devices (display and output devices) are designed for LVGL GUI application. The implementations of the two devices are different depending on different LVGL GUI applications though there are common architecture designs for them. They are implemented separately in two files. Therefore, it must clone the two files from the present elevator application and modify your LVGL GUI application. Then, enable your devices in the configuration file. Your LVGL GUI application is built on the smart

HMI platform based on the framework. The detailed modifications can be done in the MCUXpresso IDE, as shown below:

- **Implement display HAL device**

1. Copy and paste the `hal_display_lvgl_elevator.c` file under the group **sln_smart_tlhmi_slider_progress_cm4** > **framework** > **hal** > **display** on MCUXpresso project. Rename it to `hal_display_lvgl_sliderprogress.c` for your application.
2. Open the file `hal_display_lvgl_sliderprogress.c` and replace all the strings "elevator" with your application string "SliderProgress" in the file.

- **Implement output HAL device**

1. Copy and paste the `hal_output_ui_elevator.c` file under the group **sln_smart_tlhmi_slider_progress_cm4** > **framework** > **hal** > **output** on the MCUXpresso project. Rename it to `hal_output_ui_sliderprogress.c` for your application.
2. Open the file `hal_output_ui_sliderprogress.c`. Remove all the functions related to the elevator application except the following basic common functions of the HAL device:
   ```
   HAL_OutputDev_UiElevator_Init();
   HAL_OutputDev_UiElevator_Deinit();
   HAL_OutputDev_UiElevator_Start();
   HAL_OutputDev_UiElevator_Stop();
   HAL_OutputDev_UiElevator_InferComplete();
   HAL_OutputDev_UiElevator_InputNotify();
   ```
   In addition, reserve the declarations of the below two functions:
   ```
   APP_OutputDev_UiElevator_InferCompleteDecode();
   APP_OutputDev_UiElevator_InputNotifyDecode();
   ```
3. Clean the function `HAL_OutputDev_UiElevator_InferComplete()` for building your application later. In the function, remove both function calls `_InferComplete_Vision()` and `_InferComplete_Voice()` used for handling the results from vision and voice algorithms for elevator application.
4. Clean the function `HAL_OutputDev_UiElevator_InputNotify()` and keep the basic architecture for further application development. Finally, the function looks as follows:
   ```
   static hal_output_status_t HAL_OutputDev_UiElevator_InputNotify(const
    output_dev_t *dev, void *data)
   {
       hal_output_status_t error = kStatus_HAL_OutputSuccess;
       event_base_t *pEventBase  = (event_base_t *)data;
       /* Add 'inputNotify' event handler code here */
       APP_OutputDev_UiElevator_InputNotifyDecode(pEventBase);
       return error;
   }
   ```
5. Remove all the variables declarations, including the enum and array, except the ones `s_UiSurface` and `s_AsBuffer[]` used for the common implementations.
6. Replace all strings "elevator" with your application string "SliderProgress".

- **Enable and configure both HAL devices**

1. Open the `board_define.h` file under **sln_smart_tlhmi_slider_progress_cm4** > **board**. Replace all the strings "elevator" with your application string "SliderProgress" in the file. It enables and configures the display and output HAL devices by the definitions `ENABLE_DISPLAY_DEV_LVGLSliderProgress` and `ENABLE_OUTPUT_DEV_UiSliderProgress`.
2. Open the `lvgl_support.c` file under **sln_smart_tlhmi_slider_progress_cm4** > **board**. Replace all the strings "elevator" with your application string "SliderProgress" in the file. It enables camera preview on GUI at the display driver level.

- **Register both HAL devices**

Open the M4 main `sln_smart_tlhmi_cm4.cpp` file under **sln_smart_tlhmi_slider_progress_cm4** > **source**. Replace all the strings "elevator" with your application string "SliderProgress" in the file. It registers the display and output HAL device for your application instead of the elevator application.

Therefore, the integration is completed for running the basic LVGL GUI application on smart HMI. Depending on more requirements for the application, more implementations can be added based on the integrated basic application.

4 Demonstration

The "slider_progress" application demo is implemented along with this application note.

After unzipping the demo software package, put the below files and folder into the smart HMI software:

- The file `hal_display_lvgl_sliderprpgress.c` under `[demo]\framework\hal\display\` to the path `[smart HMI]\framework\hal\display\`
- The file `hal_output_ui_slider_progress.c` under `[demo]\framework\hal\output\` to the path `[smart HMI]\framework\hal\output\`
- The folder "slider_progress" to the root path of `[smart HMI]\`

The projects can be opened on MCUXpresso IDE, just like the coffee machine/elevator app presented on the smart HMI platform. After programming the built `*.axf` file to the address 0x30100000 and the resource binary file to the address 0x30700000, the LVGL GUI demo can run successfully on the smart HMI development board (see Figure 3 for the screen display).

Note: *If using v1.7.0 of MCUXpresso IDE, enable the "Manage link script" in the* ***Setting*** *>* ***MCU C++ Linker*** *>* ***Managed Linker Script*** *before building the CM4 project.*

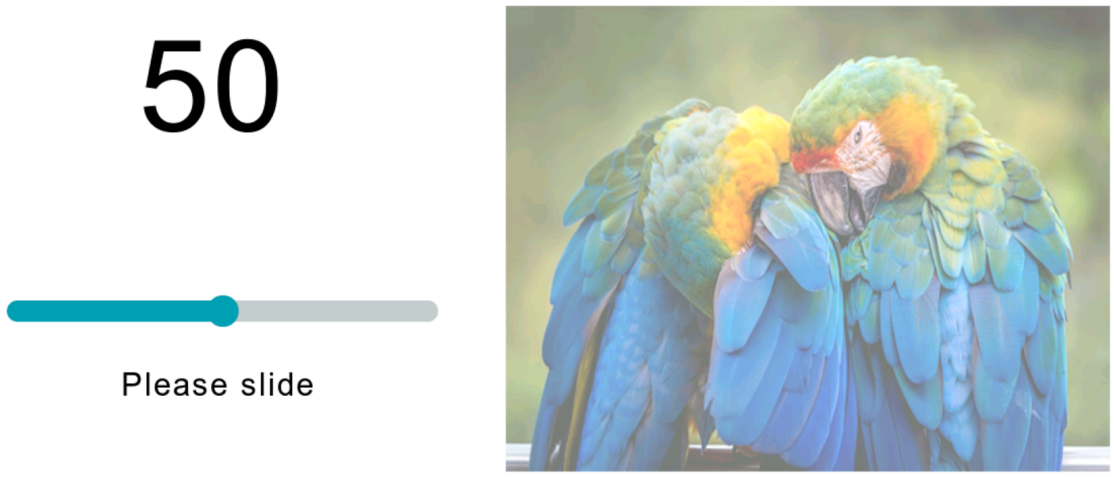

Figure 3. LVGL GUI demo display on smart HMI development board

5 Revision history

Revision history summarizes the revisions to this document.

Table 1. Revision history

Revision number	Date	Substantive changes
1	16 June 2023	Initial release

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 Legal information

7.1 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.

7.2 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 http://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 data sheet 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.

7.3 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.

i.MX — is a trademark of NXP B.V.

Contents

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

Date of release: 16 June 2023
Document identifier: AN13948