GigaDevice Semiconductor Inc.

LVGL porting guide based on GD32VW55x

Application Note
AN318

Revision 1.0

(Feb. 2026)

Table of Contents

List of Figures

List of Tables

1. Preface

LVGL is an open source and free graphics library. It provides the creation of embedded GUI everything needed, with easy-to-use graphical elements, beautiful visual effects, and low memory usage. For detailed introduction, refer to **https://github.com/lvgl/lvgl**.

Applicable Product: GD32VW55x Series.

Note: This application note is for reference only. In case of any conflict with the user manual or datasheet, the user manual or datasheet shall prevail.

2. Development Environment

The development environment is introduced as follows:

- Hardware Development Board: GD32VW553H-EVAL-V1.2.
- Chip Type: GD32VW553HMQ7.
- Operating System: Win10-64 bit.
- Development Environment: GD32EmbeddedBuilder_v1.5.4.
- Firmware Library: GD32VW55x_Firmware_Library_V1.4.0.
- LVGL: LVGL 9.2.2.

3. Project Configuration

3.1. Obtain LVGL and SDK Source Code

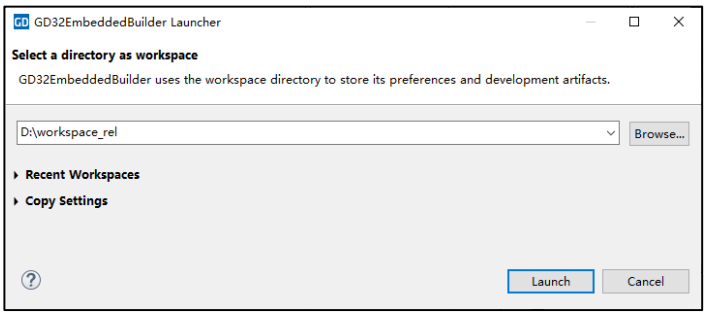
The download address of the LVGL graphics library is: https://github.com/lvgl/lvgl. This manual uses LVGL 9.2.2 version.

Download GD32VW55x SDK Official firmware development package GD32VW55x_Firmware_Library. The download address is: https://www.gd32mcu.com/cn/download/0?kw=GD32VW5. The firmware version adopted in this manual is V1.4.0.

3.2. GD32EmbeddedBuilder porting steps

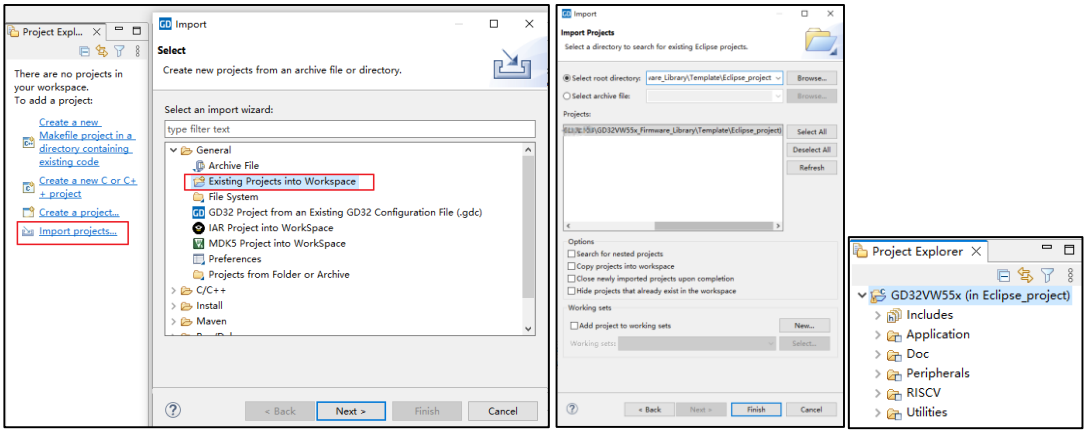
1. Open GD32EmbeddedBuilder and select a workspace. The workspace path should not be too long.

Figure 3-1. Select IDE Workspace



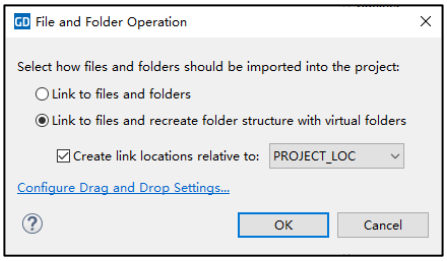
2. Unzip the downloaded GD32VW55x_Firmware_Library firmware package to the local directory. The project directory path should not be too long. Import the template project from GD32VW55x_Firmware_Library by clicking 'Import project' -> 'Existing Projects into Workspace', select the GD32VW55x_Firmware_Library/Template/Eclipse_project path. The IDE will automatically recognize the Eclipse_project and import it into the current workspace.

Figure 3-2. Import template project



3. Unzip the downloaded LVGL to GD32VW55x_Firmware_Library\Utilities\Third_Party directory, copy lv_conf_template.h and rename it to lv_conf.h (the LVGL configuration file). Add required LVGL files to the IDE by dragging the Third_Party folder to the Utilities directory in the IDE as a virtual folder:

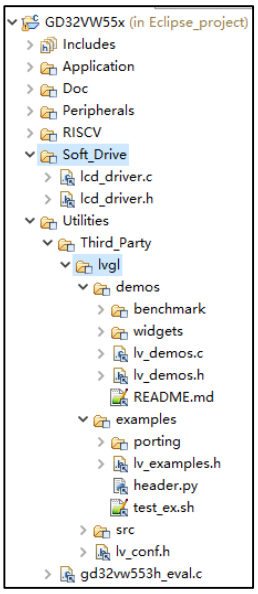
Figure 3-3. Import Template Project



Delete the lvgl/test folder, all folders in lvgl/demos except benchmark and widgets, all folders in lvgl/examples except porting, and unrelated files in the lvgl directory. The IDE does not need to compile them. This deletion will not remove files from the disk.

Add the LCD driver files in the same way by dragging the Soft_Drive folder to the IDE project. The final file directory in the IDE is shown as follows.

Figure 3-4. Import LVGL and LCD driver files

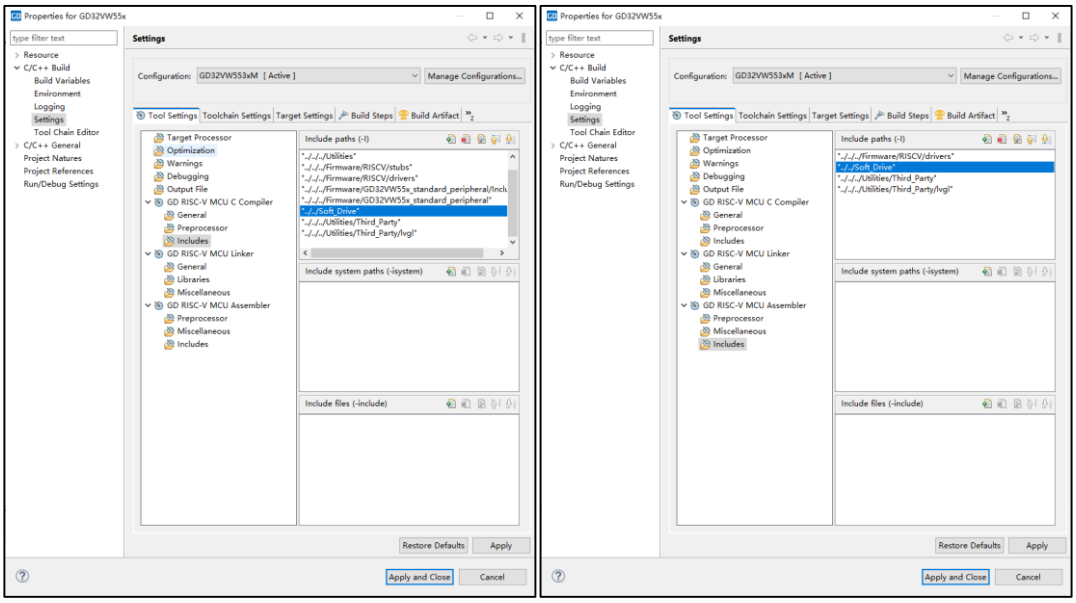


4. Add header file paths in project properties under Setting/Tool Settings/GD RISC-V MCU C Compiler/Include and Setting/Tool Settings/GD RISC-V MCU C Assembler/Include

Table 3-1. Header file paths

../../Soft_Drive
../../../Utilities/Third_Party
../../../Utilities/Third_Party/lvgl

Figure 3-5. Add header file paths



4. Code adaptation

4.1. Modify LVGL configuration file

After adding the relevant LVGL files, modify the lv_conf.h file as follows:

1. Enable this configuration file;
2. The color format should be consistent with the LCD driver. This article uses RGB565 format: #define LV_COLOR_DEPTH 16;
3. Optional configurations to ensure subsequent demos run properly: memory size, 24-point font, system monitor, memory monitor, widgets demo, benchmark demo;

Table 4-1. LVGL Configuration

```
/* clang-format off */
#if 1 /*Set it to "1" to enable content*/
#define LV_COLOR_DEPTH 16
...
/* optional configration */
#define LV_MEM_SIZE (128 * 1024U)          /*[bytes]*/
#define LV_FONT_MONTSERRAT_24 1
#define LV_USE_SYSMON     1
#define LV_USE_PERF_MONITOR 1
#define LV_USE_MEM_MONITOR 1
#define LV_USE_DEMO_WIDGETS 1
#define LV_USE_DEMO_BENCHMARK 1
```

4.2. Porting LVGL display driver

The LVGL display driver files mainly provide LCD parameter definitions, drawing buffer, and refresh callback interfaces for LVGL. Rename lvgl\examples\porting\lv_port_disp_template.c(.h) to lv_prot_disp.c(.h), and modify the code to achieve the following functions:

1. Enable the display driver file;
2. Define the screen resolution.

Table 4-2. lv_prot_disp.h source code

```
/* copy this file as "lv_port_disp.h" and set this value to "1" to enable content */
#if 1
...
#include "gd32vw55x.h"
#define XSIZE_PHYS 320
#define YSIZE_PHYS 240
```

1. Enable the display driver file;
2. lv_port_disp_init function: register drawing buffer, drawing callback function, and screen resolution;
3. disp_init function: call lcd_init function to initialize the LCD;
4. disp_flush function: use the lcd_copy function to call DMA or software to refresh the drawing area data to the corresponding position on the LCD; single or double buffering mode can be used;
5. lcd_copy function: supports both DMA and software refresh methods. SPI uses 8-bit mode to send refresh address commands, then switches to 16-bit mode to send image data for higher transfer efficiency.

Table 4-3. lv_prot_disp.c source code

```
/*Copy this file as "lv_port_disp.c" and set this value to "1" to enable content*/
#if 1
#include "lv_port_disp.h"
#include <stdbool.h>
#include "lcd_driver.h"
#define MY_DISP_HOR_RES    XSIZE_PHYS
#define MY_DISP_VER_RES    YSIZE_PHYS
#define LV_BUF_TYPE 2
#if(LV_BUF_TYPE==1)
    uint16_t buf_2[320 * 120];
#elif(LV_BUF_TYPE==2)
    uint16_t buf_2[320 * 60];
    uint16_t buf_3[320 * 60];
#endif
lv_display_t *g_disp_drv = NULL;
#define BYTE_PER_PIXEL (LV_COLOR_FORMAT_GET_SIZE(LV_COLOR_FORMAT_RGB565))
/*will be 2 for RGB565 */
static void disp_init(void);
static void disp_flush(lv_display_t * disp, const lv_area_t * area, uint8_t * px_map);

void lv_port_disp_init(void)
{
    disp_init();
    lv_display_t * disp = lv_display_create(MY_DISP_HOR_RES, MY_DISP_VER_RES);
    lv_display_set_flush_cb(disp, disp_flush);
#if(LV_BUF_TYPE==1)
    /* Example 1 * One buffer for partial rendering*/
    lv_display_set_buffers(disp, buf_2, NULL, sizeof(buf_2),
LV_DISPLAY_RENDER_MODE_PARTIAL);
#elif(LV_BUF_TYPE==2)
    /* Example 2* Two buffers for partial rendering* In flush_cb DMA or similar hardware should be
used to update the display in the background.*/
```

```
    lv_display_set_buffers(disp, buf_2, buf_3, sizeof(buf_2),
LV_DISPLAY_RENDER_MODE_PARTIAL);
#else
    /* Example 3
     * Two buffers screen sized buffer for double buffering.
     * Both LV_DISPLAY_RENDER_MODE_DIRECT and LV_DISPLAY_RENDER_MODE_FULL
works, see their comments*/
    lv_display_set_buffers(disp, buf_1, buf_2, MY_DISP_HOR_RES * MY_DISP_VER_RES * 2,
LV_DISPLAY_RENDER_MODE_DIRECT);
#endif
}
/*Initialize your display and the required peripherals.*/
static void disp_init(void)
{
    /* initialize LCD */
    lcd_init();
}
...
/*!   \brief        copy image data to LCD
    \param[in]    x: the x position of the start point
    \param[in]    y: the y position of the start point
    \param[in]    w: width of the image
    \param[in]    h: height of the image
    \param[in]    src_data: pointer to the source image data
    \param[out] none
    \retval       none
*/
void lcd_copy(uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint16_t *src_data)
{
    uint16_t *data;
    data = src_data;
#if (USE_DMA_LCD==1)
    /* SPI_FRAMESIZE_8BIT */
    spi_disable();
    SPI_CTL0 &= ~SPI_CTL0_FF16;
    spi_enable();
    /* Set the refresh area */
    lcd_set_region(x, y, x + w - 1, y + h - 1);
    /* SPI_FRAMESIZE_16BIT */
    spi_disable();
    SPI_CTL0 |= SPI_CTL0_FF16;
    spi_enable();
    /* Send refresh data */
```

```
    LCD_RS_SET;
    LCD_CS_CLR;
    dma_flag_clear(DMA_CH2, DMA_INT_FLAG_FTF);
    dma_flag_clear(DMA_CH3, DMA_INT_FLAG_FTF);
    dma_memory_address_generation_config(DMA_CH3,
DMA_MEMORY_INCREASE_ENABLE);
    dma_memory_address_config(DMA_CH3, DMA_MEMORY_0, (uint32_t)src_data);
    dma_transfer_number_config(DMA_CH3, w * h);
    dma_transfer_number_config(DMA_CH2, w * h);
    dma_channel_enable(DMA_CH2);
    dma_channel_enable(DMA_CH3);
#else
    uint16_t i, m;
    /* SPI_FRAMESIZE_8BIT */
    spi_disable();
    SPI_CTL0 &= ~SPI_CTL0_FF16;
    spi_enable();
    /* Set the refresh area */
    lcd_set_region(x, y, x + w - 1, y + h - 1);
    /* SPI_FRAMESIZE_16BIT */
    spi_disable();
    SPI_CTL0 |= SPI_CTL0_FF16;
    spi_enable();
    /* Send refresh data */
    LCD_RS_SET;
    LCD_CS_CLR;
    for(i = 0; i < h; i ++) {
        for(m = 0; m < w; m ++) {
            spi_write_halfword(*data);
            data++;
        }
    }
    LCD_CS_SET;
#endif
}

/*Flush the content of the internal buffer the specific area on the display.
 *`px_map` contains the rendered image as raw pixel map and it should be copied to `area` on the
display.
 *You can use DMA or any hardware acceleration to do this operation in the background but
 *'lv_display_flush_ready()' has to be called when it's finished.*/
static void disp_flush(lv_display_t * disp_drv, const lv_area_t * area, uint8_t * px_map)
{
```

```
    if(disp_flush_enabled) {
#if (USE_DMA_LCD==1)
        while(g_disp_drv != NULL);
        g_disp_drv = disp_drv;
        lcd_copy(area->x1, area->y1, area->x2 - area->x1 + 1, area->y2 - area->y1 + 1, (uint16_t
*)px_map);
#else
        lcd_copy(area->x1, area->y1, area->x2 - area->x1 + 1, area->y2 - area->y1 + 1, (uint16_t
*)px_map);
        lv_display_flush_ready(disp_drv);
#endif
    }
}

#if (USE_DMA_LCD==1)
void DMA_Channel2_IRQHandler(void)
{
    if(SET == dma_interrupt_flag_get(DMA_CH2, DMA_INT_FLAG_FTF)) {
        dma_interrupt_flag_clear(DMA_CH2, DMA_INT_FLAG_FTF);
    }
}
void DMA_Channel3_IRQHandler(void)
{
    if(SET == dma_interrupt_flag_get(DMA_CH3, DMA_INT_FLAG_FTF)) {
        dma_interrupt_flag_clear(DMA_CH3, DMA_INT_FLAG_FTF);
        /* infor lvgl */
        if(g_disp_drv != NULL) {
            LCD_CS_SET;
            lv_display_flush_ready(g_disp_drv);
            g_disp_drv = NULL;
        }
    }
}
#endif
#else /*Enable this file at the top*/
/*This dummy typedef exists purely to silence -Wpedantic.*/
typedef int keep_pedantic_happy;
#endif
```

4.3. Porting LVGL touch driver

The LVGL touch driver file mainly provides input events for LVGL. lvgl provides templates for touch coordinates, mouse input, keyboard input, encoder, and button input. Rename

lvgl\examples\porting\lv_port_indev_template.c(.h) to lv_prot_indev.c(.h) and modify the code to implement the following functions:

1. Enable touch driver file;
2. Modify relevant interfaces to provide hardware input events.

Note: Since the GD32VW553H-EVAL onboard screen does not have touch function, users need to modify these interfaces according to their needs to provide hardware event association for LVGL.

Table 4-4. lv_prot_indev.h source code

```
/*Copy this file as "lv_port_indev.h" and set this value to "1" to enable content*/

#if 1
```

Table 4-5. lv_prot_indev.c source code

```
/*Copy this file as "lv_port_indev.c" and set this value to "1" to enable content*/
#if 1
static void touchpad_init(void);
static void touchpad_read(lv_indev_t * indev, lv_indev_data_t * data);
static bool touchpad_is_pressed(void);
static void touchpad_get_xy(int32_t * x, int32_t * y);

static void mouse_init(void);
static void mouse_read(lv_indev_t * indev, lv_indev_data_t * data);
static bool mouse_is_pressed(void);
static void mouse_get_xy(int32_t * x, int32_t * y);

static void keypad_init(void);
static void keypad_read(lv_indev_t * indev, lv_indev_data_t * data);
static uint32_t keypad_get_key(void);

static void encoder_init(void);
static void encoder_read(lv_indev_t * indev, lv_indev_data_t * data);
static void encoder_handler(void);

static void button_init(void);
static void button_read(lv_indev_t * indev, lv_indev_data_t * data);
static int8_t button_get_pressed_id(void);

static bool button_is_pressed(uint8_t id);
```

4.4. Add LVGL time base

The LVGL time base provides the time foundation for LVGL. Use the kernel private timer

systick to configure interrupts every 1ms. In the interrupt, provide a 1ms heartbeat for LVGL. The systick configuration is implemented in systick.c(h), and the interrupt service function adds lv_tick_inc(1); to provide a 1ms time base for LVGL.

Note: The led_spark function in the template is not related to LVGL porting and has been removed.

Table 4-6. Adding LVGL time base in SysTick

```
#include "lvgl/lvgl.h"
/*!
    \brief        core private timer handle function
    \param[in]    none
    \param[out] none
    \retval       none
*/
void eclic_mtip_handler(void)
{
    ECLIC_ClearPendingIRQ(CLIC_INT_TMR);
    delay_decrement();
    lv_tick_inc(1);
}
```

5. Application Example

5.1. Main Function Code Example

The software implementation process in the Main function is as follows:

1. Initialize system clock systick_config to provide LVGL with a time base in the interrupt;
2. Call lv_init to initialize LVGL components;
3. Call lv_port_disp_init to initialize the LCD and display callbacks;
4. Call lv_demo_benchmark to implement the LVGL application demo;
5. Periodically call lv_timer_handler, which is the core implementation function of LVGL;

Table 5-1. main.c source code

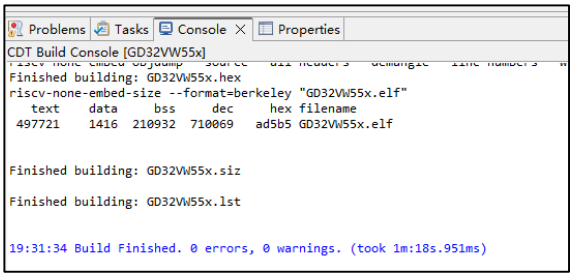
```
#include "gd32vw55x.h"
#include "systick.h"
#include <stdio.h>
#include "gd32vw553h_eval.h"

#include "lvgl/lvgl.h"
//#include "lvgl/examples/porting/lv_port_indev.h"
#include "lvgl/examples/porting/lv_port_disp.h"
#include "lvgl/demos/lv_demos.h"
/*!
    \brief        main function
    \param[in]    none
    \param[out] none
    \retval       none
*/
int main(void)
{
    /* configure systick */
    systick_config();
    /* GUI Tack */
    lv_init();
    lv_port_disp_init();
//    lv_port_indev_init();

//    lv_demo_widgets();
    lv_demo_benchmark();
    while(1){
        lv_timer_handler();
        delay_1ms(5);
    }
```

```
}
```

5.2. Project Compilation and Download

1. First, select the project, then click Project->Build Project to compile the entire project. The Console window will print the compilation information. After compiling, refresh the project directory.

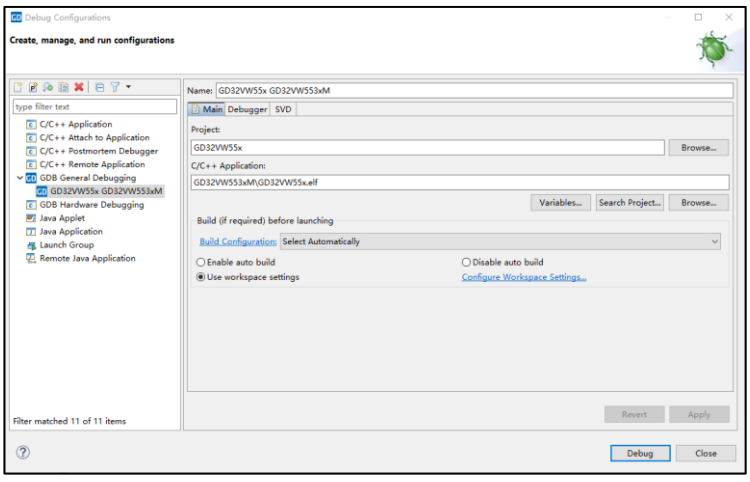
Figure 5-1. Project compilation



2. Directly connect the PC to the USB port of the GD32VW553H_EVAL board, and ensure that the GDLink driver is correctly installed on the PC.

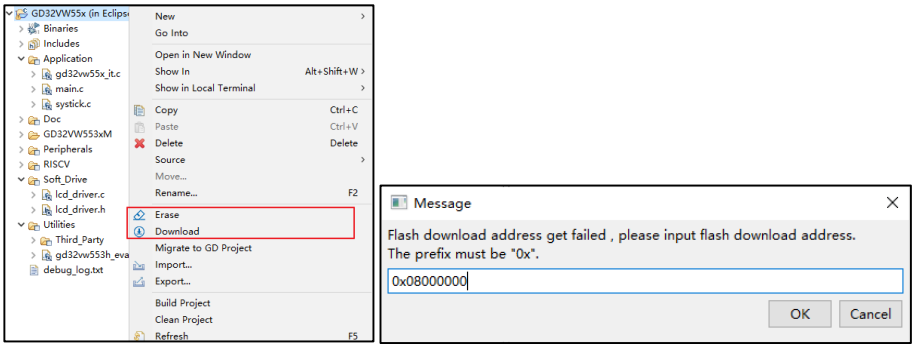
3. First, select the project, then click Run->Debug Configurations to configure the download method. Double-click GDB General Debugging under Debug Configurations to create a new project configuration. The debug file GD32VW553xM\GD32VW55x.elf will be automatically selected. Click Debug to switch windows and automatically erase and download.

Figure 5-2. Project debugging



4. If you need to erase and download, right-click the project and select Erase or Download to erase or download the firmware. A pop-up will ask for the target address 0x08000000 during download. The program will run after reset.

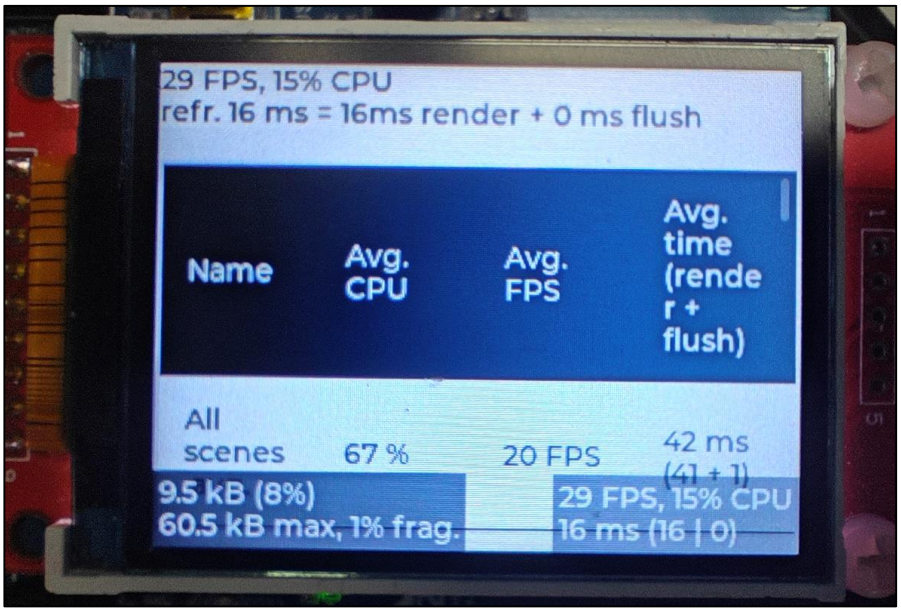
Figure 5-3. Project erase and download



5.3. LVGL Demo

This manual demonstrates the official lv_demo_benchmark sample. Users can demonstrate different samples as needed.

Figure 5-4. lv_demo_benchmark demo



6. Revision History

Figure 6-1. Revision History

Revision No.	Description	Date
1.0	First Release	Feb.05, 2026

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company according to the laws of the People’s Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.