GigaDevice Semiconductor Inc.

基于 GD32VW55x 的 LVGL 移植指南

应用笔记

AN318

1.0 版本

（2026 年 2 月）

目录

图索引

表索引

1. 前言

LVGL 是一种开源免费的图形库，它提供创建嵌入式 GUI 所需的一切，它具有易于使用的图形元素，精美的视觉效果和低内存占用。具体介绍可参考 https://github.com/lvgl/lvgl。

适用产品：GD32VW55x 系列

注意：本应用笔记仅作参考，若与用户手册或数据手册内容有冲突，以用户手册或数据手册为准。

2. 开发环境

开发环境介绍主要如下：

- 硬件开发板：GD32VW553H-EVAL-V1.2 开发板
- 芯片型号：GD32VW553HMQ7
- 操作系统：Win10-64 位
- 开发环境：GD32EmbeddedBuilder_v1.5.4
- 固件库：GD32VW55x_Firmware_Library_V1.4.0
- LVGL：LVGL 9.2.2

3. 工程配置

3.1. 获取 LVGL、SDK 源码

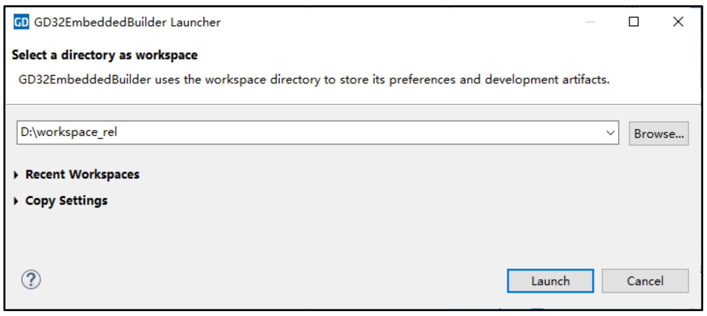
LVGL 图形库的下载地址为：https://github.com/lvgl/lvgl，本手册使用的是 LVGL9.2.2 版本。

下载 GD32VW55x SDK 官方固件开发包 GD32VW55x_Firmware_Library，下载地址为：https://www.gd32mcu.com/cn/download/0?kw=GD32VW5。本手册采用固件版本为 V1.4.0。

3.2. GD32EmbeddedBuilder 移植步骤

1、打开 GD32EmbeddedBuilder 并选择工作区，工作区路径不宜太长。

图 3-1. 选择 IDE 工作区



GD32EmbeddedBuilder Launcher

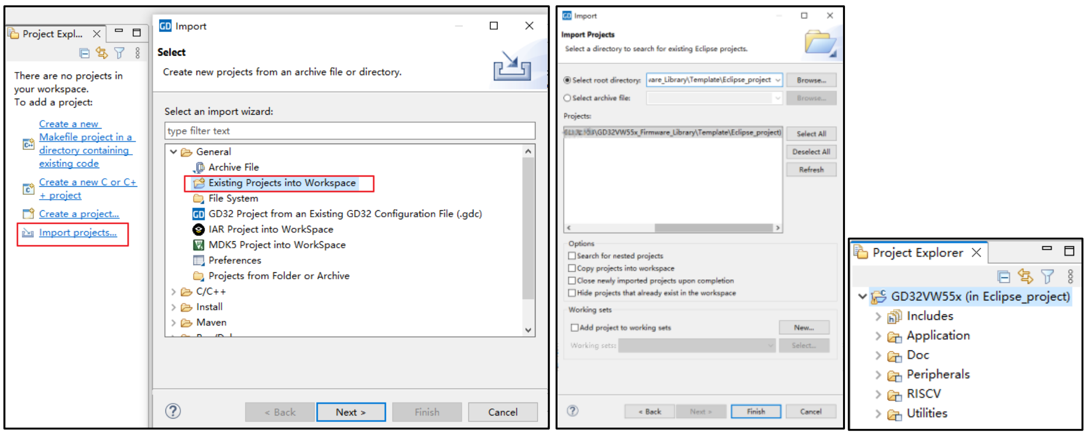
Select a directory as workspace

GD32EmbeddedBuilder uses the workspace directory to store its preferences and development artifacts.

D:\workspace_rel

2、将下载的 GD32VW55x_Firmware_Library 固件包解压到本地，工程目录路径不宜过长，导入 GD32VW55x_Firmware_Library 中的模板工程，依次点击 “Import project"->"Existing Projects into Workspace"，选择 GD32VW55x_Firmware_Library/Template/Eclipse_project 路径，IDE 会自动识别 Eclipse_project 工程，将 Eclipse_project 工程导入到当前工作区。

图 3-2. 导入模板工程



Project Expl...

There are no projects in your workspace.
To add a project:

Create a new Makefile project in a directory containing existing code
Create a new C or C++ project
Create a project...
Import projects...

Import

Select

Create new projects from an archive file or directory.

Select an import wizard:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - GD32 Project from an Existing GD32 Configuration File (.gdc)
  - IAR Project into WorkSpace
  - MDK5 Project into WorkSpace
  - Preferences
  - Projects from Folder or Archive
- C/C++
- Install
- Maven

Import

Import Projects

Select a directory to search for existing Eclipse projects.

(●) Select root directory: ware_Library\Template\Eclipse_project
( ) Select archive file:

Projects:

[redacted]\GD32VW55x_Firmware_Library\Template\Eclipse_project)

Options
- [ ] Search for nested projects
- [ ] Copy projects into workspace
- [ ] Close newly imported projects upon completion
- [ ] Hide projects that already exist in the workspace

Working sets
- [ ] Add project to working sets

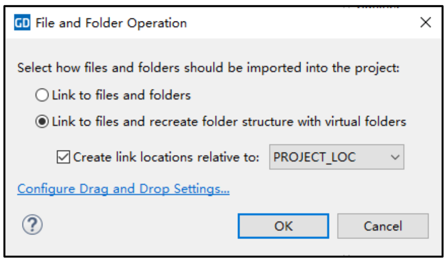
Working sets:

Project Explorer

- GD32VW55x (in Eclipse_project)
  - Includes
  - Application
  - Doc
  - Peripherals
  - RISCV
  - Utilities

3、将下载到的 LVGL 解压到工程目录 GD32VW55x_Firmware_Library\Utilities\Third_Party 下，复制 lv_conf_template.h 并重命名到 lv_conf.h，lv_conf.h 为 LVGL 配置文件。添加 LVGL 必要文件到 IDE 中，拖拽 Third_Party 文件夹到 IDE 工程的 Utilities 目录，选择虚拟文件夹方式：

图 3-3. 导入模板工程


File and Folder Operation

Select how files and folders should be imported into the project:

( ) Link to files and folders

(●) Link to files and recreate folder structure with virtual folders

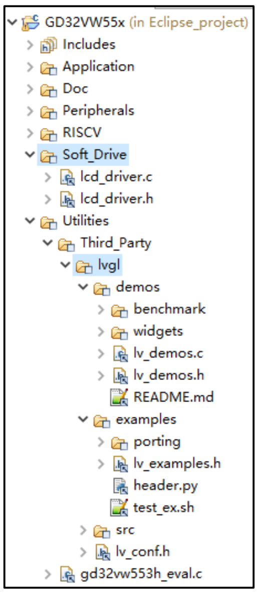
[✓] Create link locations relative to: PROJECT_LOC

Configure Drag and Drop Settings...

删除 lvgl/test 文件夹、lvgl/demos 中除 benchmark 和 widgets 外的其他文件夹、lvgl/examples 中除 porting 外的其他文件夹、lvgl 目录下无关文件，IDE 无需编译它们，该删除不会删除磁盘文件。

相同方式添加 LCD 驱动文件，拖拽 Soft_Drive 文件夹到 IDE 工程上。最终 IDE 中文件目录如下图所示。

图 3-4. 导入 LVGL 和 LCD 驱动文件

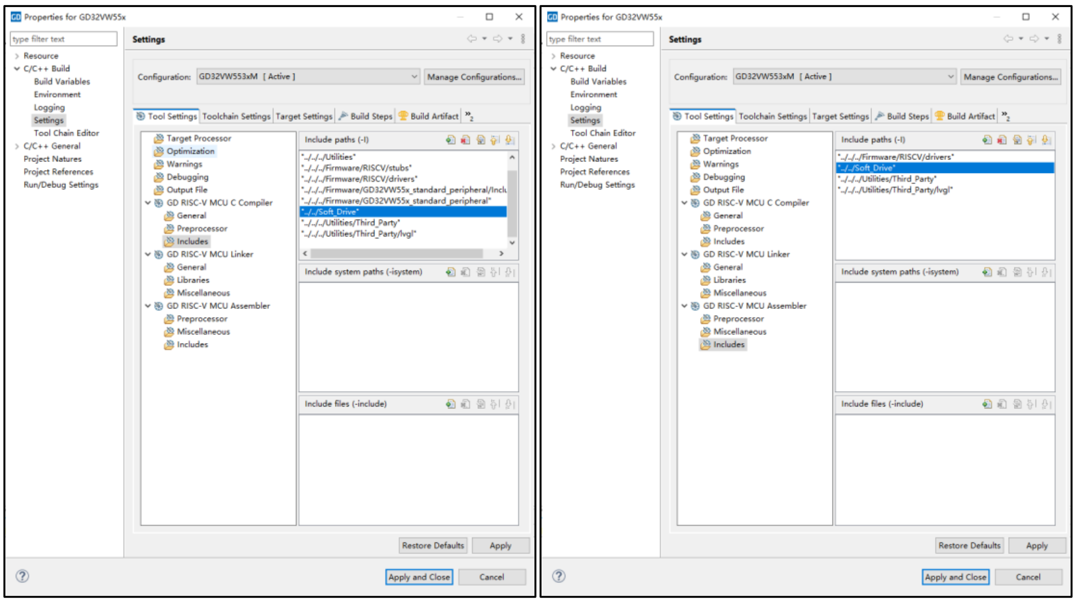

```
GD32VW55x (in Eclipse_project)
  Includes
  Application
  Doc
  Peripherals
  RISCV
  Soft_Drive
    lcd_driver.c
    lcd_driver.h
  Utilities
    Third_Party
      lvgl
        demos
          benchmark
          widgets
          lv_demos.c
          lv_demos.h
          README.md
        examples
          porting
          lv_examples.h
          header.py
          test_ex.sh
        src
        lv_conf.h
    gd32vw553h_eval.c
```

4、在工程属性的 Setting/Tool Seettings/GD RISC-V MCU C Compiler/Include 和 Setting/Tool Seettings/GD RISC-V MCU C Assembler/Include 添加头文件路径

表 3-1.头文件路径

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

图 3-5. 添加头文件路径



Configuration: GD32VW553xM [Active]

Include paths (-I)

```
"../../../Utilities"
"../../../Firmware/RISCV/stubs"
"../../../Firmware/RISCV/drivers"
"../../../Firmware/GD32VW55x_standard_peripheral/Inclu
"../../../Firmware/GD32VW55x_standard_peripheral"
"../../Soft_Drive"
"../../../Utilities/Third_Party"
"../../../Utilities/Third_Party/lvgl"
```

Include system paths (-isystem)

Include files (-include)

Configuration: GD32VW553xM [Active]

Include paths (-I)

```
"../../../Firmware/RISCV/drivers"
"../../Soft_Drive"
"../../../Utilities/Third_Party"
"../../../Utilities/Third_Party/lvgl"
```

Include system paths (-isystem)

Include files (-include)

4. 代码适配

4.1. 修改 LVGL 配置文件

在添加完相应的 LVGL 文件后，需要修改 lv_conf.h 文件，修改内容如下：

1. 使能该配置文件；
2. 颜色格式与 LCD 驱动保持一致，本文使用 RGB565 格式：#define LV_COLOR_DEPTH 16；
3. 可选配置，保证后续 demo 可运行：内存大小、24 号字体、系统监控、内存监控、widgets demo、benchmark demo。

表 4-1. 配置 LVGL

```
/* 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. 移植 LVGL 显示驱动

LVGL 的显示驱动文件主要为 LVGL 提供 LCD 参数定义、绘制缓冲区和刷新回调接口。将 lvgl\examples\porting\lv_port_disp_template.c(.h)重命名为 lv_prot_disp.c(.h)，修改代码实现以下功能：

1. 使能显示驱动文件；
2. 定义屏幕分辨率。

表 4-2. lv_prot_disp.h 源码

```
/* 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. 使能显示驱动文件；

2. lv_port_disp_init 函数：注册绘制缓冲区、绘制回调函数、屏幕分辨率；
3. disp_init 函数：调用 lcd_init 函数初始化 LCD；
4. disp_flush 函数：使用 lcd_copy 函数调用 DMA 或软件将绘制区数据刷新到 LCD 的对应位置，可使用单缓冲或双缓冲模式；
5. lcd_copy 函数：支持 DMA 或软件两种刷屏方式，SPI 采用 8 位方式发送刷新地址命令，后切换 16 位方式发送图像数据以提升传输效率。

表 4-3. lv_prot_disp.c 源码

```
/*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. 移植 LVGL 触摸驱动

LVGL 的触摸驱动文件主要为 LVGL 提供输入事件，lvgl 已提供触摸坐标、鼠标输入、键盘输入、编码器和按键输入模板，将 lvgl\examples\porting\lv_port_indev_template.c(.h)重命名为 lv_prot_indev.c(.h)，修改代码以实现以下功能：

1. 使能触摸驱动文件；
2. 修改相关接口，提供硬件输入事件。

注意：由于 GD32VW553H-EVAL 板载屏幕没有触摸功能，用户需根据需求修改这些接口，为 LVGL 提供硬件事件关联。

表 4-4. lv_prot_indev.h 源码

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

#if 1
```

表 4-5. lv_prot_indev.c 源码

```
/*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. 添加 LVGL 时基

LVGL 时基为 LVGL 提供时间基础，使用内核私有定时器 systick 配置 1ms 进入中断，在中断中为 LVGL 提供 1ms 的心跳节拍。systick 相关配置在 systick.c(h)中实现，中断服务函数添加 lv_tick_inc(1);为 LVGL 提供 1ms 时基。

注意：模板中的 led_spark 函数和 LVGL 移植无关，已删除。

表 4-6. 在 SysTick 中添加 LVGL 时基

```
#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. 应用示例

5.1. Main 函数代码示例

在 Main 函数中软件实现流程如下：

1. 初始化系统时钟 systick_config，中断中为 LVGL 提供时基；
2. 调用 lv_init 初始化 LVGL 组件；
3. 调用 lv_port_disp_init 初始化 LCD 和显示回调；
4. 调用 lv_demo_benchmark 实现 LVGL 应用 demo；
5. 周期调用 lv_timer_handler 实现，该函数为 LVGL 的核心实现；

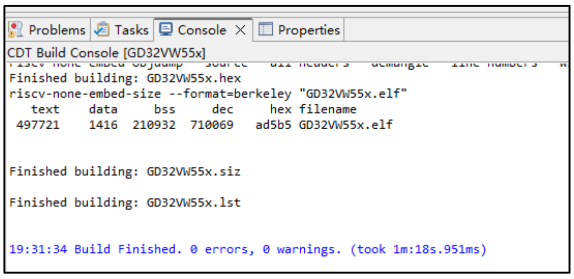
表 5-1. main.c 源码

```
#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. 工程编译和下载

1. 先选中工程，后点击 Project->Build Project 编译整个工程，Console 窗口将会打印编译信息，编译完成后刷新工程目录。

图 5-1. 工程编译



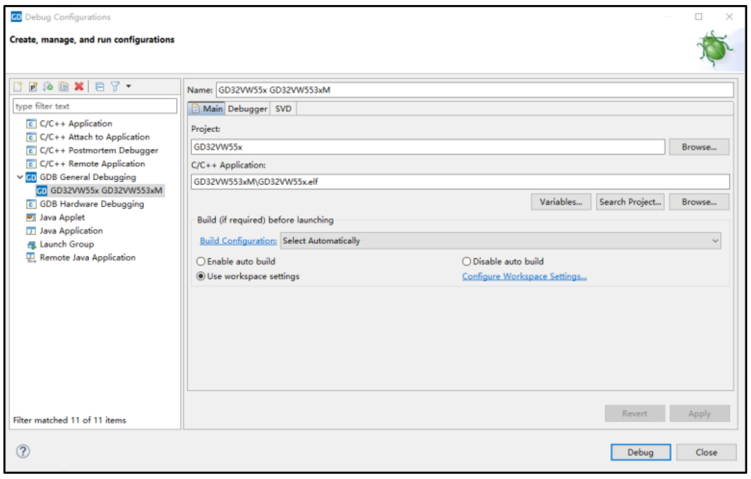
CDT Build Console [GD32VW55x]

```
Finished building: GD32VW55x.hex
riscv-none-embed-size --format=berkeley "GD32VW55x.elf"
   text    data     bss     dec     hex filename
 497721    1416  210932  710069   ad5b5 GD32VW55x.elf


Finished building: GD32VW55x.siz

Finished building: GD32VW55x.lst


19:31:34 Build Finished. 0 errors, 0 warnings. (took 1m:18s.951ms)
```

2. 将 PC 与 GD32VW553H_EVAL 板的 USB 口直接相连，请确保 PC 上已正确安装 GDLink 驱动。

3. 先选中工程，后点击 Run->Debug Configurations 配置下载方式，双击 Debug Configurations 中的 GDB General Debugging 新建工程配置，调试文件 GD32VW553xM\GD32VW55x.elf 将会被自动选择，点击 Debug 后切换窗口并自动擦除和下载。

图 5-2. 工程调试



Name: GD32VW55x GD32VW553xM

Project: GD32VW55x

C/C++ Application: GD32VW553xM\GD32VW55x.elf

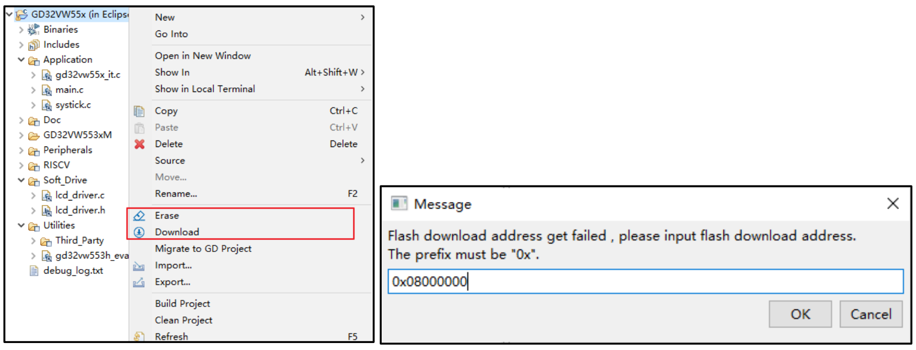
Build (if required) before launching

Build Configuration: Select Automatically

Enable auto build
[✓ Use workspace settings]
Disable auto build

4. 若需要擦除和下载，可右击工程选择 Erase 或 Download 可擦除或下载固件。下载时会弹窗输入目标地址 0x08000000。复位后程序运行。

图 5-3. 工程擦除和下载



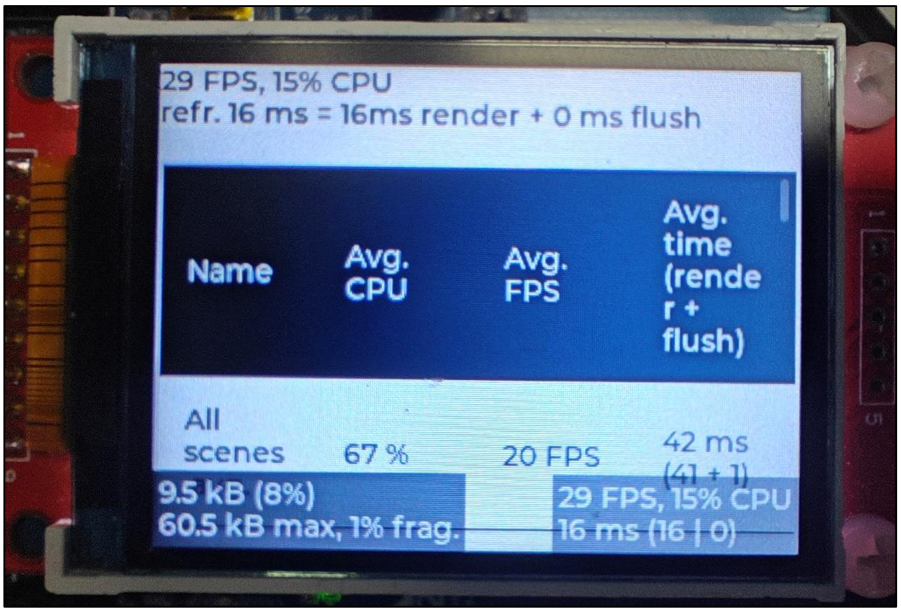
Flash download address get failed , please input flash download address.
The prefix must be "0x".

0x08000000

5.3. LVGL 演示效果

本手册演示官方提供的 lv_demo_benchmark 例程演示，用户可根据需要进行不同例程的演示。

图 5-4. lv_demo_benchmark 演示效果


```
29 FPS, 15% CPU
refr. 16 ms = 16ms render + 0 ms flush

Name      Avg.   Avg.    Avg.
          CPU    FPS     time
                         (render +
                         flush)
All
scenes    67 %   20 FPS  42 ms
                         (41 + 1)
9.5 kB (8%)
60.5 kB max, 1% frag.
                         29 FPS, 15% CPU
                         16 ms (16 | 0)
```

6. 版本历史

表 6-1. 版本历史

版本号.	说明	日期
1.0	首次发布	2026 年 02 月 05 日

Important Notice