AN13730

How to Develop LVGL GUI Demo on Memory-constrained MCU with GUI Guider

Rev. 0 — 9 September 2022 Application note

Document information

Information	Content
Keywords	GUI Guider 1.3.1, LVGL, LPC55S06
Abstract	This application note introduces the use of the LVGL file system mechanism to support external SPI Flash and the use of the LVGL input device mechanism to support hardware buttons for screen switching.

1 Introduction

An attractive GUI is reliant upon well designed images and fonts. The more complex the GUI demo is, the more of these assets are required, leading to greater memory resources being consumed. If the MCU selected for a design does not have abundant on-chip Flash and on-chip RAM to store images and fonts, it means that you have to use off-chip Flash and off-chip RAM.

Fortunately, LVGL provides file system mechanism to support external storage device like SD card or serial Flash. This application note uses **LPC55S06** as the target MCU. It takes the implementation of an **E-Bike** UI as an example to introduce how to use the LVGL file system to support a low-cost external serial Flash. The external serial flash used in this application note is a Winbond **W25Q64**.

In addition to providing graphic functionality, LVGL supports an input device mechanism. This application note introduces how to use hardware buttons as LVGL input devices to achieve screen switching.

2 LPC55S06 overview

LPC55S0x/LPC550x is a family of highly cost effective Arm Cortex-M33-based micro-controllers for embedded applications and includes the following features:

- Running at a frequency of up to **96 MHz**
- TrustZone option for isolation of secure and non-secure code
- Floating Point Unit (FPU) and Memory Protection Unit (MPU)
- Up to **96 kB** of **on-chip RAM**
- Up to **256 kB** of **on-chip Flash**
- CAN-FD
- Five general-purpose timers
- SCTimer/PWM
- RTC/alarm timer
- 24-bit Multi-Rate Timer (MRT)
- Windowed Watchdog Timer (WWDT)
- Code Watchdog
- **High-speed SPI (50 MHz)**
- Eight flexible serial communication peripherals (each of which can be a USART, SPI, I2C, or I2S interface)
- 16-bit 2.0 M samples/sec ADC capable of simultaneous conversions
- Temperature sensor.

The MCU features listed above are closely related to display performance include system frequency, Flash capacity, RAM capacity, and SPI communication rate. This demo uses high-speed SPI to connect to external serial Flash.

3 LVGL overview

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

Key features:

- Open source and free to use under MIT license
- Written in C (C++ compatible) and hosted on GitHub
- More than 30 powerful, fully customizable widgets, such as, button, image button, checkbox, switch, slider, label, arc, bar, line, canvas, image, roller, slider, meter, table, text area, animation, calendar, chart, list, menu, message box, tabview
- Display of any resolution, GPU support, Multi display support
- Supports multiple types of input devices, such as:
  - Pointer-like input device like touchpad or mouse
  - Keypads like a normal keyboard or simple numeric keypad
  - Encoders with left/right turn and push options
  - External hardware buttons which are assigned to specific points on the screen
- Drawing features, such as:
  - anti-aliasing
  - shadow
  - line, arc, polygon
  - mask
- Text features, such as;
  - UTF-8 support
  - Kerning
  - word wrap and auto texts scrolling
  - Arabic and Persian support
  - font compression
  - subpixel rendering
  - online and offline font converter
  - interface for custom font engine
  - FreeType integration example
  - multi-language support
- Image features, such as:
  - various color formats: RGB, ARGB, Chroma keyed, indexed, alpha only
  - Real-time recoloring of images
  - Real-time zoom and rotation
  - Images can be stored in flash or files (such as, SD card)
  - Online and offline image converter
  - Image decoder interface for caching
  - PNG integration example
- Styles, such as:
  - Cascade styles (like in CSS)
  - Reuse the styles in multiple widgets
  - Local styles for simple changes
  - Themes to give a default appearance
  - Transitions (animations) on state change
- Micropython support
- Rich demo examples and documents
- Supported by GUI Guider, free UI design tool of NXP

For more details, see the LVGL page.

LVGL version used in this application note is **8.0.2**.

4 GUI Guider overview

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, 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 an MCUXpresso IDE, IAR Embedded Workbench, or Keil uVision project. It accelerats the development process and allows you to seamlessly add an embedded user interface to your application.

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

For more details, refer to GUI Guider.

GUI Guider version used in this application note is **1.3.1**.

5 E-bike demo overview

The E-Bike demo is a GUI application with three screens which are named as Overview, Ride 1 and Ride 2 respectively, as shown in Figure 2, Figure 3, and Figure 4. At the bottom of these screens, three buttons with labels **<**, **>**, and **^** are used to switch the current screen to the previous, next, and home screen respectively. **Overview** is the first screen displayed after the system reboot, so it is referred to the home screen from here on.

Figure 1 shows the hardware platform. It is specially customized for the E-Bike demo.

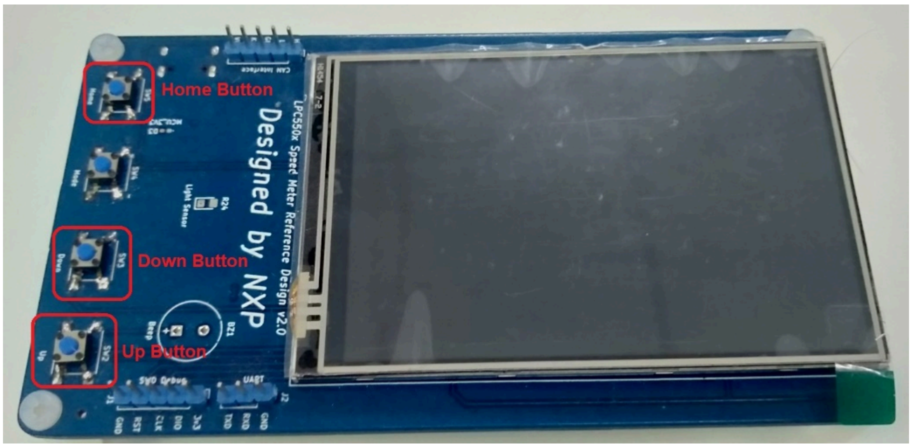


Figure 1. Hardware platform

The Home button is used to switch back to the home screen from another active screen. The Down button is used to switch to the next screen, and the Up button is used to switch to the previous screen.

For example, assuming that the current active screen is Ride 1,

- if the Down button is pressed, it switches to the Ride 2 screen.
- if the Home button or Up button is pressed, it switches to the home screen.

Figure 2. Overview (home) screen

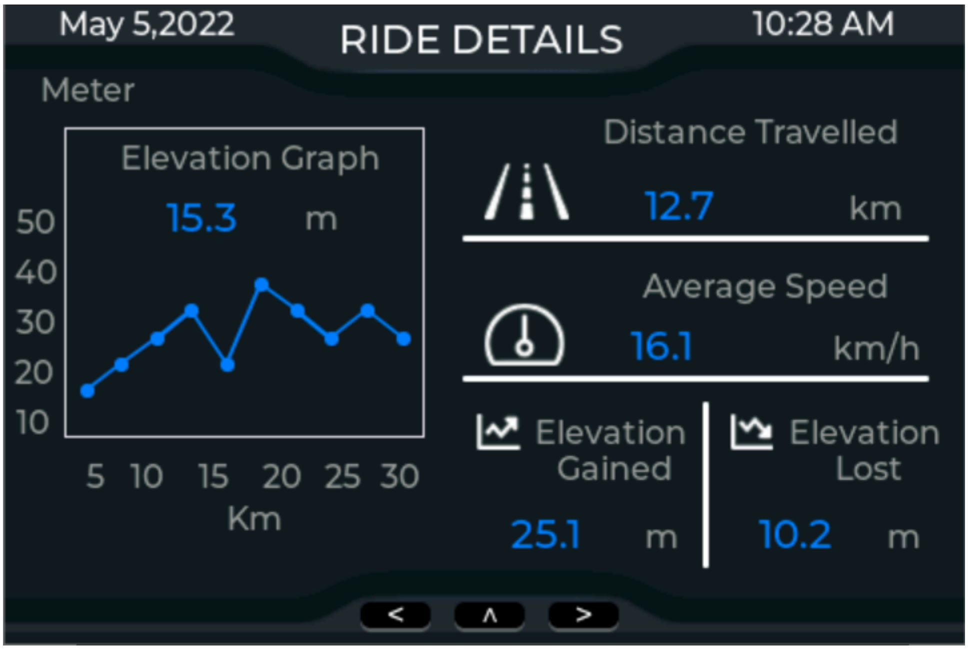


Figure 3. Ride 1 screen

Figure 4. Ride 2 screen

To learn how to use GUI Guider to perform GUI design, including creating projects, adding widgets, setting widget properties, adding events, simulating, generating code and downloading to the target board and running, see *Smart Home Demo on GUI Guider for LPC546xx* (document AN13694).

6 External serial flash support

As mentioned above, the on-chip Flash on LPC55S06 is 256 kB, which is not enough to store the image resources used in this demo. The rest of this section describes how to set up the design to use the external flash, step by step.

1. Setup the pins for the alternate function related to the high-speed SPI interface, as shown in Table 1 and Figure 5.

Table 1. Alternative function for IOs connected to external serial flash

SPI	Pin	Alternate Function
MOSI	PIO0_26	9
SSEL1	PIO1_1	5
SCK	PIO1_2	6
MISO	PIO1_3	6

2. Initialize the high-speed SPI interface with **HS_SPI_Init** function shown in Figure 5.

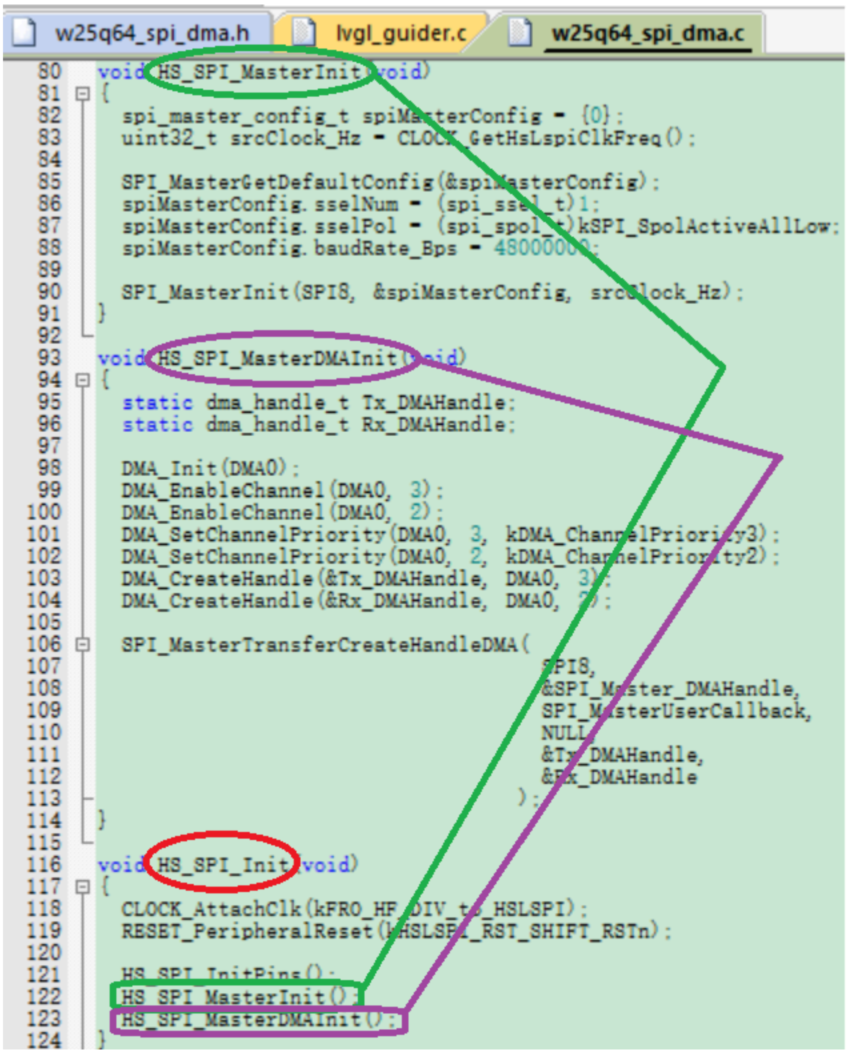


Figure 5. High-speed SPI initialization

3. Add source file and header file for driving external serial flash to the generated code project. Here we assume that you have used GUI Guider to complete the GUI page design and have generated a code project based on the Keil IDE. Figure 6 shows the directory where the driver files are stored in the code project folder. Figure 7 shows the group where the driver files are located in a Keil project.

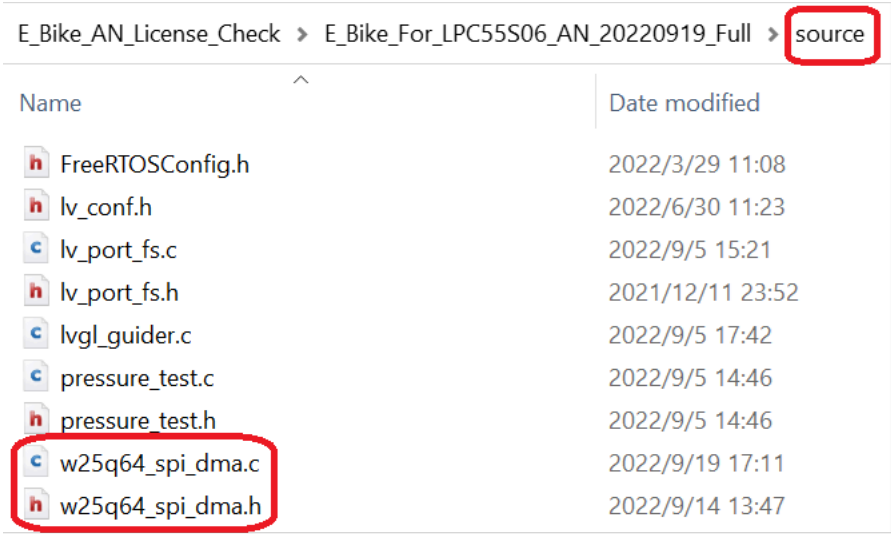


Figure 6. External serial flash driver directory

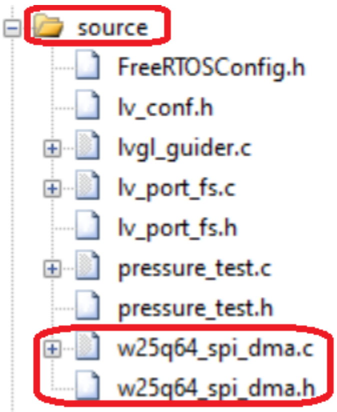


Figure 7. Keil group for external serial flash driver

4. Add a call to the initialization function for driving external serial flash to the main() function, as shown in Figure 8.

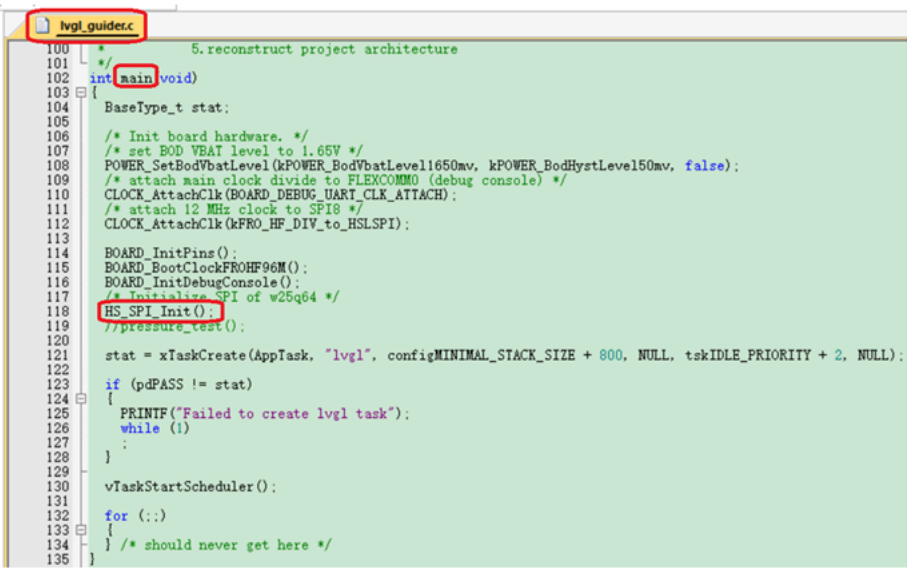


Figure 8. Call initialization function to main() function

7 External storage for image resources

This demo uses a separate and external operation for loading the images into flash. To download the image resources used in this demo to the external serial flash, prepare the image files and then download them using a debug probe (SEGGER J-Link).

1. Convert BMP, JPG, or PNG images to binary format using online image converter. The image converter is available at [Online image converter - BMP, JPG or PNG to C array or binary](https://lvgl.io/tools/imageconverter).

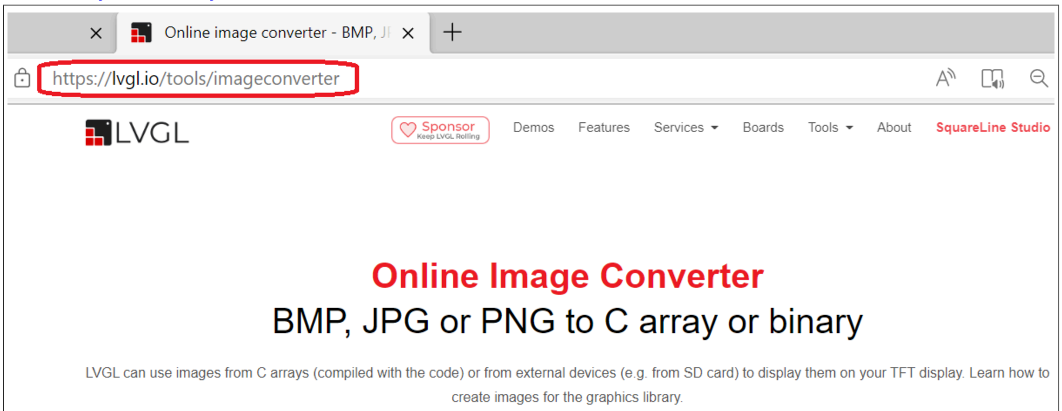


Figure 9. URL for LVGL online image converter

One key point is to select the correct color format and output format for the image converter. GUI Guider version 1.3.1 supports two color formats, as shown in Figure 10. Here, this demo selects **True Color Alpha**.

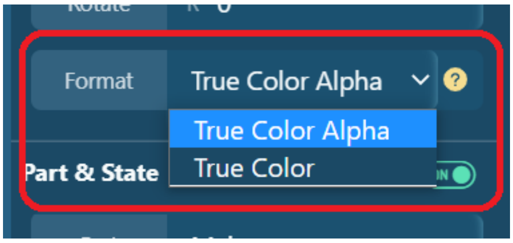


Figure 10. Color formats supported by GUI Guider

For the output format, see the configuration file called **lv_conf.h** in the code project. Here, this demo selects **16-bit color depth without swap**.

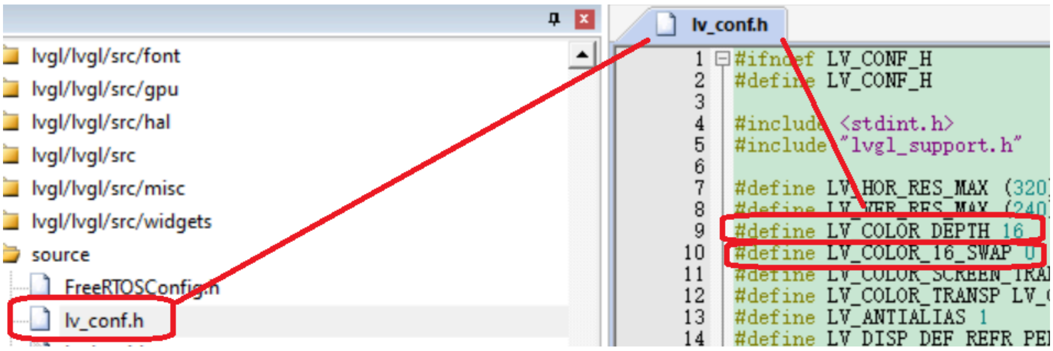


Figure 11. Color depth supported by GUI Guider

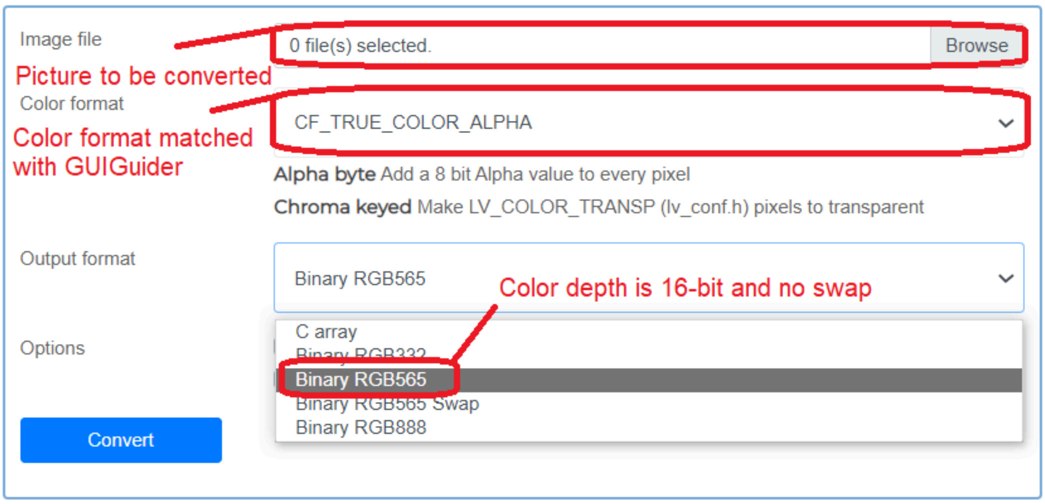


Figure 12. Color format and output format configuration for image converter

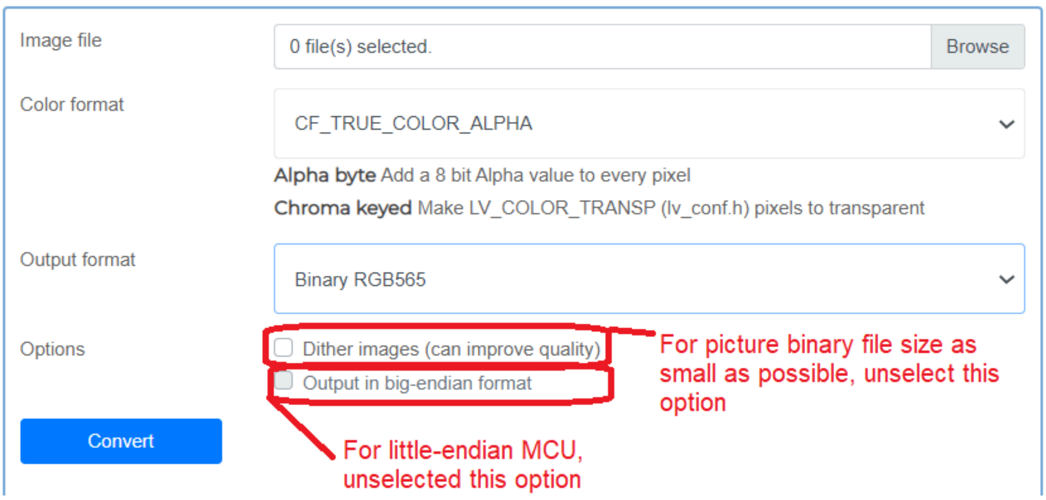


Figure 13. Dither and endian configuration for image converter

2. Merge the binary image files generated in Step 1 into a single binary file called **mergeBinFile.bin**, using binary merge tool called **MultipleBinFileMergeTool.cpp**, as shown in Figure 14.

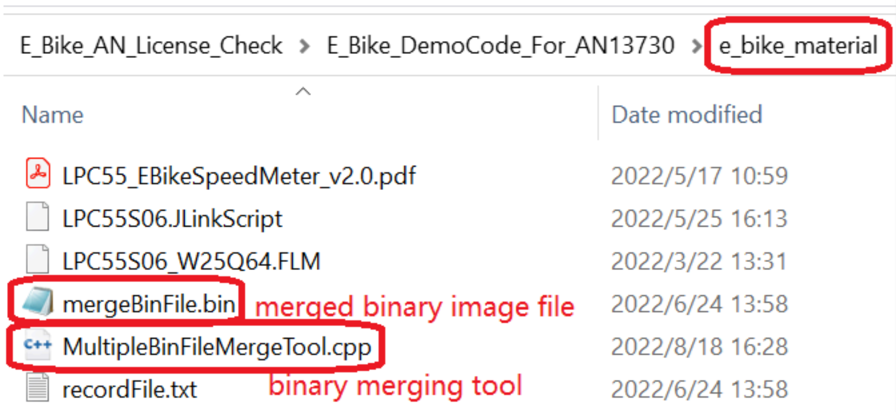


Figure 14. Merging of binary image files

3. Download the merged binary image file generated in Step 2 to the external serial Flash. To achieve this operation, create a flash driver for the J-Link and the J-Flash utility used to perform the programming operation, as described below.
   a. To program the flash memory in our design with J-Link probe, a driver or an algorithm file (called an FLM file) is required. Place the programming algorithm file for the external serial flash to the specific file directory for NXP devices in the SEGGER driver installation, as shown in Figure 15.

Figure 15. External serial flash programming algorithm file directory

b. Locate the *JLinkDevices.xml* within the J-Link installation directory. For example, the *JLinkDevices.xml* is placed at the directory shown in Figure 16. This file is used by the J-Link driver to identify all supported flash devices and to find their associated drivers.

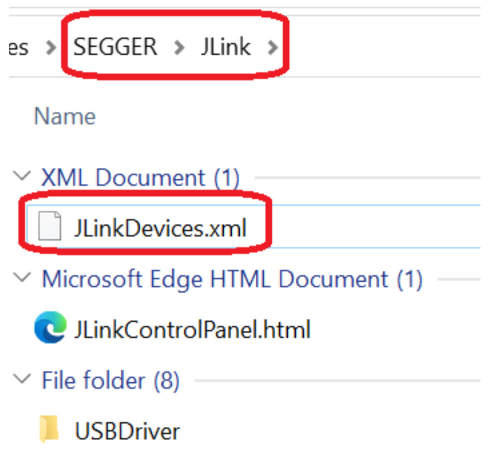


Figure 16. JLinkDevices.xml directory

Add the algorithm index entry shown below for the external serial flash that we are using at the end of the *JLinkDevices.xml* file, as shown in Figure 17. The index entry is shown below:

```
<Device>
 <ChipInfo Vendor="NXP" Name="LPC55S06_SPIFlash_W25Q64"
 WorkRAMAddr="0x20000000" WorkRAMSize="0x8000"
 Core="JLINK_CORE_CORTEX_M33" />
 <!-- MCU does not have memory mapped flash area, instead
 a virtuell address is used. -->
 <FlashBankInfo Name="EXTSPI" BaseAddr="0xC0000000"
 MaxSize="0x400000" Loader="Devices/NXP/
LPC55S06_W25Q64.FLM"
 LoaderType="FLASH_ALGO_TYPE_CMSIS" />
<Device>
```

JLinkDevices.xml

```
    <FlashBankInfo Name="Flash Block B" BaseAddr="0x11080000" MaxSize="0x40000" Loader="Devices/AnalogDevices/ADSP-CM41/CM41x_FlashB_256.FLM" Lo
  </Device>
  <Device>
  <Device>
  <Device>
  <Device>
  <Device>
  <Device>
  <Device>
  <!--                  -->
  <!-- Maxim (MAX32600) -->
  <!--                  -->
  <Device>
  <!--                 -->
  <!-- Samsung (ARTIK) -->
  <!--                 -->
  <Device>
  <!--                                      -->
  <!-- Analog Devices (Cortex-M33 devices)-->
  <!--                                      -->
  <Device>
  <!--                                      -->
  <!-- O2Micro Devices                      -->
  <!--                                      -->
  <Device>
  <Device>
  <Device>
  <Device>
  <Device>
   <ChipInfo Vendor="NXP" Name="LPC5506_externalFlash" WorkRAMAddr="0x20000000" WorkRAMSize="0x8000" Core="JLINK_CORE_CORTEX_M33" />
   <!-- MCU does not have memory mapped flash area, instead a virtuell address is used. -->
   <FlashBankInfo Name="EXTSPI" BaseAddr="0xC0000000" MaxSize="0x400000" Loader="Devices/NXP/LPC5506.FLM" LoaderType="FLASH_ALGO_TYPE_CMSIS" />
  </Device>
</DataBase>
```

Figure 17. External flash programming algorithm index location

Note: ***Loader="Devices/NXP/LPC55S06_W25Q64.FLM"*** *indicates the file path of the flash programming algorithm file. Users can modify the file name and file path according to your needs, but ensure that they are synchronized with the actual file name and file directory.*

c. Start up *J-Flash.exe* located in the directory, as shown in Figure 18. After J-Flash is started, the main interface is as shown in Figure 19.

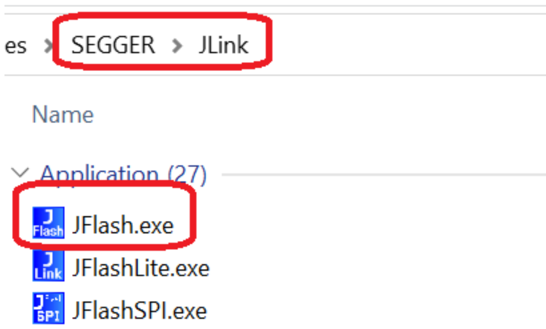


Figure 18. J-Flash directory

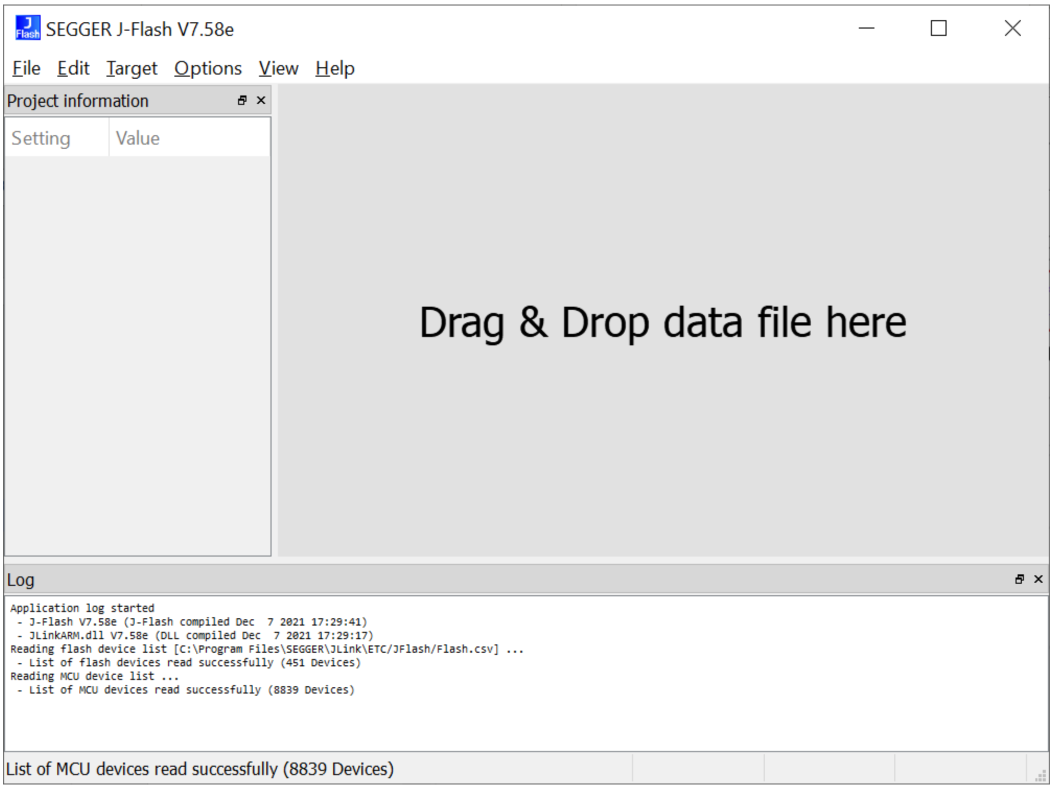


Figure 19. J-Flash main interface

d. Click **File** -> **New project**, and the **Create New Project** dialog box pops up, as shown in Figure 20. Select **Target interface** and **Speed** according to the actual situation. Here, we select **SWD** and **4000 kHz**.

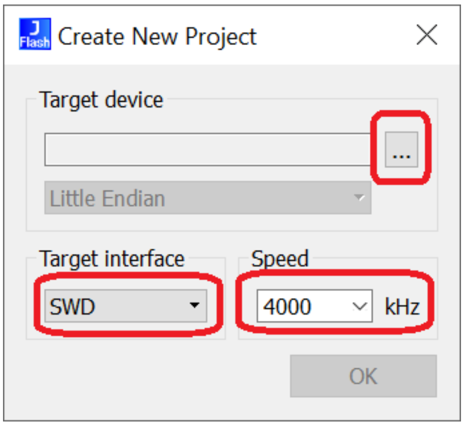


Figure 20. Create New Project dialog box

e. Click **…** button and the **Target Device Settings** dialog box pops up, as shown in Figure 21. Since the target device is LPC55S06, use **LPC55S06** as key word to search the desired target device.

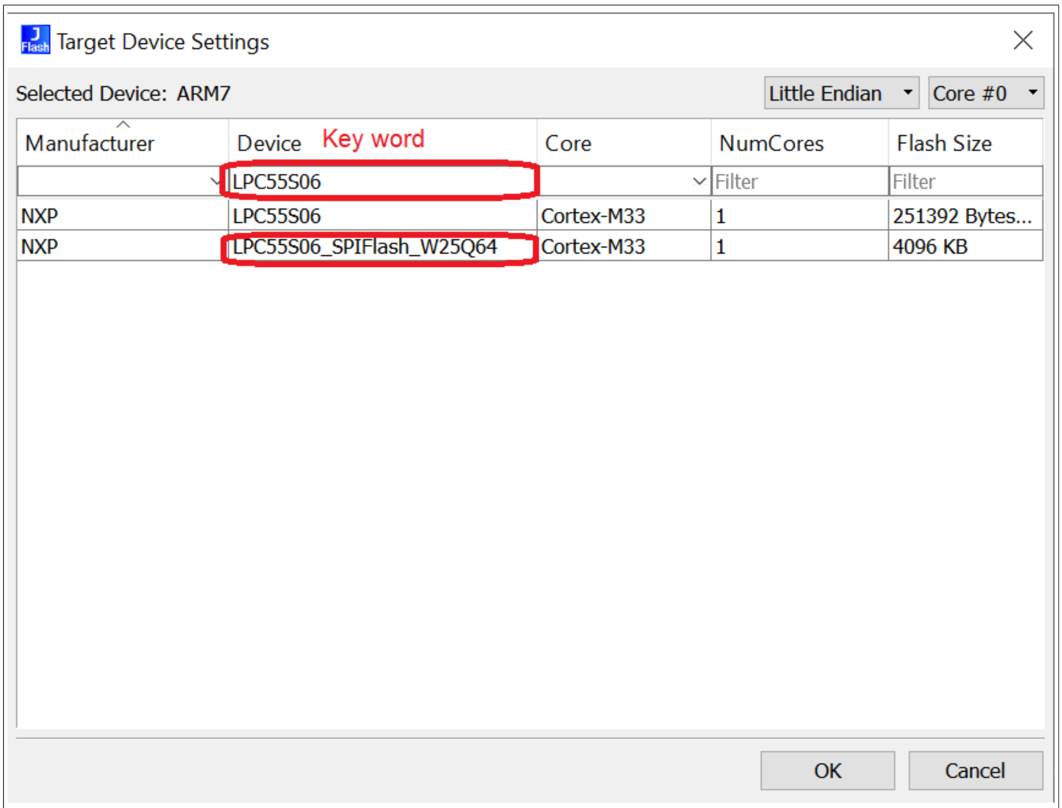


Figure 21. Target Device Settings dialog box

Select **LPC55S06_SPIFlash_W25Q64**, and it is exactly the flash programming algorithm index added in the *JLinkDevices.xml* in Step b.

f. In Figure 21, click **OK** to return Figure 20. In Figure 20, click **OK** to complete project creation, as shown in Figure 22.

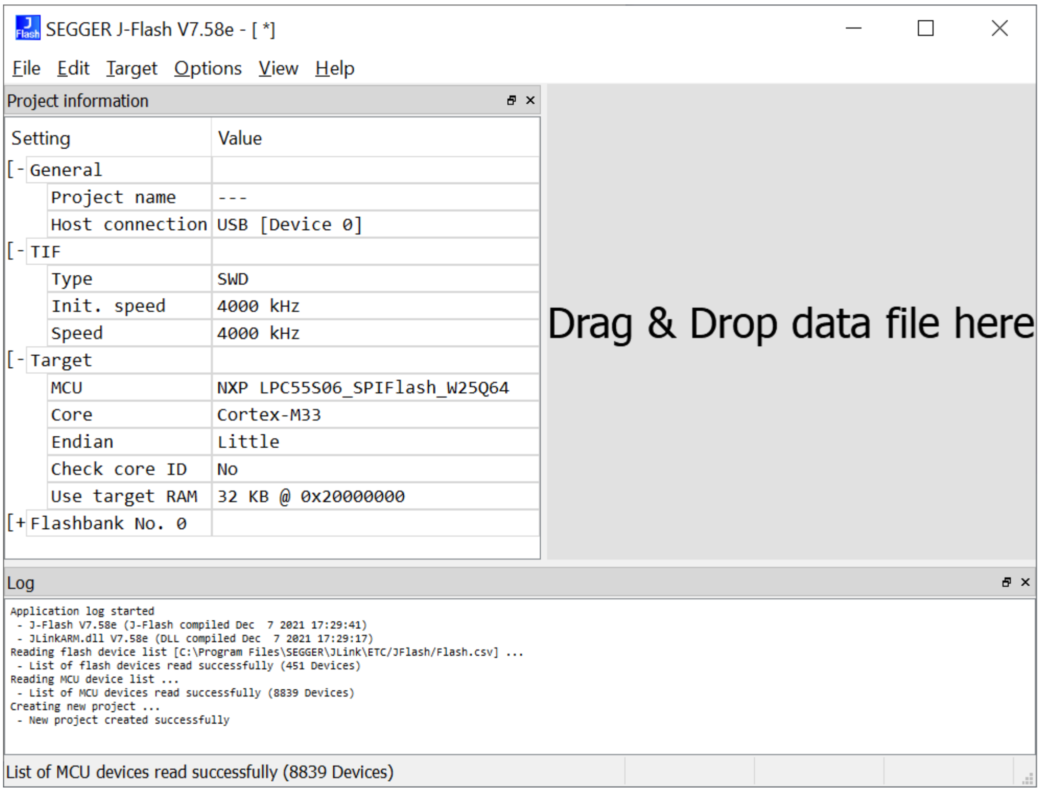


Figure 22. Created J-Flash project

g. Click **Options** -> **Project settings**, and the **Project settings** dialog box pops up, as shown in Figure 23. Click **MCU**, enable **Use J-Link script file**, and select script file for LPC55S06. This script file used in this demo is **LPC55S06.JLinkScript** and its contents are as shown in Figure 24.

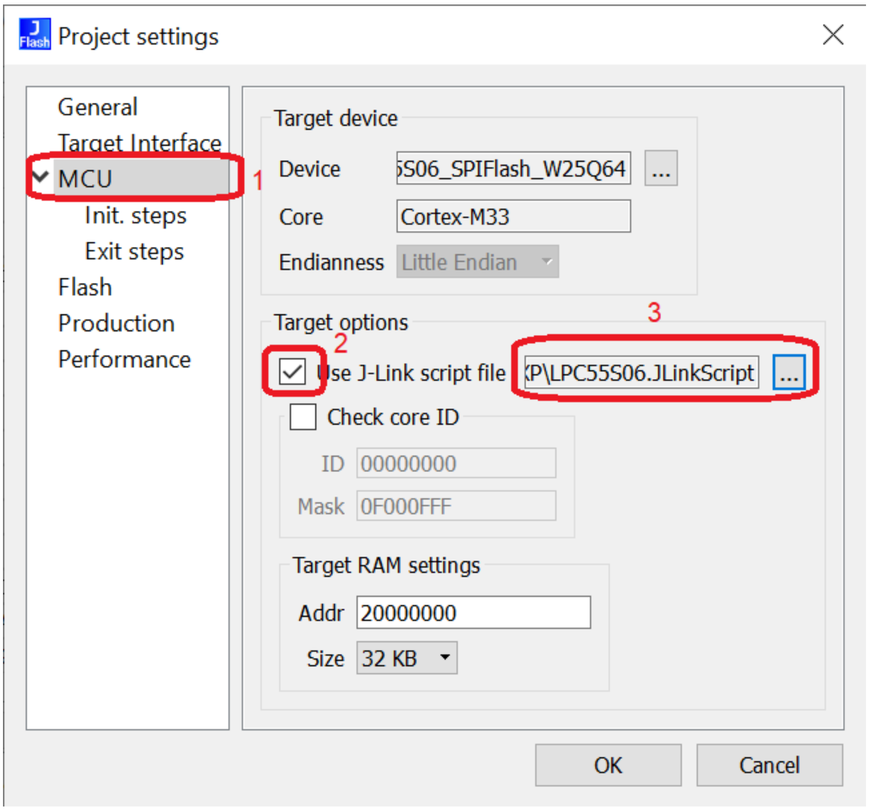


Figure 23. Use script to configure J-Flash project

LPC55S06.JLinkScript

```
*
*        ResetTarget
* This device requires a special reset as default reset does not work for blank device.
*/
// Overwrite ResetTarget for blank device download only.
void ResetTarget(void) {}


/********************************************************************
*
*        InitTarget
*/
void InitTarget(void) {
  int v;

  JLINK_SYS_Report("******************************************************");
  JLINK_SYS_Report("J-Link script: LPC55xx Cortex-M33 core J-Link script");
  JLINK_SYS_Report("******************************************************");
  JLINK_CORESIGHT_Configure("IRPre=0;DRPre=0;IRPost=0;DRPost=0;IRLenDevice=4");
  CPU = CORTEX_M33;      // Pre-select that we have a Cortex-M33 connected
  JTAG_AllowTAPReset = 0;   // J-Link is allowed to use a TAP reset for JTAG-chain auto-detection

  JTAG_SetDeviceId(0, 0x6BA02477);  // 4-bits IRLen

  // Select ISP-AP
  JLINK_CORESIGHT_WriteDP(2, 0x020000f0);
  v = JLINK_CORESIGHT_ReadAP(3);
  JLINK_SYS_Report1("DAP-IDCODE:", v);
  JLINK_CORESIGHT_WriteDP(2, 0x02000000);
  JLINK_CORESIGHT_ReadDP(0);

  // Active DebugMailbox
  JLINK_CORESIGHT_WriteAP(0, 0x21);
  JLINK_CORESIGHT_ReadAP(0);

  // Enter Debug Session
  JLINK_CORESIGHT_WriteAP(1, 0x07);
  JLINK_CORESIGHT_ReadAP(0);
}
```

Figure 24. Script file for J-Flash project setting

h. To save the created project, click **File** -> **Save project as…**.

i. To load the merged binary image file, **mergeBinFile.bin**, to J-Flash, click **File** -> **Open data file…**.

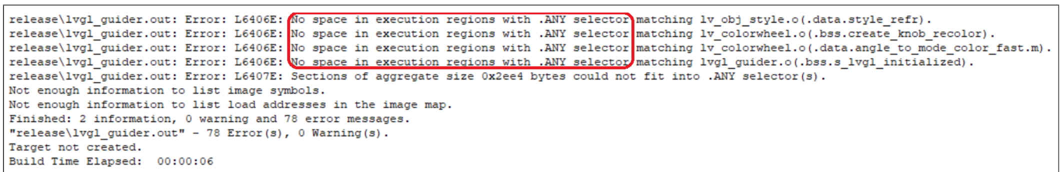
j. To establish connection between J-Flash tool on PC and the debugger on the E-Bike hardware platform, click **Target** -> **Connect**.

k. To erase the entire external serial flash, click **Target** -> **Manual Programming** -> **Erase Chip**.

l. To program the merged binary image file, **mergeBinFile.bin**, to the external serial flash on the E-Bike hardware platform, click **Target** -> **Manual Programming** -> **Program**.

8 SRAM3 enablement

When developing a GUI application, you may encounter errors shown in Figure 25.



```
release\lvgl_guider.out: Error: L6406E: No space in execution regions with .ANY selector matching lv_obj_style.o(.data.style_refr).
release\lvgl_guider.out: Error: L6406E: No space in execution regions with .ANY selector matching lv_colorwheel.o(.bss.create_knob_recolor).
release\lvgl_guider.out: Error: L6406E: No space in execution regions with .ANY selector matching lv_colorwheel.o(.data.angle_to_mode_color_fast.m).
release\lvgl_guider.out: Error: L6406E: No space in execution regions with .ANY selector matching lvgl_guider.o(.bss.s_lvgl_initialized).
release\lvgl_guider.out: Error: L6407E: Sections of aggregate size 0x2ee4 bytes could not fit into .ANY selector(s).
Not enough information to list image symbols.
Not enough information to list load addresses in the image map.
Finished: 2 information, 0 warning and 78 error messages.
"release\lvgl_guider.out" - 78 Error(s), 0 Warning(s).
Target not created.
Build Time Elapsed:  00:00:06
```

Figure 25. No space error

To solve this error, first understand the root cause of the error. The root cause of the no space error is that the Flash and/or RAM is not large enough to store the GUI application. There are two reasons for the lack of Flash and/or RAM:

- The first is that the memory resources of the MCU are not fully utilized.
- The second is that although the memory resources of the MCU are fully utilized, they are not enough to store the entire GUI application due to the small storage capacity of the selected MCU.

To judge whether a GUI application fully utilizes memory resources or not, check the files describing the allocation of memory resources, such as, scatter-loading (or linker) file of Keil IDE. This document takes **LPC55S06** as an example to explain how to judge whether the memory resources are fully utilized.

To obtain the memory resources of LPC55S06, see *LPC55S0x/LPC550x User Manual* (document UM11424), as shown in Figure 26. LPC55S06 has a total of 256 kB **on-chip Flash**, of which the system reserves **12 kB** and user applications use the remaining **244 kB**. LPC55S06 has a total of 96 kB **on-chip RAM**, including **16 kB SRAMX, 32 kB SRAM 0, 16 kB SRAM 1, 16 kB SRAM 2, and 16 kB SRAM 3**.

To create a GUI application using GUI Guider, generate a code project based on Keil IDE and then open the scatter-loading file to view the memory allocation, as shown in Figure 27. For RAM, the scatter-loading file specifies the RAM address space from `0x20000000` to `0x2000FFFF` as the data section, excluding SRAM3 (0x20010000-0x20013FFF). **The RAM address space is not fully utilized**. As for Flash, all 244 kB on-chip flash are used for code section whose size is `0x0003CE00` plus `0x00000200`. **Therefore, the Flash address space is fully utilized.**

Since we have found that the RAM is not fully utilized, we can enable SRAM 3 together with SRAM 0, 1, 2 as the data section. The available RAM space is increased from 64 kB to 80 kB. For how to enable SRAM 3, see *SRAM3 Usage in LPC55(s)06* (document AN13628).

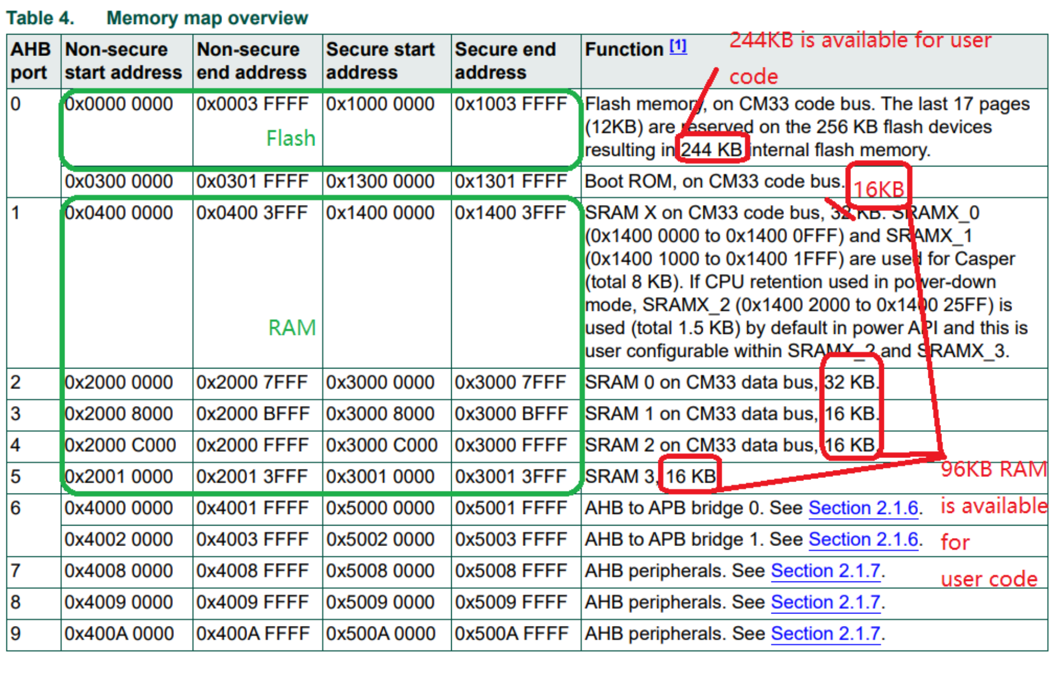
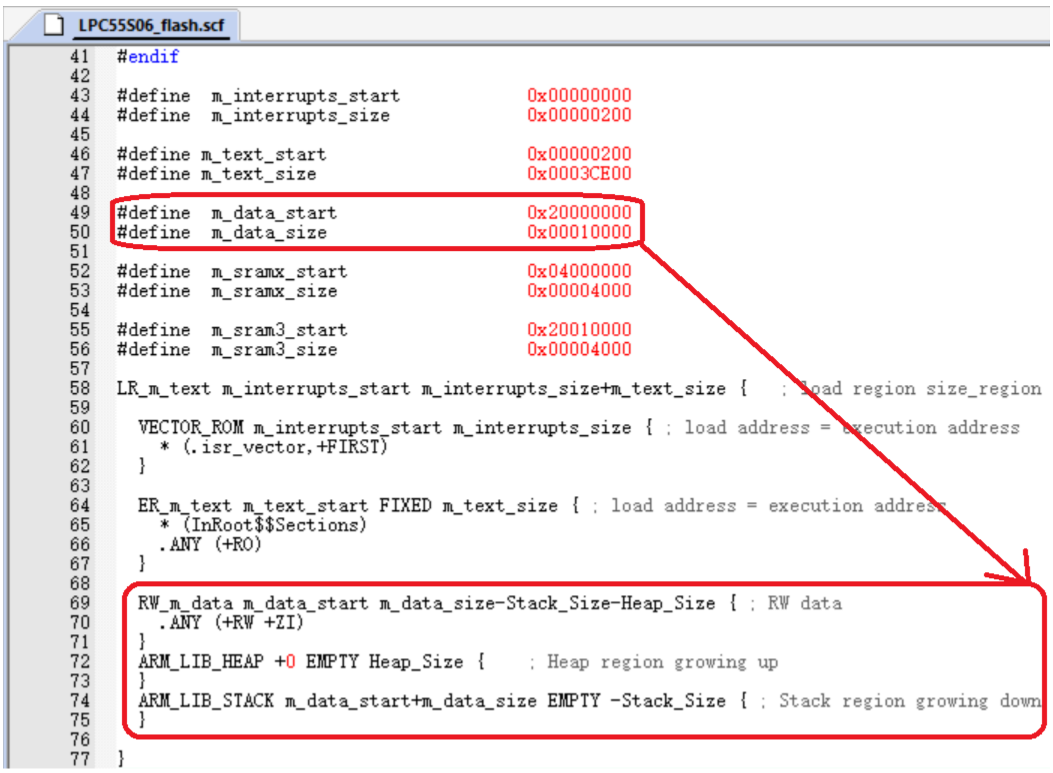


Table 4. Memory map overview

AHB port	Non-secure start address	Non-secure end address	Secure start address	Secure end address	Function [1]
0	0x0000 0000	0x0003 FFFF	0x1000 0000	0x1003 FFFF	Flash memory, on CM33 code bus. The last 17 pages (12KB) are reserved on the 256 KB flash devices resulting in 244 KB internal flash memory.
	0x0300 0000	0x0301 FFFF	0x1300 0000	0x1301 FFFF	Boot ROM, on CM33 code bus.
1	0x0400 0000	0x0400 3FFF	0x1400 0000	0x1400 3FFF	SRAM X on CM33 code bus, 32 KB. SRAMX_0 (0x1400 0000 to 0x1400 0FFF) and SRAMX_1 (0x1400 1000 to 0x1400 1FFF) are used for Casper (total 8 KB). If CPU retention used in power-down mode, SRAMX_2 (0x1400 2000 to 0x1400 25FF) is used (total 1.5 KB) by default in power API and this is user configurable within SRAMX_2 and SRAMX_3.
2	0x2000 0000	0x2000 7FFF	0x3000 0000	0x3000 7FFF	SRAM 0 on CM33 data bus, 32 KB.
3	0x2000 8000	0x2000 BFFF	0x3000 8000	0x3000 BFFF	SRAM 1 on CM33 data bus, 16 KB.
4	0x2000 C000	0x2000 FFFF	0x3000 C000	0x3000 FFFF	SRAM 2 on CM33 data bus, 16 KB.
5	0x2001 0000	0x2001 3FFF	0x3001 0000	0x3001 3FFF	SRAM 3, 16 KB
6	0x4000 0000	0x4001 FFFF	0x5000 0000	0x5001 FFFF	AHB to APB bridge 0. See Section 2.1.6.
	0x4002 0000	0x4003 FFFF	0x5002 0000	0x5003 FFFF	AHB to APB bridge 1. See Section 2.1.6.
7	0x4008 0000	0x4008 FFFF	0x5008 0000	0x5008 FFFF	AHB peripherals. See Section 2.1.7.
8	0x4009 0000	0x4009 FFFF	0x5009 0000	0x5009 FFFF	AHB peripherals. See Section 2.1.7.
9	0x400A 0000	0x400A FFFF	0x500A 0000	0x500A FFFF	AHB peripherals. See Section 2.1.7.

Figure 26. LPC55S06 memory resources



LPC55S06_flash.scf

```
#endif

#define  m_interrupts_start            0x00000000
#define  m_interrupts_size             0x00000200

#define m_text_start                   0x00000200
#define m_text_size                    0x0003CE00

#define  m_data_start                  0x20000000
#define  m_data_size                   0x00010000

#define  m_sramx_start                 0x04000000
#define  m_sramx_size                  0x00004000

#define  m_sram3_start                 0x20010000
#define  m_sram3_size                  0x00004000

LR_m_text m_interrupts_start m_interrupts_size+m_text_size {   ; load region size_region

  VECTOR_ROM m_interrupts_start m_interrupts_size { ; load address = execution address
    * (.isr_vector,+FIRST)
  }

  ER_m_text m_text_start FIXED m_text_size { ; load address = execution address
    * (InRoot$$Sections)
    .ANY (+RO)
  }

  RW_m_data m_data_start m_data_size-Stack_Size-Heap_Size { ; RW data
    .ANY (+RW +ZI)
  }
  ARM_LIB_HEAP +0 EMPTY Heap_Size {    ; Heap region growing up
  }
  ARM_LIB_STACK m_data_start+m_data_size EMPTY -Stack_Size { ; Stack region growing down
  }

}
```

Figure 27. Scatter-loading file for LPC55S06

9 Hardware button control for screen switching

This section describes how to implement screen switching using hardware buttons. LVGL supports the following types of input devices:

- Pointer-like input device, such as touchpad or mouse
- Keypads, such as, a normal keyboard or a simple numeric keypad
- Encoders with left/right turn and push options
- External hardware buttons which are assigned to specific points on the screen

To implement screen switching using hardware buttons, follow the steps below:

1. To register an input device, initialize an **lv_indev_drv_t** variable, as shown in Figure 28.

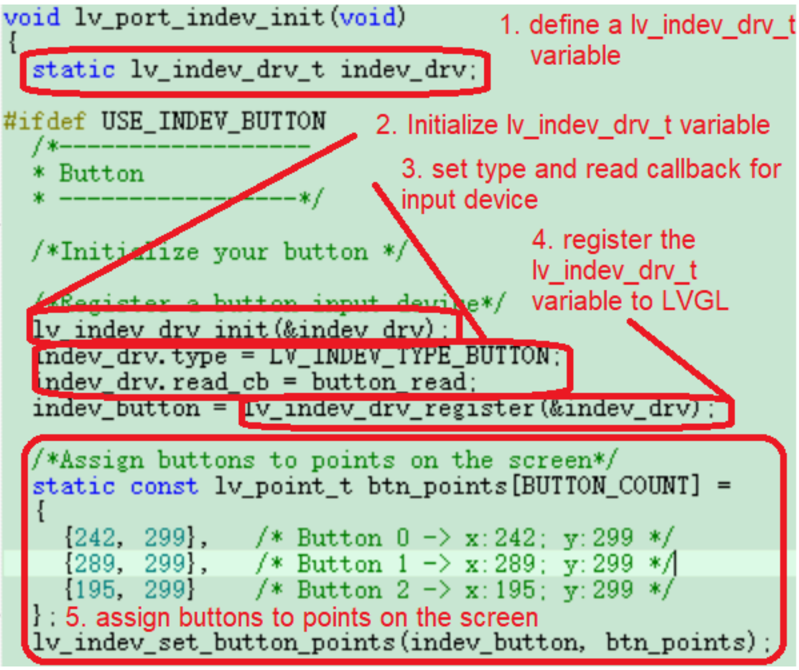


Figure 28. Register an input device

2. Implement button reading related functions, including `button_read`, `button_get_pressed_id`, and `button_is_pressed`, as shown in Figure 29, Figure 31, and Figure 32.

```
/* Will be called by the library to read the button */
static void button_read(lv_indev_drv_t * indev_drv, lv_indev_data_t * data)
{
  static uint8_t last_btn = 0;

  /* Get the pressed button's ID */
  int8_t btn_act = button_get_pressed_id();

  if(btn_act >= 0)
  {
    data->state = LV_INDEV_STATE_PR;
    last_btn = btn_act;
  }
  else
  {
    data->state = LV_INDEV_STATE_REL;
  }

  /* Save the last pressed button's ID */
  data->btn_id = last_btn;
}
```

Figure 29. Implementation of `button_read`

```c
/* Button counts */
#define BUTTON_COUNT            3
```

Figure 30. Set button quantity

```c
/* Get ID  (0, 1, 2 ..) of the pressed button */
static int8_t button_get_pressed_id(void)
{
  uint8_t i;

  /*
     Check to buttons see which is being
     pressed (assume there are 3 buttons)
  */
  for(i = 0; i < BUTTON_COUNT; i++)
  {
    /* Return the pressed button's ID */
    if(button_is_pressed(i))
    {
      return i;
    }
  }

  /* No button pressed */
  return -1;
}
```

Figure 31. Implementation of `button_get_pressed_id`

```
/*Test if `id` button is pressed or not*/
static bool button_is_pressed(uint8_t id)
{
  /*Your code comes here*/
  switch(id)
  {
    case 0:
    {
      if(GPIO_PinRead(GPIO, 1, 23) == 0)
      {/* Home */
        return true;
      }
    }
    break;
    case 1:
    {
      if(GPIO_PinRead(GPIO, 1, 5) == 0)
      {/* Down */
        return true;
      }
    }
    break;
    case 2:
    {
      if(GPIO_PinRead(GPIO, 1, 21) == 0)
      {/* Up */
        return true;
      }
    }
    break;
    default:
    {
    }
    break;
  }

  return false;
}
```

Figure 32. Implementation of `button_is_pressed`

For more details for inputting device, see input device in LVGL.

10 File system support for external serial Flash

LVGL has a **File system** abstraction module that enables you to attach any type of file system. Here, we build a simple file system for external serial flash, which makes it possible to operate image files stored on external serial Flash through file API functions. For more details about file system in LVGL, see File System in LVGL.

10.1 Get file system template file and add them to the code project

For fast file system porting, LVGL provides a file system template file. First, clone the LVGL graphics library located on Git hub. The Git hub link for the LVGL graphics library is here.

The LVGL file system template file is *lv_port_fs_template.c* and the corresponding header file is *lv_port_fs_template.h*. The directory of these two files is shown in Figure 33.

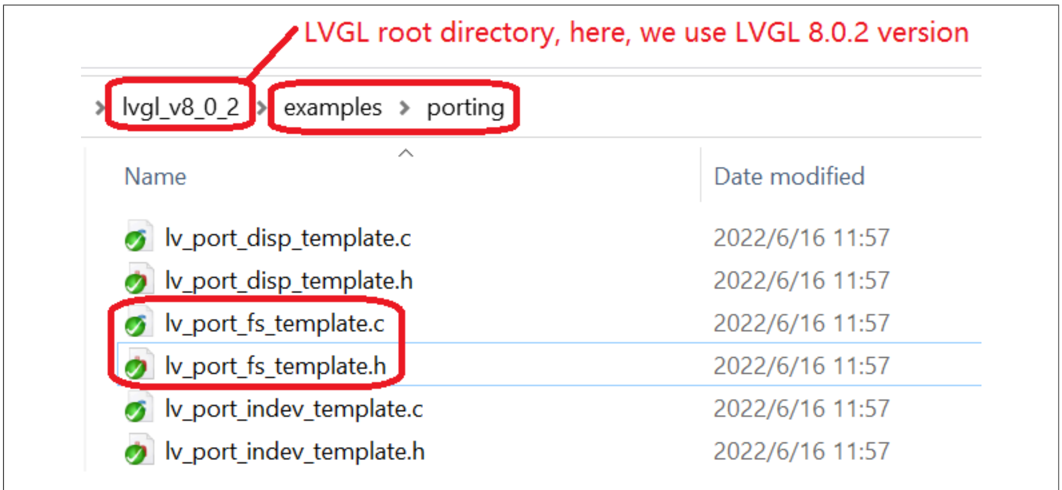


Figure 33. File system template file directory

Copy these two files to the code project directory and rename them to `lv_port_fs.c` and `lv_port_fs.h`, as shown in Figure 34.

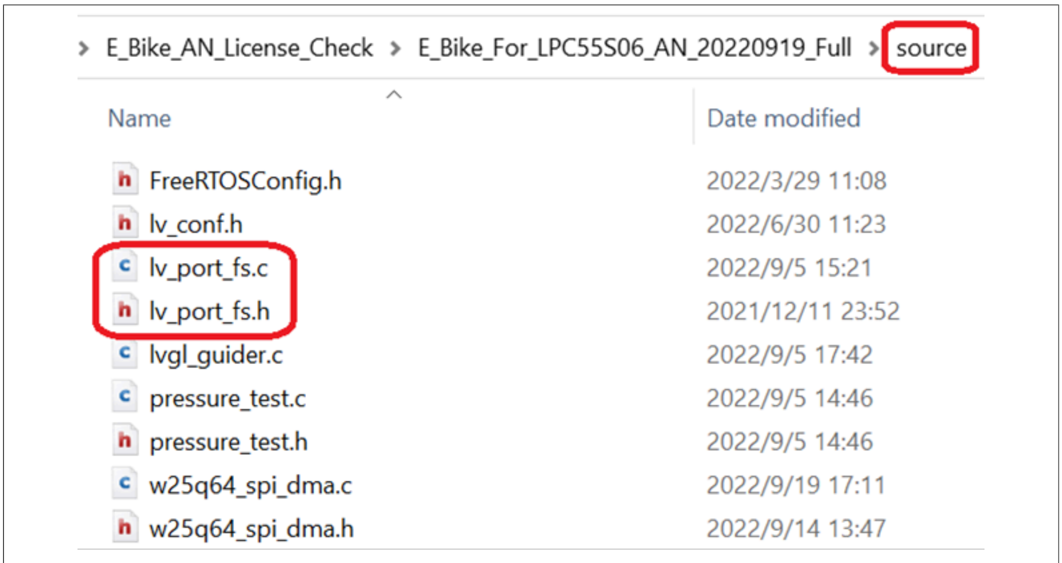


Figure 34. File system porting files directory

Add `lv_port_fs.c` and `lv_port_fs.h` to the source group of the Keil project, as shown in Figure 35.

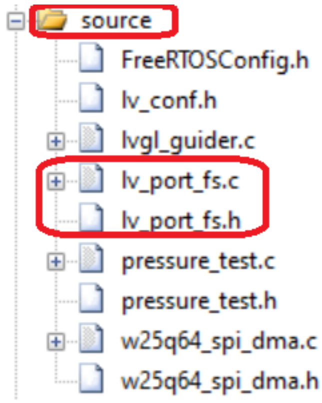


Figure 35. Add File system porting files to Keil project

10.2 Implement file operation functions

To implement file operation functions, see the *lv_port_fs.c* file in the attached code project. Noteworthy file operation functions include **lv_port_fs_init**, **fs_init**, **fs_open**, **fs_close**, **fs_read**, **fs_seek**, **fs_tell**, and **fs_size**.

Here we focus on the **fs_open** function. The **fs_open** function consists of several **if** statements, each **if** statement corresponds to a binary image file, as shown in Figure 36.

```
static void* fs_open(struct _lv_fs_drv_t * drv, const char * path, lv_fs_mode_t mode)
{
  lv_fs_res_t res = LV_FS_RES_NOT_IMP;
  uint32_t* file_p = NULL;

  if (mode == LV_FS_MODE_WR)
  {
  else if (mode == LV_FS_MODE_RD)
  {
  else if (mode == (LV_FS_MODE_WR | LV_FS_MODE_RD))
  {
  /* For Read SPI Flash directly without FileSystem */
  /* ebike_bg.bin */
  if (0 == strcmp(path, "/ebike_bg.bin"))
  {
  /* ebike_header_bg.bin */
  if (0 == strcmp(path, "/ebike_header_bg.bin"))
  {
  /* ebike_gps_arrow.bin */
  if (0 == strcmp(path, "/ebike_gps_arrow.bin"))
  {
    //The location addr of the image in the flash
    FIL.base_addr = 534248;
    //Always 0 at this moment
    FIL.offset = 0;
    //The size of the image
    FIL.size = 4036;
    file_p = (uint32_t*)&FIL;
  }
```

Figure 36. Part of implementation of fs_open

As shown in Figure 36, **ebike_gps_arrow.bin** is the binary image file used in this demo. The **base_addr** and **offset** are the base address and address offset in the external serial flash which stores ebike_gps_arrow.bin. Here, **ebike_gps_arrow.bin** is stored at address 534248, so we can specify the **base_addr as 534248** and the **offset as 0**. The **size** is the file size of **ebike_gps_arrow.bin**. Here, it is 4036 bytes.

That means, if you want to add a new image to the GUI application, convert the image file to binary format, download it to the external serial flash, and add an **if** statement. In this **if** statement, specify the base address, the offset, and the file size.

To display the image file which has been stored in external serial flash, use the code shown in Figure 37 to load image data from the external flash to LVGL.

```
//Write codes Overview_image_GPS_icon
ui->Overview_image_GPS_icon = lv_img_create(ui->Overview);
lv_obj_set_pos(ui->Overview_image_GPS_icon, 274, 60);
lv_obj_set_size(ui->Overview_image_GPS_icon, 28, 48);

//Write style state: LV_STATE_DEFAULT for style_overview_image_gps_icon
static lv_style_t style_overview_image_gps_icon_main_main_default;
if (style_overview_image_gps_icon_main_main_default.prop_cnt > 1)
  lv_style_reset(&style_overview_image_gps_icon_main_main_default);
else
  lv_style_init(&style_overview_image_gps_icon_main_main_default);
lv_style_set_img_recolor(&style_overview_image_gps_icon_main_main_defau
lv_style_set_img_recolor_opa(&style_overview_image_gps_icon_main_main_d
lv_style_set_img_opa(&style_overview_image_gps_icon_main_main_default,
lv_obj_add_style(ui->Overview_image_GPS_icon, &style_overview_image_gps
lv_obj_add_flag(ui->Overview_image_GPS_icon, LV_OBJ_FLAG_CLICKABLE);
lv_img_set_src(ui->Overview_image_GPS_icon,"F:/ebike_gps_arrow.bin");
lv_img_set_pivot(ui->Overview_image_GPS_icon, 0,0);
lv_img_set_angle(ui->Overview_image_GPS_icon, 0);
```

Figure 37. Load image data in external serial flash to LVGL

11 Summary

This application note focuses on the application of LVGL and GUI Guider on memory-constrained MCU, including external serial flash support, storing images to external serial flash, making full use of memory resources, using hardware buttons to switch screen, and file system support.

The information related to this demo, including code, images, and image merging tools, are available together with this application note.

Note: *The J-Link loading is same for IAR and MCUXpresso IDE except for differences around linker file. For MCUXpresso IDE, linker file can be a little more difficult, and for IAR, it should be straightforward though.*

12 Legal information

12.1 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.

12.2 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.

12.3 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.

Contents

Date of release: 9 September 2022
Document identifier: AN13730