GigaDevice Semiconductor Inc.

Arm® Cortex®-M3/4/23/33 32-bit MCU

Application Note
AN042

Table of Contents

List of Figures

List of Tables

1. Introduction

LVGL is an open source and free graphics library that provides everything needed to create an embedded GUI. It has easy-to-use graphic elements, exquisite visual effects and low memory usage. For specific introduction, please refer to https://github.com/littlevgl/lvgl.

2. Development Environment

The development environment is mainly introduced as follows:

- Hardware development board: GD32F450i-EVAL-V1.1 development board
- Cortex-M4: GD32F450IKH6
- Operating system: Win10-64 bit
- Development environment: KEIL 5.27
- Firmware library: GD32F4xx_Firmware_Library V2.1.3
- GUI: LVGL 8.1.0

3. Code porting process

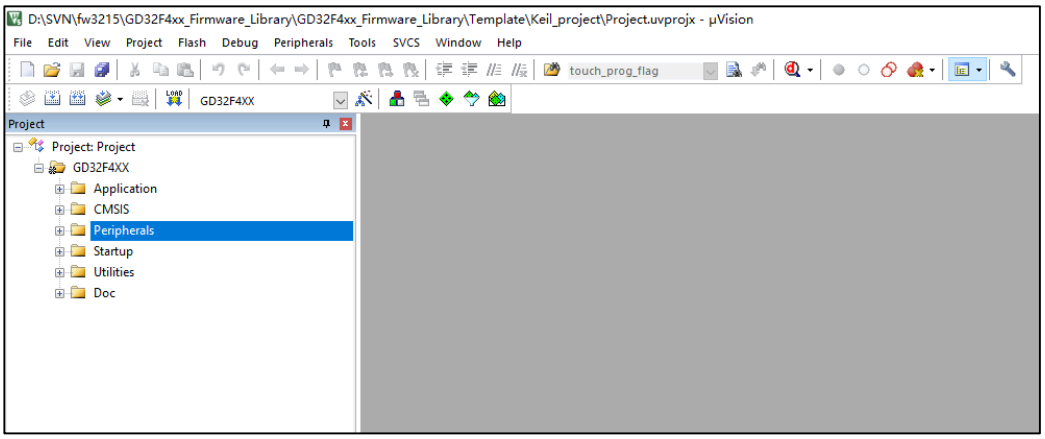
3.1. Material preparation

The LVGL graphics library is needed during the porting process. The download address of the LVGL graphics library is: https://github.com/lvgl/lvgl.This manual uses the LVGL 8.1.0 version. Download GD32F4xx SDK official firmware development package GD32F4xx Firmware Library, the download address is: http://gd32mcu.com/cn/download/7?kw=GD32F4.The firmware version used in this manual is V2.1.3.

3.2. Build the project

After decompressing the downloaded GD32F4xx Firmware Library firmware package, enter the directory "GD32F4xx_Firmware_Library\Template\Keil_project", the default project is opened with KEIL4, and the project name suffix "Project.uvproj" is changed to "Project.uvprojx", then open it with KEIL5 , The interface after opening is shown in ***Figure 3-1. Open the project*** as shown

Figure 3-1. Open the project



The manual porting LVGL will use the SDRAM hardware on the development board, so add gd32f450i_eval_exmc_sdram.c in the Utilities folder, modify some of the code, and change the 38th line "#include "drv_usb_hw.h"" to "#include "systick. h"", "usb_mdelay (10);" in line 188 is modified to "delay_1ms(10);". After the modification is completed, the compilation can be successful.

3.3. Import LVGL into KEIL project

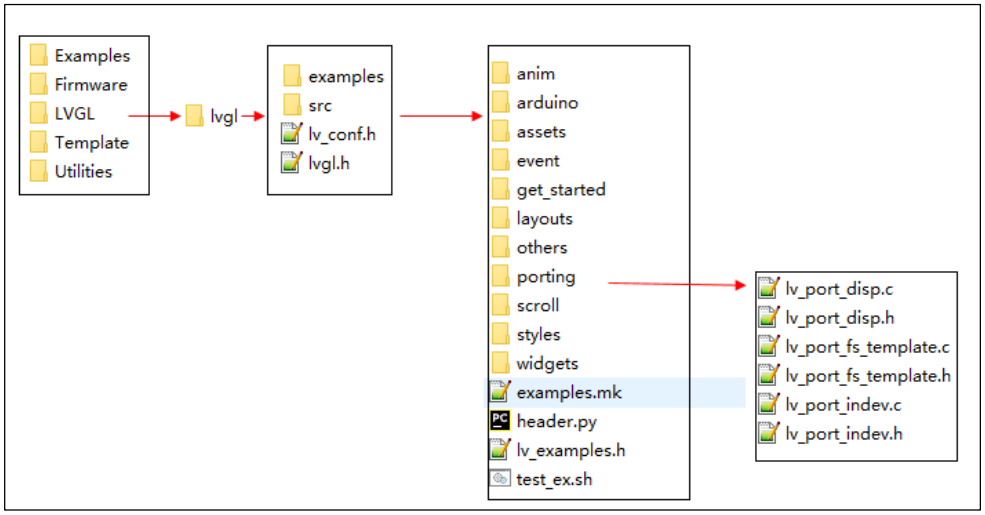
Import LVGL into KEIL, the process is as follows:

1. Create a new LVGL folder in the GD32F4xx_Firmware_Library directory.

2. After decompressing the downloaded lvgl source code, extract the example and src folders, and place the "lv_conf_template.h" and "lvgl.h" files in the LVGL\lvgl folder directory, and modify the name of “lv_conf_template.h” to “lv_conf.h”.

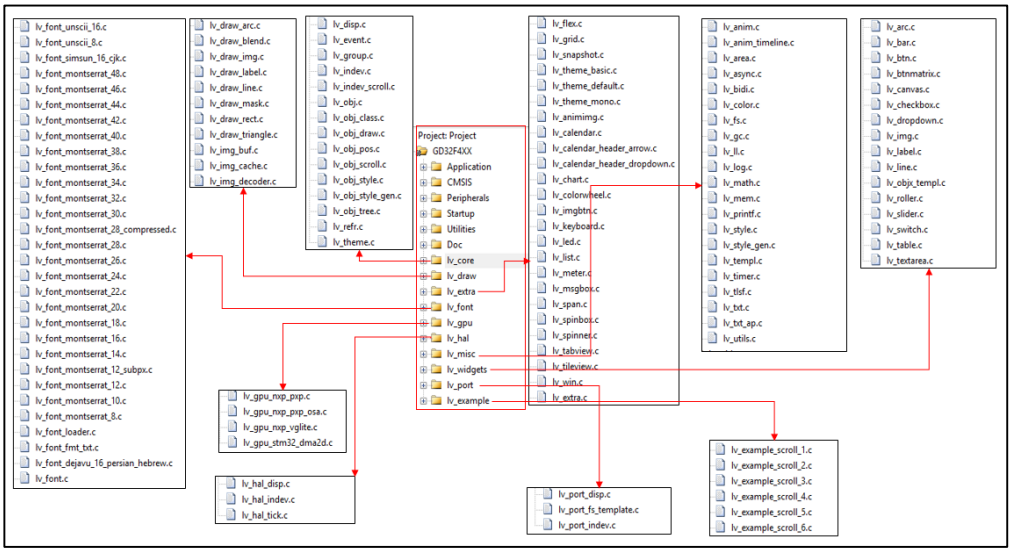
3. In the “LVGL\lvgl\examples\porting” folder, modify the file names of “lv_port_disp_template.c”and “lv_port_disp_template.h” to “lv_port_disp.c” and “lv_port_disp.h”, and modify the file names of “lv_port_indev_template.c” and “lv_port_indev_template.h” to “lv_port_indev.c” and “lv_port_indev.h”, the project directory structure is shown in ***Figure 3-2. Project directory structure***.

Figure 3-2. Project directory structure



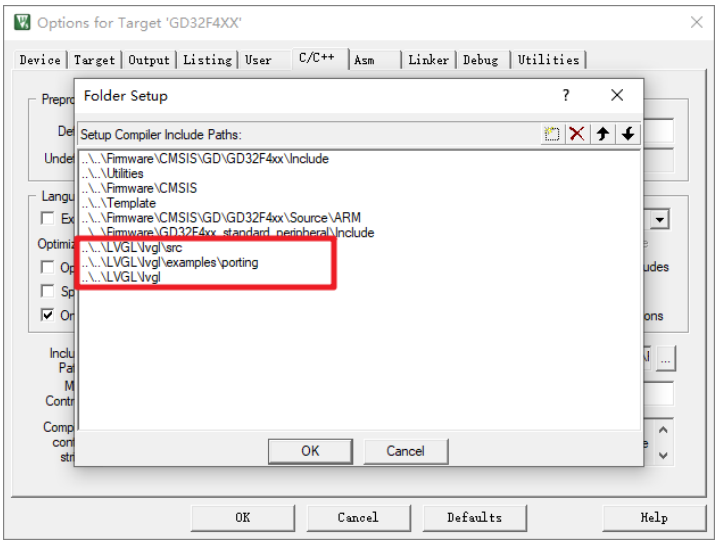
4. Add the project management items of LVGL related files in the KEIL project, and add the .c files in the corresponding LVGL to the corresponding management items. The lv_example folder contains examples of the use of various controls provided by the government. This manual imports the scroll control into this example as a demonstration. The added KEIL project structure is shown in ***Figure 3-3. Project directory structure***.

Figure 3-3. Project directory structure



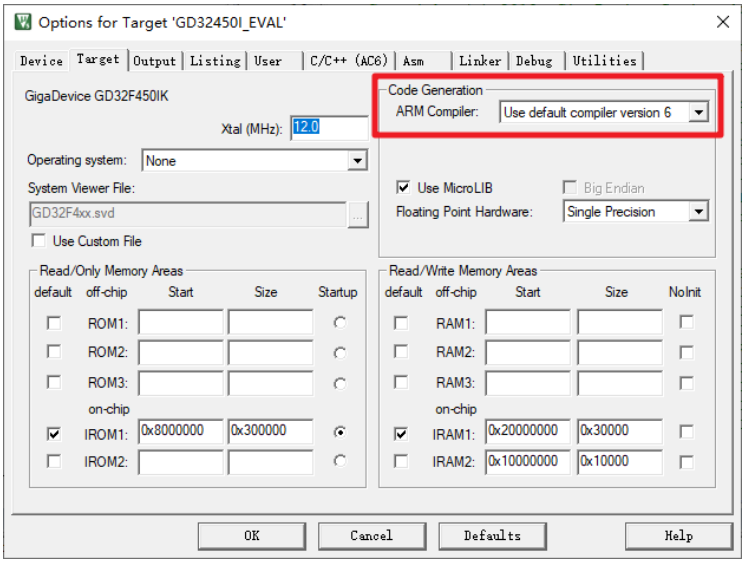
5. Add the corresponding header file in the project, as shown in ***Figure 3-4. KEIL Folder Setup settings***.

Figure 3-4. KEIL Folder Setup settings



6. Modify the compiler version from version 5 to version 6, as shown in ***Figure 3-5. KEIL compiler version setting***.

Figure 3-5. KEIL compiler version setting



3.4. Modify the configuration file

After adding the corresponding LVGL file, the lv_conf.h file needs to be modified. The modification content is as follows:

1. Modify the 10th line "#if 0" to "#if 1" to enable the configuration file.
2. Modify the 24th line and change "LV_COLOR_DEPTH 32" to "LV_COLOR_DEPTH 16", that is, the display mode of the configuration color is RGB565.
3. The GD32F450i-EVAL-V1.1 development board is equipped with a screen with a resolution of 480*272, so add macro definitions: "#define LV_HOR_RES_MAX (480)" and "#define LV_VER_RES_MAX (272)", where LV_HOR_RES_MAX is the display Horizontal resolution, LV_VER_RES_MAX is the vertical resolution of the display.

3.5. Add timer to provide heartbeat for LVGL

Configure TIMER6 to enter the interrupt at a time of 1ms, and provide 1ms heartbeat for LVGL in the interrupt. TIMER6 related configuration and interrupt service functions are shown in ***Table 3-1. TIMER configuration function***.

Table 3-1. TIMER configuration function

```
/* this function handles Timer6 Handler exception */
void TIMER6_IRQHandler(void)
{
    if(SET == timer_interrupt_flag_get(TIMER6, TIMER_INT_FLAG_UP)){
        /* clear channel 0 interrupt bit */
        timer_interrupt_flag_clear(TIMER6, TIMER_INT_FLAG_UP);
        lv_tick_inc(1);
    }
```

```
}

void timer6_config()
{
    /* ----------------------------------------------------------------------
    TIMER6 Configuration:
    TIMER6CLK = SystemCoreClock/20000 = 10kHz, the period is 1s(0.1ms).
    ---------------------------------------------------------------------- */
    timer_parameter_struct timer_initpara;
    rcu_periph_clock_enable(RCU_TIMER6);
    timer_deinit(TIMER6);
    /* initialize TIMER init parameter struct */
    timer_struct_para_init(&timer_initpara);
    /* TIMER6 configuration */
    timer_initpara.prescaler         = 19999;
    timer_initpara.alignedmode       = TIMER_COUNTER_EDGE;
    timer_initpara.counterdirection  = TIMER_COUNTER_UP;
    timer_initpara.period            = 9;
    timer_initpara.clockdivision     = TIMER_CKDIV_DIV1;
    timer_initpara.repetitioncounter = 0;
    timer_init(TIMER6, &timer_initpara);
    timer_enable(TIMER6);
    timer_interrupt_flag_clear(TIMER6, TIMER_INT_FLAG_UP);
    timer_interrupt_enable(TIMER6, TIMER_INT_UP);
}
    nvic_irq_enable(TIMER6_IRQn, 1, 1);
```

3.6. Port display driver

The LVGL display driver files mainly modify the lv_prot_disp.h and lv_prot_disp.c files. Because the development board has a MT48LC16M16A2 SDRAM chip with a size of 32MB and a large storage space, the frame buffer in the TLI is defined to the address in the code 0x0c000000, refer to ***Table 3-8. The source code of main.c***. Define the COLOR_BUF_SIZE buffer in the LVGL display driver interface as the LVGL buffer, and refresh the screen pixel data through DMA. The modified code is as follows: ***Table 3-2. The source code of lv_prot_disp.h*** and ***Table 3-3. The source code of lv_prot_disp.c***.

Table 3-2. The source code of lv_prot_disp.h

```
/* copy this file as "lv_port_disp.h" and set this value to "1" to enable content */
#if 1
#ifndef LV_PORT_DISP_TEMPL_H
#define LV_PORT_DISP_TEMPL_H
#ifdef __cplusplus
```

```
extern "C" {
#endif
#include "lvgl.h"
void lv_port_disp_init(void);
#ifdef __cplusplus
}
#endif
#endif /*LV_PORT_DISP_TEMPL_H*/
#endif /*Disable/Enable content*/
```

Table 3-3. The source code of lv_prot_disp.c

```
#include "lv_port_disp_template.h"
#include "../../lvgl.h"
#include "gd32f4xx.h"

static void disp_flush(lv_disp_drv_t *disp_drv, const lv_area_t *area, lv_color_t *color_p);

static int32_t              x1_flush;
static int32_t              y1_flush;
static int32_t              x2_flush;
static int32_t              y2_fill;
static int32_t              y_fill_act;
static const lv_color_t *buf_to_flush;
static lv_disp_t *our_disp = NULL;
static lv_disp_drv_t disp_drv;
extern uint16_t *my_fb;
#define LCD_FRAME_BUF_ADDR          0XC0000000
#define TLI_LCD_FRAMEBUF_SIZE       (480*272*2)
#define COLOR_BUF_SIZE              (LV_HOR_RES_MAX*LV_VER_RES_MAX)
static lv_color_t color_buf2[COLOR_BUF_SIZE]__attribute__(( section(".ARM.__at_0xC007E900")));

/*!
    \brief      dma transfer data
    \param[in]  src_addr: source address
    \param[in]  dst_addr: destination address
    \param[in]  datalength: data length
    \param[out] none
    \retval     none
*/
void dma_transfer(uint32_t src_addr, uint32_t dst_addr, uint32_t datalength)
{
    dma_periph_address_config(DMA1, DMA_CH0, src_addr);
    dma_memory_address_config(DMA1, DMA_CH0, DMA_MEMORY_0, dst_addr);
    dma_transfer_number_config(DMA1, DMA_CH0, datalength);
```

```
    dma_interrupt_disable(DMA1, DMA_CH0, DMA_CHXCTL_HTFIE);
    dma_interrupt_enable(DMA1, DMA_CH0, DMA_CHXCTL_FTFIE);
    dma_channel_enable(DMA1, DMA_CH0);
}

void lv_port_disp_init(void)
{
    static lv_disp_draw_buf_t buf;
    lv_disp_draw_buf_init(&buf, color_buf2, NULL, LV_HOR_RES_MAX * LV_VER_RES_MAX);
    lv_disp_drv_init(&disp_drv);
    disp_drv.draw_buf = &buf;
    disp_drv.flush_cb = disp_flush;
    disp_drv.hor_res = LV_HOR_RES_MAX;
    disp_drv.ver_res = LV_VER_RES_MAX;
    lv_disp_drv_register(&disp_drv);
}

/*Flush the content of the internal buffer the specific area on the display
 *You can use DMA or any hardware acceleration to do this operation in the background but
 *'lv_disp_flush_ready()' has to be called when finished.*/
static void disp_flush(lv_disp_drv_t *disp_drv, const lv_area_t *area, lv_color_t *color_p)
{
    /*Return if the area is out the screen*/
    if(area->x2 < 0) {
        return;
    }
    if(area->y2 < 0) {
        return;
    }
    if(area->x1 > LV_HOR_RES_MAX - 1) {
        return;
    }
    if(area->y1 > LV_VER_RES_MAX - 1) {
        return;
    }

    /*Truncate the area to the screen*/
    int32_t act_x1 = area->x1 < 0 ? 0 : area->x1;
    int32_t act_y1 = area->y1 < 0 ? 0 : area->y1;
    int32_t act_x2 = area->x2 > LV_HOR_RES_MAX - 1 ? LV_HOR_RES_MAX - 1 : area->x2;
    int32_t act_y2 = area->y2 > LV_VER_RES_MAX - 1 ? LV_VER_RES_MAX - 1 : area->y2;

    x1_flush = act_x1;
```

```
    y1_flush = act_y1;
    x2_flush = act_x2;
    y2_fill = act_y2;
    y_fill_act = act_y1;
    buf_to_flush = color_p;
    dma_transfer((uint32_t)buf_to_flush, (uint32_t)&my_fb[y_fill_act * LV_HOR_RES_MAX +
x1_flush],(x2_flush - x1_flush + 1));
}

/* this function handles DMA1 Handler exception */
void DMA1_Channel0_IRQHandler(void)
{
    if(dma_interrupt_flag_get(DMA1, DMA_CH0, DMA_INT_FLAG_FTF)) {
        dma_interrupt_flag_clear(DMA1, DMA_CH0, DMA_INT_FLAG_FTF);
        y_fill_act ++;

        if(y_fill_act > y2_fill) {
            lv_disp_flush_ready(&disp_drv);
        } else {
            buf_to_flush += x2_flush - x1_flush + 1;
            dma_transfer((uint32_t)buf_to_flush, (uint32_t)&my_fb[y_fill_act * LV_HOR_RES_MAX
+ x1_flush], (x2_flush - x1_flush + 1));
        }
    }
}
```

3.7. Transplant touch driver

LVGL's touch driver files mainly modify the lv_prot_indev.h and lv_prot_indev.c files. The modified content is shown in ***Table 3-4. The source code of lv_prot_indev.h*** and ***Table 3-5. The source code of lv_prot_indev.c***. The touch.c and touch.h files contained in it refer to ***Table 3-6. The source code of touch.h*** and ***Table 3-7. The source code of touch.c***.

Table 3-4. The source code of lv_prot_indev.h

```
/* copy this file as "lv_port_indev.h" and set this value to "1" to enable content */
#if 1
#ifndef LV_PORT_INDEV_TEMPL_H
#define LV_PORT_INDEV_TEMPL_H
#ifdef __cplusplus
extern "C" {
#endif
#include "lvgl.h"
void lv_port_indev_init(void);
```

```
#ifdef __cplusplus
} /*extern "C"*/
#endif
#endif /*LV_PORT_INDEV_TEMPL_H*/
#endif /*Disable/Enable content*/
```

Table 3-5. The source code of lv_prot_indev.c

```
#include "lv_port_indev_template.h"
#include "../../lvgl.h"
#include "touch_panel.h"

extern uint16_t touch_ad_x,touch_ad_y;
static void touchpad_read(lv_indev_drv_t * indev_drv, lv_indev_data_t * data);
static void touchpad_get_xy(lv_coord_t * x, lv_coord_t * y);

lv_indev_t * indev_touchpad;
lv_indev_t * indev_mouse;
lv_indev_t * indev_keypad;
lv_indev_t * indev_encoder;
lv_indev_t * indev_button;

static int32_t encoder_diff;
static lv_indev_state_t encoder_state;

void lv_port_indev_init(void)
{
    static lv_indev_drv_t indev_drv;
    /*Register a touchpad input device*/
    lv_indev_drv_init(&indev_drv);
    indev_drv.type = LV_INDEV_TYPE_POINTER;
    indev_drv.read_cb = touchpad_read;
    indev_touchpad = lv_indev_drv_register(&indev_drv);
}

/* will be called by the library to read the touchpad */
static void touchpad_read(lv_indev_drv_t * indev_drv, lv_indev_data_t * data)
{
    static lv_coord_t last_x = 0;
    static lv_coord_t last_y = 0;

    /* save the pressed coordinates and the state */
    if(touch_scan()) {
        touchpad_get_xy(&last_x, &last_y);
        data->state = LV_INDEV_STATE_PR;
```

```
    } else {
        data->state = LV_INDEV_STATE_REL;
    }

    /*set the last pressed coordinates*/
    data->point.x = last_x;
    data->point.y = last_y;
}

/*Get the x and y coordinates if the touchpad is pressed*/
static void touchpad_get_xy(lv_coord_t * x, lv_coord_t * y)
{
    (*x) = touch_coordinate_x_get(touch_ad_x);
    (*y) = LCD_Y - touch_coordinate_y_get(touch_ad_y);
}
/*Get the x and y coordinates if the touchpad is pressed*/
static void touchpad_get_xy(lv_coord_t * x, lv_coord_t * y)
{
    /*Your code comes here*/

    (*x) = touch_coordinate_x_get(touch_ad_x);
    (*y) = LCD_Y - touch_coordinate_y_get(touch_ad_y);
}
```

Table 3-6. The source code of touch.h

```
#ifndef TOUCH_PANEL_H
#define TOUCH_PANEL_H

#include "gd32f4xx.h"

#define AD_Left       300
#define AD_Right      3850
#define AD_Top        220
#define AD_Bottom     3850

#define LCD_X         480
#define LCD_Y         272

#define CH_X          0XD2
#define CH_Y          0X92

/* SPI SCK pin */
#define   SPI_SCK_PIN                 GPIO_PIN_7
#define   SPI_SCK_PORT                GPIOF
```

```
#define    SPI_SCK_LOW()              gpio_bit_reset(SPI_SCK_PORT, SPI_SCK_PIN)
#define    SPI_SCK_HIGH()             gpio_bit_set(SPI_SCK_PORT, SPI_SCK_PIN)

/* SPI MOSI pin */
#define    SPI_MOSI_PIN               GPIO_PIN_9
#define    SPI_MOSI_PORT               GPIOF
#define    SPI_MOSI_LOW()             gpio_bit_reset(SPI_MOSI_PORT, SPI_MOSI_PIN)
#define    SPI_MOSI_HIGH()            gpio_bit_set(SPI_MOSI_PORT, SPI_MOSI_PIN)

/* SPI MISO pin */
#define    SPI_MISO_PIN               GPIO_PIN_8
#define    SPI_MISO_PORT               GPIOF
#define    SPI_MISO_READ()            gpio_input_bit_get(SPI_MISO_PORT, SPI_MISO_PIN)

/* SPI Chip select pin */
#define    SPI_TOUCH_CS_PIN            GPIO_PIN_6
#define    SPI_TOUCH_CS_PORT            GPIOF
#define    SPI_TOUCH_CS_LOW()          gpio_bit_reset(SPI_TOUCH_CS_PORT,
SPI_TOUCH_CS_PIN)
#define    SPI_TOUCH_CS_HIGH()         gpio_bit_set(SPI_TOUCH_CS_PORT,
SPI_TOUCH_CS_PIN)

/* LCD touch interrupt request pin */
#define    TOUCH_PEN_INT_PIN           GPIO_PIN_3
#define    TOUCH_PEN_INT_PORT           GPIOI
#define    TOUCH_PEN_INT_READ()        gpio_input_bit_get(TOUCH_PEN_INT_PORT,
TOUCH_PEN_INT_PIN)
/* touch panel gpio configure */
void touch_panel_gpio_configure(void);
/* touch start */
void touch_start(void);
/* write data to touch screen */
void touch_write(uint8_t d);
/* read the touch AD value */
uint16_t touch_read(void);
/* read the touch pen interrupt request signal */
FlagStatus touch_pen_irq(void);
/* get the AD sample value of touch location at X coordinate */
uint16_t touch_ad_x_get(void);
/* get the AD sample value of touch location at Y coordinate */
uint16_t touch_ad_y_get(void);
/* get channel X+ AD average sample value */
uint16_t touch_average_ad_x_get(void);
```

```
/* get channel Y+ AD average sample value */
uint16_t touch_average_ad_y_get(void);
/* get X coordinate value of touch point on LCD screen */
uint16_t touch_coordinate_x_get(uint16_t adx);
/* get Y coordinate value of touch point on LCD screen */
uint16_t touch_coordinate_y_get(uint16_t ady);

uint16_t touch_data_filter(uint8_t channel_select);
ErrStatus touch_ad_xy_get(uint16_t *ad_x, uint16_t *ad_y);

ErrStatus touch_scan(void);
```

Table 3-7. The source code of touch.c

```
#include "gd32f4xx.h"
#include "touch_panel.h"
#include "math.h"
#include <stdlib.h>

/* number of filter reads */
#define FILTER_READ_TIMES                5
/* lost value of filter */
#define FILTER_LOST_VAL                  1
/* error range of AD sample value */
#define AD_ERR_RANGE                     6
uint16_t touch_ad_x=0,touch_ad_y=0;
/* set or reset touch screen chip select pin */
static void spi_cs(uint8_t a);
/* set or reset SPI MOSI pin */
static void spi_mosi(uint8_t a);
/* set or reset touch screen clock SPI SCK pin */
static void spi_clk(uint8_t a);
/* get SPI MISO pin input status */
static FlagStatus spi_miso(void);
/* SPI delay function */
static void spi_delay(uint16_t i);
/*!
    \brief        touch panel gpio configure
    \param[in]    none
    \param[out] none
    \retval       none
*/
void touch_panel_gpio_configure(void)
{
    /* GPIO clock enable */
```

```
    rcu_periph_clock_enable(RCU_GPIOI);
    rcu_periph_clock_enable(RCU_GPIOB);
    rcu_periph_clock_enable(RCU_GPIOF);

     gpio_af_set(SPI_SCK_PORT, GPIO_AF_5, SPI_SCK_PIN);
    gpio_mode_set(SPI_SCK_PORT, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE,
SPI_SCK_PIN);
    gpio_output_options_set(SPI_SCK_PORT, GPIO_OTYPE_PP, GPIO_OSPEED_50MHZ,
SPI_SCK_PIN);

     gpio_af_set(SPI_MOSI_PORT, GPIO_AF_5, SPI_MOSI_PIN);
    gpio_mode_set(SPI_MOSI_PORT, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE,
SPI_MOSI_PIN);
    gpio_output_options_set(SPI_MOSI_PORT, GPIO_OTYPE_PP, GPIO_OSPEED_50MHZ,
SPI_MOSI_PIN);

    gpio_af_set(SPI_MISO_PORT, GPIO_AF_5, SPI_MISO_PIN);
    gpio_mode_set(SPI_MISO_PORT, GPIO_MODE_INPUT, GPIO_PUPD_NONE,
SPI_MISO_PIN);
    gpio_output_options_set(SPI_MISO_PORT, GPIO_OTYPE_PP, GPIO_OSPEED_50MHZ,
SPI_MISO_PIN);
    gpio_mode_set(SPI_TOUCH_CS_PORT, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE,
SPI_TOUCH_CS_PIN);
    gpio_output_options_set(SPI_TOUCH_CS_PORT, GPIO_OTYPE_PP,
GPIO_OSPEED_50MHZ, SPI_TOUCH_CS_PIN);

    /* touch pen IRQ pin PI3 configure */
    gpio_mode_set(TOUCH_PEN_INT_PORT, GPIO_MODE_INPUT, GPIO_PUPD_NONE,
TOUCH_PEN_INT_PIN);
    gpio_output_options_set(TOUCH_PEN_INT_PORT, GPIO_OTYPE_PP,
GPIO_OSPEED_50MHZ, TOUCH_PEN_INT_PIN);

    /* Set chip select pin high */
    SPI_TOUCH_CS_HIGH();
}
/*!
    \brief        set or reset touch screen chip select pin
    \param[in]    a: specified the low or high level of chip select pin output
    \param[out] none
    \retval       none
*/
static void spi_cs(uint8_t a)
{
```

```
    if(a){
        SPI_TOUCH_CS_HIGH();
    }else{
        SPI_TOUCH_CS_LOW();
    }
}

/*!
    \brief      set or reset SPI MOSI pin
    \param[in]  a: specified the low or high level of SPI MOSI pin output
    \param[out] none
    \retval     none
*/
static void spi_mosi(uint8_t a)
{
    if(a){
        SPI_MOSI_HIGH();
    }else{
        SPI_MOSI_LOW();
    }
}

/*!
    \brief      set or reset touch screen clock SPI SCK pin
    \param[in]  a: specified the low or high level of SPI SCK pin output
    \param[out] none
    \retval     none
*/
static void spi_clk(uint8_t a)
{
    if(a){
        SPI_SCK_HIGH();
    }else{
        SPI_SCK_LOW();
    }
}

/*!
    \brief      get SPI MISO pin input status
    \param[in]  none
    \param[out] none
    \retval     input status of gpio pin: SET or RESET
*/
```

```
static FlagStatus spi_miso(void)
{
    return SPI_MISO_READ();
}

/*!
    \brief      SPI delay function
    \param[in]  none
    \param[out] none
    \retval     none
*/
static void spi_delay(uint16_t i)
{
    uint16_t k;
    for (k=0;k<i;k++);
}

/*!
    \brief      touch start
    \param[in]  none
    \param[out] none
    \retval     none
*/
void touch_start(void)
{
    spi_clk(0);
    spi_cs(1);
    spi_mosi(1);
    spi_clk(1);
    spi_cs(0);
}

/*!
    \brief      write data to touch screen
    \param[in]  d: the data to be written
    \param[out] none
    \retval     none
*/
void touch_write(uint8_t d)
{
    uint8_t buf, i ;
    spi_clk(0);
    for( i = 0; i < 8; i++){
```

```
            buf = ((d >> (7-i)) & 0x1);
            spi_mosi(buf);
            spi_clk(0);
            spi_clk(1);
            spi_clk(0);
        }
}

/*!
    \brief          read the touch AD value
    \param[in]    None
    \param[out] none
    \retval         the value of touch AD
*/
uint16_t touch_read(void)
{
    uint16_t buf ;
    uint8_t i ;

    buf=0;
    for( i = 0; i < 12; i++){
        buf = buf << 1 ;
        spi_clk(1);
        spi_clk(0);
        if(RESET != spi_miso()){
            buf = buf + 1 ;
        }
    }
    return( buf );
}

/*!
    \brief          read the touch pen interrupt request signal
    \param[in]    none
    \param[out] none
    \retval         the status of touch pen: SET or RESET
      \arg              SET: touch pen is inactive
      \arg              RESET: touch pen is active
*/
FlagStatus touch_pen_irq(void)
{
    return TOUCH_PEN_INT_READ();
}
```

```
/*!
    \brief      get the AD sample value of touch location at X coordinate
    \param[in]  none
    \param[out] none
    \retval     channel X+ AD sample value
*/
uint16_t touch_ad_x_get(void)
{
    if (RESET != touch_pen_irq()){
        /* touch pen is inactive */
        return 0;
    }
    touch_start();
    touch_write(0x00);
    touch_write(CH_X);
    return (touch_read());
}

/*!
    \brief      get the AD sample value of touch location at Y coordinate
    \param[in]  none
    \param[out] none
    \retval     channel Y+ AD sample value
*/
uint16_t touch_ad_y_get(void)
{
    if (RESET != touch_pen_irq()){
        /* touch pen is inactive */
        return 0;
    }
    touch_start();
    touch_write(0x00);
    touch_write(CH_Y);
    return (touch_read());
}

/*!
    \brief      get channel X+ AD average sample value
    \param[in]  none
    \param[out] none
    \retval     channel X+ AD average sample value
*/
```

```
uint16_t touch_average_ad_x_get(void)
{
    uint8_t i;
    uint16_t temp=0;
    for (i=0;i<8;i++){
        temp+=touch_ad_x_get();
        spi_delay(1000);
    }
    temp>>=3;

    return temp;
}

/*!
    \brief      get channel Y+ AD average sample value
    \param[in]  none
    \param[out] none
    \retval     channel Y+ AD average sample value
*/
uint16_t touch_average_ad_y_get(void)
{
    uint8_t i;
    uint16_t temp=0;
    for (i=0;i<8;i++){
        temp+=touch_ad_y_get();
        spi_delay(1000);
    }
    temp>>=3;

    return temp;
}

/*!
    \brief      get X coordinate value of touch point on LCD screen
    \param[in]  adx : channel X+ AD average sample value
    \param[out] none
    \retval     X coordinate value of touch point
*/
uint16_t touch_coordinate_x_get(uint16_t adx)
{
    uint16_t sx = 0;
    uint32_t
    r = adx - AD_Left;
```

```
    r *= LCD_X - 1;
    sx =    r / (AD_Right - AD_Left);
    if (sx <= 0 || sx > LCD_X)
        return 0;
    return sx;
}

/*!
    \brief          get Y coordinate value of touch point on LCD screen
    \param[in]    ady : channel Y+ AD average sample value
    \param[out] none
    \retval         Y coordinate value of touch point
*/
uint16_t touch_coordinate_y_get(uint16_t ady)
{
    uint16_t sy = 0;
    uint32_t
    r = ady - AD_Top;
    r *= LCD_Y - 1;
    sy =    r / (AD_Bottom - AD_Top);
    if (sy <= 0 || sy > LCD_Y)
        return 0;
    return sy;
}

/*!
    \brief          get a value (X or Y) for several times. Order these values,
                      remove the lowest and highest and obtain the average value
    \param[in]    channel_select: select channel X or Y
      \arg              CH_X: channel X
      \arg              CH_Y: channel Y
    \param[out] none
    \retval         a value(X or Y) of touch point
*/
uint16_t touch_data_filter(uint8_t channel_select)
{
    uint16_t i=0, j=0;
    uint16_t buf[FILTER_READ_TIMES];
    uint16_t sum=0;
    uint16_t temp=0;
    /* Read data in FILTER_READ_TIMES times */
    for(i=0; i < FILTER_READ_TIMES; i++){
        if (CH_X == channel_select){
```

```
                buf[i] = touch_ad_x_get();
            }else{     /* CH_Y == channel_select */
                buf[i] = touch_ad_y_get();
            }
        }
        /* Sort in ascending sequence */
        for(i = 0; i < FILTER_READ_TIMES - 1; i++){
            for(j = i + 1; j < FILTER_READ_TIMES; j++){
                if(buf[i] > buf[j]){
                    temp = buf[i];
                    buf[i] = buf[j];
                    buf[j] = temp;
                }
            }
        }
        sum = 0;
        for(i = FILTER_LOST_VAL; i < FILTER_READ_TIMES - FILTER_LOST_VAL; i++){
            sum += buf[i];
        }
        temp = sum / (FILTER_READ_TIMES - 2 * FILTER_LOST_VAL);

        return temp;
}

/*!
    \brief          get the AD sample value of touch location.
                    get the sample value for several times,order these values,remove the lowest and
highest and obtain the average value
    \param[in]    channel_select: select channel X or Y
    \param[out] none
      \arg              ad_x: channel X AD sample value
      \arg              ad_y: channel Y AD sample value
    \retval         ErrStatus: SUCCESS or ERROR
*/
ErrStatus touch_ad_xy_get(uint16_t *ad_x, uint16_t *ad_y)
{
    uint16_t ad_x1=0, ad_y1=0, ad_x2=0, ad_y2=0;

    ad_x1 = touch_data_filter(CH_X);
    ad_y1 = touch_data_filter(CH_Y);
    ad_x2 = touch_data_filter(CH_X);
    ad_y2 = touch_data_filter(CH_Y);
```

```
    if((abs(ad_x1 - ad_x2) > AD_ERR_RANGE) || (abs(ad_y1 - ad_y2) > AD_ERR_RANGE)){
        return ERROR;
    }
    *ad_x = (ad_x1 + ad_x2) / 2;
    *ad_y = (ad_y1 + ad_y2) / 2;

    return SUCCESS;
}

/*!
    \brief      detect the touch event
    \param[in]  none
    \param[out] none
    \retval     ErrStatus: SUCCESS or ERROR
*/
ErrStatus touch_scan(void)
{
    uint8_t invalid_count = 0;
    if (RESET == touch_pen_irq()){ /* touch pen is active */
        while((SUCCESS != touch_ad_xy_get(&touch_ad_x, &touch_ad_y))&& (invalid_count <
20)){
            invalid_count++;
        }

        if(invalid_count >= 20){
            touch_ad_x = 0;
            touch_ad_y = 0;
            return ERROR;
        }
    }else{
        touch_ad_x = 0;
        touch_ad_y = 0;
        return ERROR;
    }
    return SUCCESS;

}
```

3.8. Main function code

In the main function, perform EXMC interface to SDRAM initialization, TLI interface to initialize LCD, SPI interface to initialize touch chip, DMA initialization and LVGL initialization. After the initialization is completed, call the official example routine, and the manual calls the

lv_example_scroll_6() function. The lv_task_handler() function is continuously called in while(1), the specific code is shown in ***Table 3-8. The source code of main.c***.

Table 3-8. The source code of main.c

```
#include "gd32f4xx.h"
#include "gd32f450i_eval.h"
#include "systick.h"
#include <stdio.h>
#include "exmc_sdram.h"
#include "../../lvgl.h"
#include "lv_port_disp_template.h"
#include "lv_port_indev_template.h"
#include "../lv_examples.h"
#include "lv_demo.h"
#include "touch_panel.h"

#define HORIZONTAL_SYNCHRONOUS_PULSE    41
#define HORIZONTAL_BACK_PORCH             2
#define ACTIVE_WIDTH                    480
#define HORIZONTAL_FRONT_PORCH            2

#define VERTICAL_SYNCHRONOUS_PULSE       10
#define VERTICAL_BACK_PORCH               2
#define ACTIVE_HEIGHT                   272
#define VERTICAL_FRONT_PORCH              2

__IO uint16_t * my_fb = (__IO uint16_t*) (0xc0000000);

static void tli_config(void);
static void tli_blend_config(void);
static void tli_gpio_config(void);
static void lcd_config(void);
static void lcd_init(void);
static void dma_config(void);
/*!
    \brief        main function
    \param[in]    none
    \param[out] none
    \retval       none
*/
int main(void)
{
    /* configure systick */
    systick_config();
```

```
    nvic_priority_group_set(NVIC_PRIGROUP_PRE1_SUB3);

    /* config the EXMC access mode */
    exmc_synchronous_dynamic_ram_init(EXMC_SDRAM_DEVICE0);
    lcd_config();
    lcd_init();
    /* configure the GPIO of SPI touch panel */
    touch_panel_gpio_configure();
    delay_1ms(50);
    dma_config();
    delay_1ms(1000);

    lv_init();
    lv_port_disp_init();
    lv_port_indev_init();
    lv_example_scroll_6();
    while(1){
        lv_task_handler();
    }
}

static void dma_config(void)
{
    dma_multi_data_parameter_struct    dma_init_parameter;
    /* peripheral clock enable */
    rcu_periph_clock_enable(RCU_DMA1);

    /* DMA peripheral configure */
    dma_deinit(DMA1,DMA_CH0);

    dma_init_parameter.periph_width = DMA_PERIPH_WIDTH_16BIT;
    dma_init_parameter.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
    dma_init_parameter.memory_width = DMA_MEMORY_WIDTH_16BIT;
    dma_init_parameter.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
    dma_init_parameter.memory_burst_width = DMA_MEMORY_BURST_4_BEAT;
    dma_init_parameter.periph_burst_width = DMA_PERIPH_BURST_4_BEAT;
    dma_init_parameter.critical_value = DMA_FIFO_4_WORD;
    dma_init_parameter.circular_mode = DMA_CIRCULAR_MODE_DISABLE;
    dma_init_parameter.direction = DMA_MEMORY_TO_MEMORY;
    dma_init_parameter.priority = DMA_PRIORITY_ULTRA_HIGH;
    dma_multi_data_mode_init(DMA1,DMA_CH0,&dma_init_parameter);

    nvic_irq_enable(DMA1_Channel0_IRQn, 0, 0);
```

```
}

/*!
    \brief        LCD Configuration
    \param[in]    none
    \param[out] none
    \retval       none
*/
static void lcd_config(void)
{
    /* configure the GPIO of TLI */
    tli_gpio_config();
    /* configure TLI peripheral */
    tli_config();
}

static void lcd_init(void)
{
    tli_layer_enable(LAYER0);
    tli_layer_disable(LAYER1);
    tli_reload_config(TLI_RL_FBR);
    /* enable TLI */
    tli_enable();
}
/*!
    \brief        configure TLI peripheral
    \param[in]    none
    \param[out] none
    \retval       none
*/
static void tli_config(void)
{
    tli_parameter_struct              tli_init_struct;
    tli_layer_parameter_struct        tli_layer_init_struct;

    rcu_periph_clock_enable(RCU_TLI);
    tli_gpio_config();

    /* configure the PLLSAI clock to generate lcd clock */
    if(ERROR == rcu_pllsai_config(192, 2, 3)){
        while(1);
    }
```

```
    rcu_tli_clock_div_config(RCU_PLLSAIR_DIV8);
    rcu_osci_on(RCU_PLLSAI_CK);
    if(ERROR == rcu_osci_stab_wait(RCU_PLLSAI_CK)){
        while(1);
    }
    /* configure TLI parameter struct */
    tli_init_struct.signalpolarity_hs = TLI_HSYN_ACTLIVE_LOW;
    tli_init_struct.signalpolarity_vs = TLI_VSYN_ACTLIVE_LOW;
    tli_init_struct.signalpolarity_de = TLI_DE_ACTLIVE_LOW;
    tli_init_struct.signalpolarity_pixelck = TLI_PIXEL_CLOCK_TLI;
    /* LCD display timing configuration */
    tli_init_struct.synpsz_hpsz = HORIZONTAL_SYNCHRONOUS_PULSE - 1;
    tli_init_struct.synpsz_vpsz = VERTICAL_SYNCHRONOUS_PULSE - 1;
    tli_init_struct.backpsz_hbpsz = HORIZONTAL_SYNCHRONOUS_PULSE +
HORIZONTAL_BACK_PORCH - 1;
    tli_init_struct.backpsz_vbpsz = VERTICAL_SYNCHRONOUS_PULSE +
VERTICAL_BACK_PORCH - 1;
    tli_init_struct.activesz_hasz = HORIZONTAL_SYNCHRONOUS_PULSE +
HORIZONTAL_BACK_PORCH + ACTIVE_WIDTH - 1;
    tli_init_struct.activesz_vasz = VERTICAL_SYNCHRONOUS_PULSE +
VERTICAL_BACK_PORCH + ACTIVE_HEIGHT - 1;
    tli_init_struct.totalsz_htsz = HORIZONTAL_SYNCHRONOUS_PULSE +
HORIZONTAL_BACK_PORCH + ACTIVE_WIDTH + HORIZONTAL_FRONT_PORCH - 1;
    tli_init_struct.totalsz_vtsz = VERTICAL_SYNCHRONOUS_PULSE + VERTICAL_BACK_PORCH
+ ACTIVE_HEIGHT + VERTICAL_FRONT_PORCH - 1;
    tli_init_struct.backcolor_red = 0;
    tli_init_struct.backcolor_green = 0;
    tli_init_struct.backcolor_blue = 0;
    tli_init(&tli_init_struct);
    memset((void*)my_fb, 0x0000, LV_HOR_RES_MAX * LV_VER_RES_MAX*16);
    /* TLI layer0 configuration */
    /* TLI window size configuration */
    tli_layer_init_struct.layer_window_leftpos = HORIZONTAL_SYNCHRONOUS_PULSE +
HORIZONTAL_BACK_PORCH;
    tli_layer_init_struct.layer_window_rightpos = (480 + HORIZONTAL_SYNCHRONOUS_PULSE +
HORIZONTAL_BACK_PORCH - 1);
    tli_layer_init_struct.layer_window_toppos = VERTICAL_SYNCHRONOUS_PULSE +
VERTICAL_BACK_PORCH;
    tli_layer_init_struct.layer_window_bottompos = (272 + VERTICAL_SYNCHRONOUS_PULSE +
VERTICAL_BACK_PORCH - 1);
    /* TLI window pixel format configuration */
    tli_layer_init_struct.layer_ppf = LAYER_PPF_RGB565;
    /* TLI window specified alpha configuration */
```

```
    tli_layer_init_struct.layer_sa = 255;
    /* TLI layer default alpha R,G,B value configuration */
    tli_layer_init_struct.layer_default_blue = 0;
    tli_layer_init_struct.layer_default_green = 0;
    tli_layer_init_struct.layer_default_red = 0;
    tli_layer_init_struct.layer_default_alpha = 0;
    /* TLI window blend configuration */
    tli_layer_init_struct.layer_acf1 = LAYER_ACF1_SA;
    tli_layer_init_struct.layer_acf2 = LAYER_ACF2_SA;
    /* TLI layer frame buffer base address configuration */
    tli_layer_init_struct.layer_frame_bufaddr = (uint32_t)my_fb;
    tli_layer_init_struct.layer_frame_line_length = ((480 * 2) + 3);
    tli_layer_init_struct.layer_frame_buf_stride_offset = (480 * 2);
    tli_layer_init_struct.layer_frame_total_line_number = 272;
    tli_layer_init(LAYER0, &tli_layer_init_struct);
}

/*!
    \brief      configure TLI GPIO
    \param[in]  none
    \param[out] none
    \retval     none
*/
static void tli_gpio_config(void)
{
    /* GPIO clock enable */
    rcu_periph_clock_enable(RCU_GPIOB);
    rcu_periph_clock_enable(RCU_GPIOE);
    rcu_periph_clock_enable(RCU_GPIOH);
    rcu_periph_clock_enable(RCU_GPIOI);
    rcu_periph_clock_enable(RCU_GPIOG);
    rcu_periph_clock_enable(RCU_GPIOF);

    /* configure HSYNC(PI10), VSYNC(PI9), PCLK(PG7) */
    /* configure LCD_R7(PG6), LCD_R6(PH12), LCD_R5(PH11), LCD_R4(PH10),
LCD_R3(PH9),LCD_R2(PH8),
                LCD_R1(PH3), LCD_R0(PH2), LCD_G7(PI2), LCD_G6(PI1), LCD_G5(PI0),
LCD_G4(PH15),
                LCD_G3(PH14), LCD_G2(PH13),LCD_G1(PE6), LCD_G0(PE5),LCD_B7(PI7),
LCD_B6(PI6),
                LCD_B5(PI5), LCD_B4(PI4), LCD_B3(PG11),LCD_B2(PG10), LCD_B1(PG12),
LCD_B0(PE4) */
    /* TLI pins AF configure */
```

```
    gpio_af_set(GPIOE,GPIO_AF_14,GPIO_PIN_5);
    gpio_af_set(GPIOE,GPIO_AF_14,GPIO_PIN_6);
    gpio_af_set(GPIOE,GPIO_AF_14,GPIO_PIN_4);

    gpio_af_set(GPIOH,GPIO_AF_14,GPIO_PIN_2);
    gpio_af_set(GPIOH,GPIO_AF_14,GPIO_PIN_3);
    gpio_af_set(GPIOH,GPIO_AF_14,GPIO_PIN_8);
    gpio_af_set(GPIOH,GPIO_AF_14,GPIO_PIN_9);
    gpio_af_set(GPIOH,GPIO_AF_14,GPIO_PIN_10);
    gpio_af_set(GPIOH,GPIO_AF_14,GPIO_PIN_11);
    gpio_af_set(GPIOH,GPIO_AF_14,GPIO_PIN_12);
    gpio_af_set(GPIOH,GPIO_AF_14,GPIO_PIN_13);
    gpio_af_set(GPIOH,GPIO_AF_14,GPIO_PIN_14);
    gpio_af_set(GPIOH,GPIO_AF_14,GPIO_PIN_15);

    gpio_af_set(GPIOI,GPIO_AF_14,GPIO_PIN_0);
    gpio_af_set(GPIOI,GPIO_AF_14,GPIO_PIN_1);
    gpio_af_set(GPIOI,GPIO_AF_14,GPIO_PIN_2);
    gpio_af_set(GPIOI,GPIO_AF_14,GPIO_PIN_4);
    gpio_af_set(GPIOI,GPIO_AF_14,GPIO_PIN_5);
    gpio_af_set(GPIOI,GPIO_AF_14,GPIO_PIN_6);
    gpio_af_set(GPIOI,GPIO_AF_14,GPIO_PIN_7);
    gpio_af_set(GPIOI,GPIO_AF_14,GPIO_PIN_9);
    gpio_af_set(GPIOI,GPIO_AF_14,GPIO_PIN_10);

    gpio_af_set(GPIOG,GPIO_AF_14,GPIO_PIN_6);
    gpio_af_set(GPIOG,GPIO_AF_14,GPIO_PIN_7);
    gpio_af_set(GPIOG,GPIO_AF_14,GPIO_PIN_10);
    gpio_af_set(GPIOG,GPIO_AF_14,GPIO_PIN_11);
    gpio_af_set(GPIOG,GPIO_AF_14,GPIO_PIN_12);

    /* configure TLI GPIO */
    gpio_mode_set(GPIOE,          GPIO_MODE_AF,          GPIO_PUPD_NONE,
GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6);
    gpio_output_options_set(GPIOE,                       GPIO_OTYPE_PP,
GPIO_OSPEED_200MHZ,GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6);

    gpio_mode_set(GPIOH,          GPIO_MODE_AF,          GPIO_PUPD_NONE,
GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10
                  |GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15);
    gpio_output_options_set(GPIOH,                       GPIO_OTYPE_PP,
GPIO_OSPEED_200MHZ,GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_8|GPIO_PIN_9
```

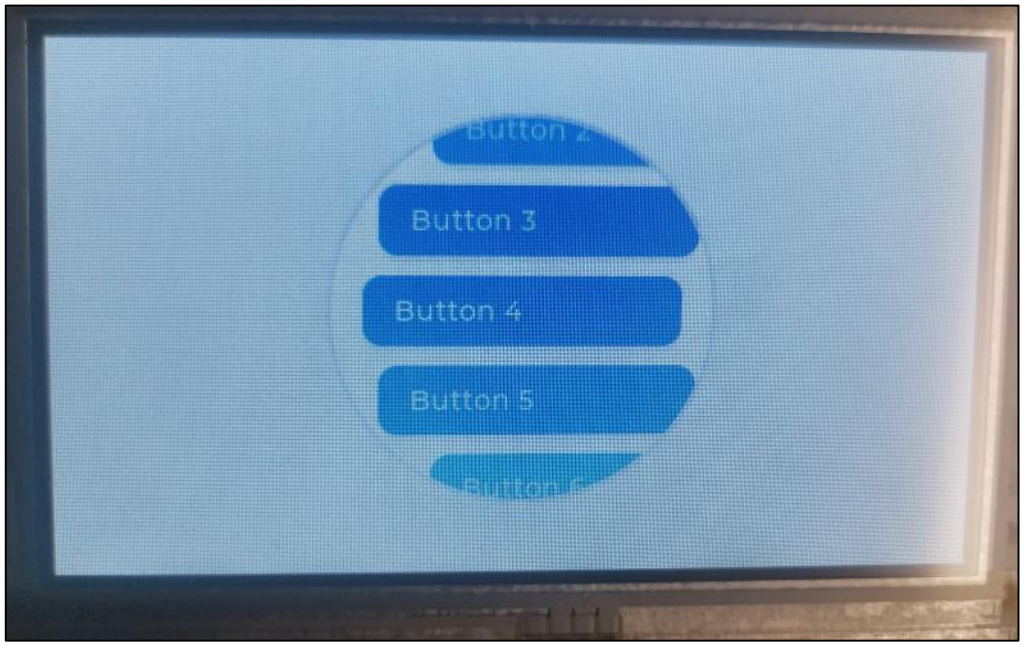
```
|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15);

    gpio_mode_set(GPIOI, GPIO_MODE_AF, GPIO_PUPD_NONE,
GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_4
                  |GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_9|GPIO_PIN_10);
    gpio_output_options_set(GPIOI, GPIO_OTYPE_PP,
GPIO_OSPEED_200MHZ,GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_4
                  |GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_9|GPIO_PIN_10);

    gpio_mode_set(GPIOG, GPIO_MODE_AF, GPIO_PUPD_NONE,
GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12);
    gpio_output_options_set(GPIOG, GPIO_OTYPE_PP,GPIO_OSPEED_200MHZ,
GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12);

    /* LCD PWM BackLight(PB15) */
    gpio_mode_set(GPIOB, GPIO_MODE_OUTPUT, GPIO_PUPD_PULLUP, GPIO_PIN_15);
    gpio_output_options_set(GPIOB, GPIO_OTYPE_PP, GPIO_OSPEED_50MHZ,GPIO_PIN_15);
    gpio_bit_set(GPIOB,GPIO_PIN_15);
}
```

4. LVGL example demo

This manual demonstrates the scroll_1 and scroll_6 routines provided by the government. Users can demonstrate different routines according to their needs.

Figure 4-1. lv_example_scroll_1 demo

Figure 4-2. lv_example_scroll_6 demo



5. Revision history

Table 5-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Nov.30, 2021

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 under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. 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.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s 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 Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.