GigaDevice Semiconductor Inc.

GDemWin GD32H7xx Series Porting Guide

Application Note
AN212

Revision 1.1

(Feb. 2026)

Table of Contents

List of Figures

List of Tables

1. Introduction

This document primarily introduces how to use GDemWin as a GUI middleware for developing UI projects on the GD32H7xx series.

The GD32H737/757/759 series are ultra-high-performance microcontrollers based on the Arm® Cortex®-M7 core. GD32H7 series MCUs offer outstanding processing efficiency, rich connectivity features, and multiple security mechanisms, making them suitable for various applications including data signal processing, motor frequency conversion, power and energy storage systems, audio and video processing, graphics, and imaging.

emWin, developed by SEGGER in Germany, provides advanced support for graphic LCD design, significantly simplifying LCD development. With the emWin graphic library and associated tools like AppWizard, users can easily perform modular designs. This enhances the graphical quality of the interface design, reduces product iteration and update costs, and significantly minimizes development time.

GDemWin is a renamed version of emWin, licensed by GigaDevice from SEGGER. Users can develop projects on GDemWin software libraries free of charge on GD32 MCU Arm® Cortex®-M chips for commercial purposes. The applicable products are as shown in ***Table 1-1. Applicable product***.

Table 1-1. Applicable product

Product series	Model
GD32H77x	GD32H779, GD32H77D, GD32H77E series
GD32H78x	GD32H789, GD32H78D, GD32H78E series
GD32H73x	GD32H737 series
GD32H75x	GD32H757, GD32H759, GD32H75E series

2. Development environment

2.1. Hardware environment

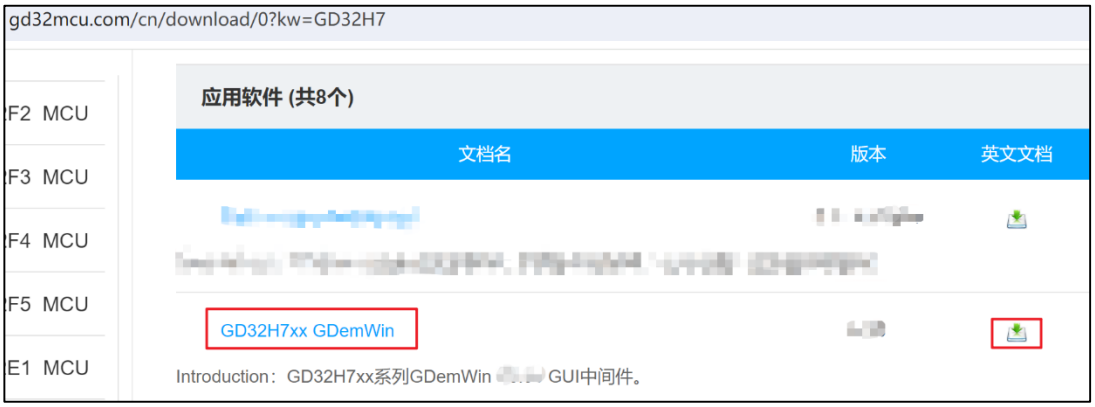
- Development Board: GD32H759I-EVAL Evaluation Board
- Programming Tool: On-board GD-Link

2.2. Software environment

- Operating System: Win10-64 bit
- Development Environment: KEIL 5.29
- Firmware Library: GD32H7xx_Firmware_Library
- GUI: GDemWin

The link to download GDemWin and its associated tools is: www.gd32mcu.com/cn/download/0?kw=GD32H7. Find GD32H7xx GDemWin from the page, as shown in the figure below.

Figure 2-1. Obtaining GDemWin resources from the official website



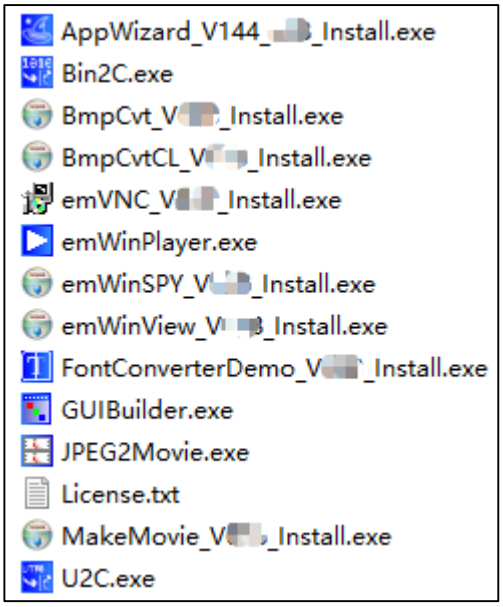
The GD32H7xx_Firmware_Library can be obtained similarly to GDemWin and its associated tools. Both can be downloaded from the website link: www.gd32mcu.com.

3. Porting

3.1. Tool Installation

Select and install the package from the \Tool directory based on your needs, as shown in the figure below.

Figure 3-1. Tool package



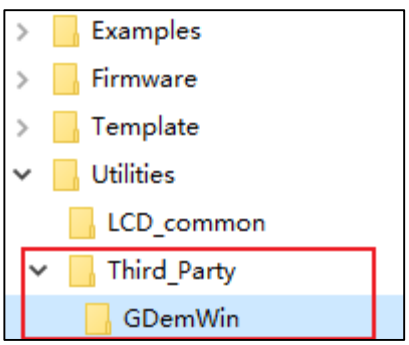
3.2. Driver preparation

First, extract the downloaded GD32H7xx Firmware Library package.

3.2.1. GDemWin library

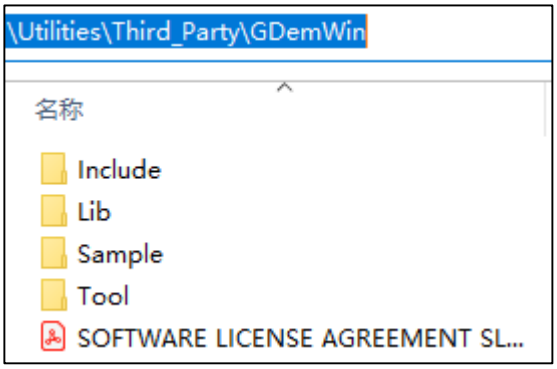
Create a Third_Party folder in the Utilities directory, then go to the \Utilities\Third_Party directory and create a new GDemWin folder.

Figure 3-2. Create a new GDemWin folder



Copy the downloaded GD32H7xx GDemWin content obtained from www.gd32mcu.com/cn/download/0?kw=GD32H7 into the GDemWin folder.

Figure 3-3. Copy the data to the GDemWin folder



3.2.2. TLI LCD driver

The GD32H759I-EVAL development board is equipped with an RGB interface LCD, and the driver can directly use the gd32h759i_lcd_eval.c/.h files located in the Utilities folder.

3.2.3. SDRAM driver

The SDRAM driver can be referenced from \Examples\EXMC\SDRAM\exmc_sdram.c/.h. Copy these files into the Utilities folder.

3.2.4. Touch driver

The screen touch chip on the GD32H759I-EVAL development board communicates with the MCU using an SPI interface. Refer to touch_panel.c/.h.

3.3. Project setup

Open the Keil project located in \Template\Keil_project. The template project is shown in the figure below.

Figure 3-4. Template project

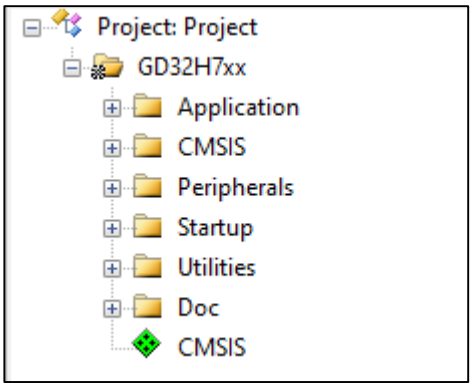


Figure 3-5. Add groups and files

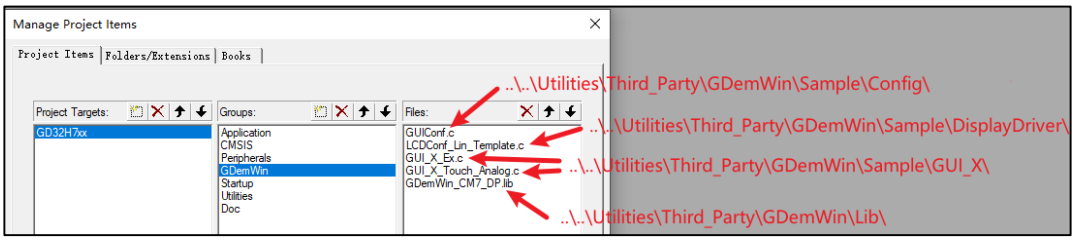
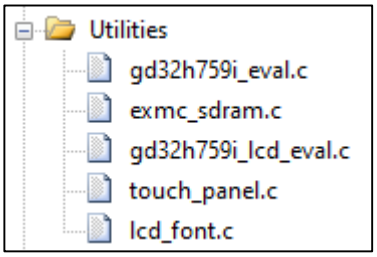


Figure 3-6. Add dirver files



3.3.1. Create a new AppWizard project

AppWizard is a graphical interface development tool for emWin, designed to create efficient, high-quality graphical user interfaces on any embedded system in the simplest way. It enables the creation of complete, ready-to-run applications, simplifying complex graphical applications, reducing UI development difficulty, and improving efficiency.

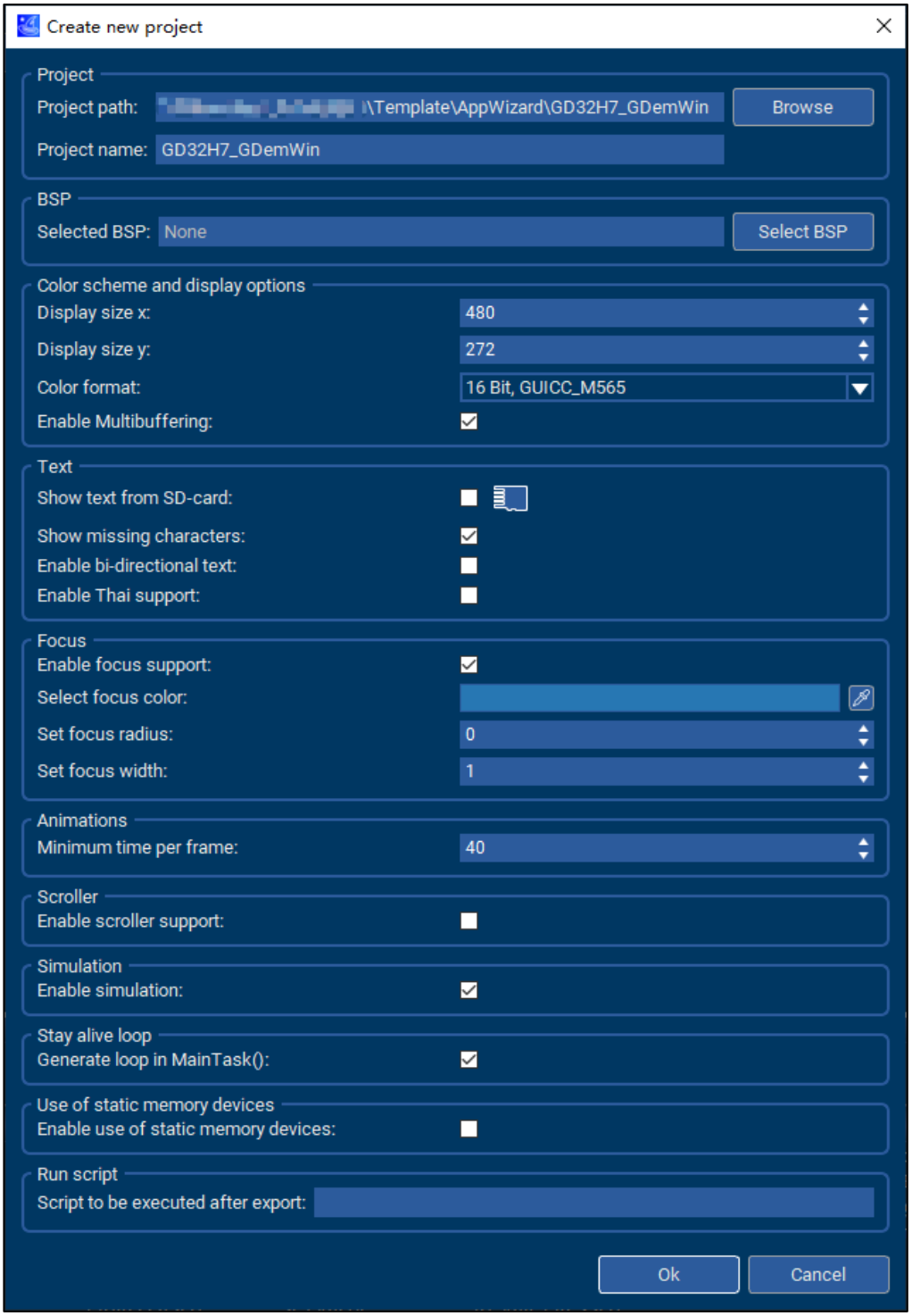
After installing AppWizard, open the software to see the currently supported components, as shown below. For detailed guidance, please refer to the "UM03003 AppWizard User Guide & Reference Manual".

Figure 3-7. Components supported by AppWizard

To facilitate the verification of the display effects of pages generated by AppWizard on the development board, create a new AppWizard folder in the Template directory to store the AppWizard project and the generated output files.

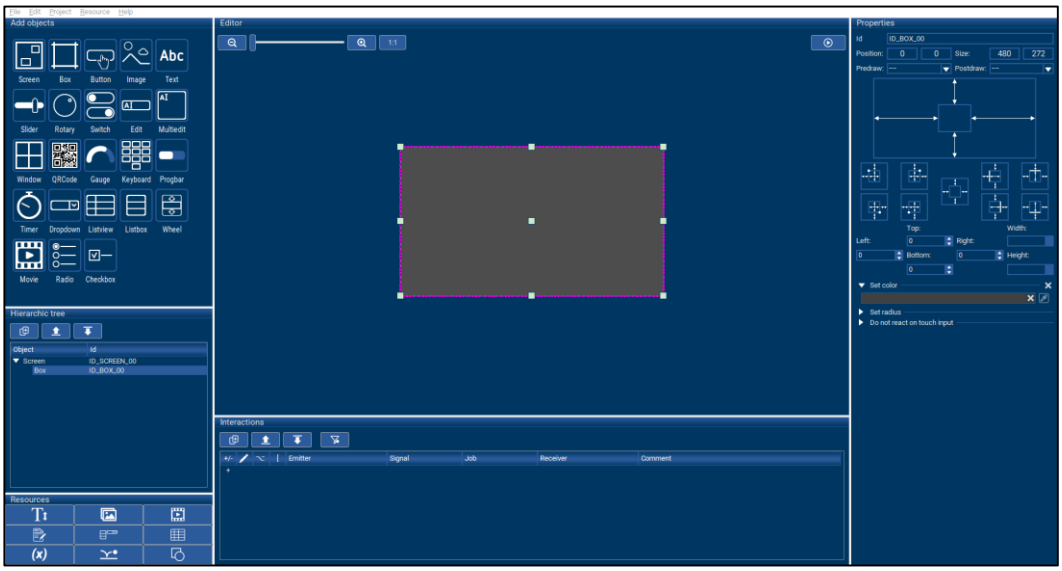
First, create a new AppWizard project, and the project configuration is shown below.

Figure 3-8. Create new AppWizard project



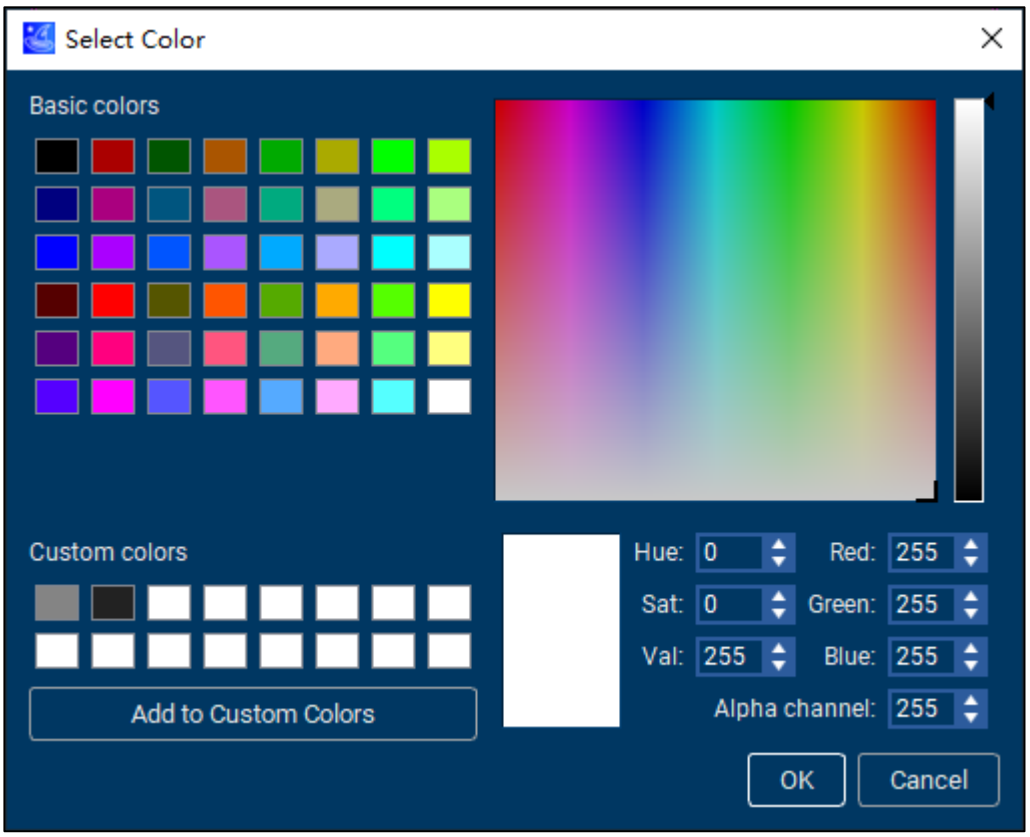
In the AppWizard project, create a new Screen component, followed by a new Box component. The effect is shown below.

Figure 3-9. Create screen



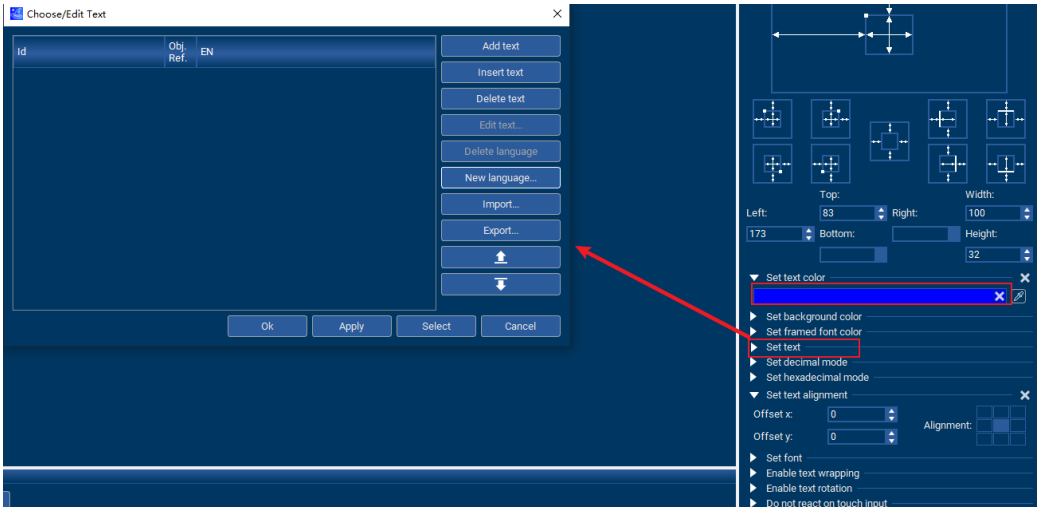
In the Properties panel on the right sidebar, set the color of the Box to white (255, 255, 255), as shown below.

Figure 3-10. Set Box color



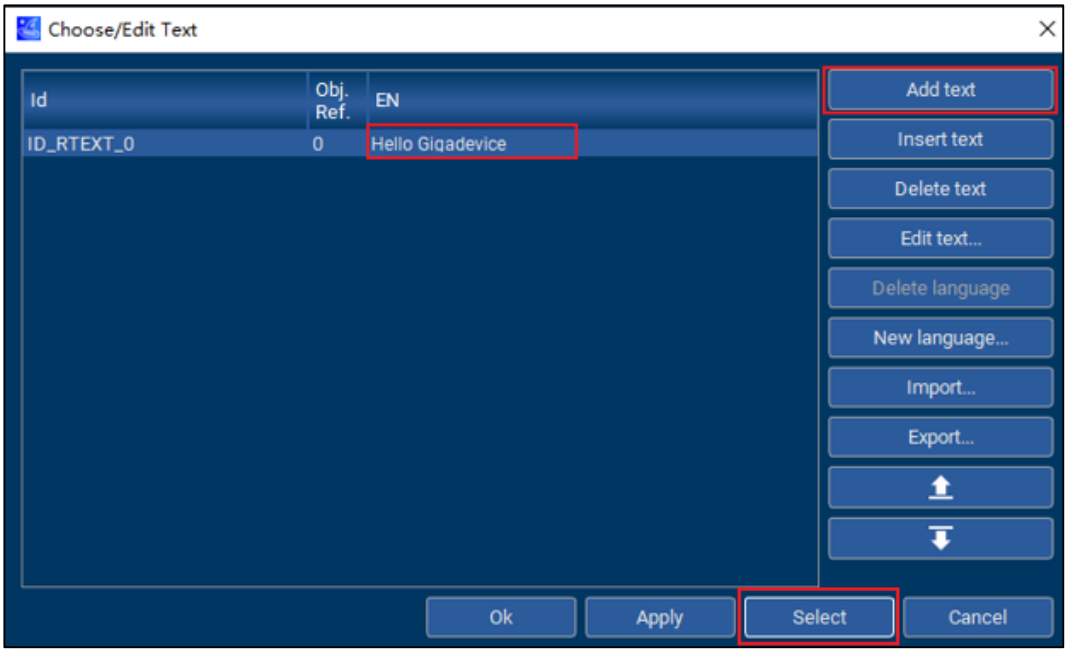
Create a new Text component. In the Properties panel on the right sidebar, set the Text color to blue (0, 0, 255), then click "Set Text" to configure the displayed text. Next, select "New language..." to define the text name, as shown below.

Figure 3-11. Set text display



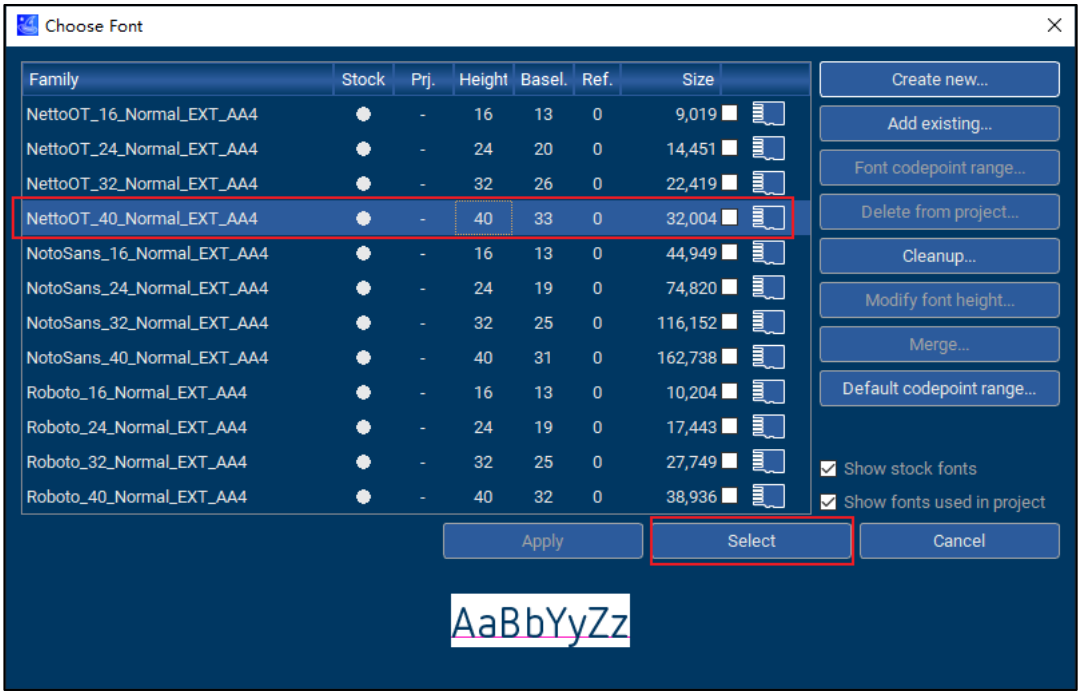
Select "Add text," then enter the text you want to display for ID_RTEXT_0. Click the "Select" button to choose the text, as shown below.

Figure 3-12. Add and select the text



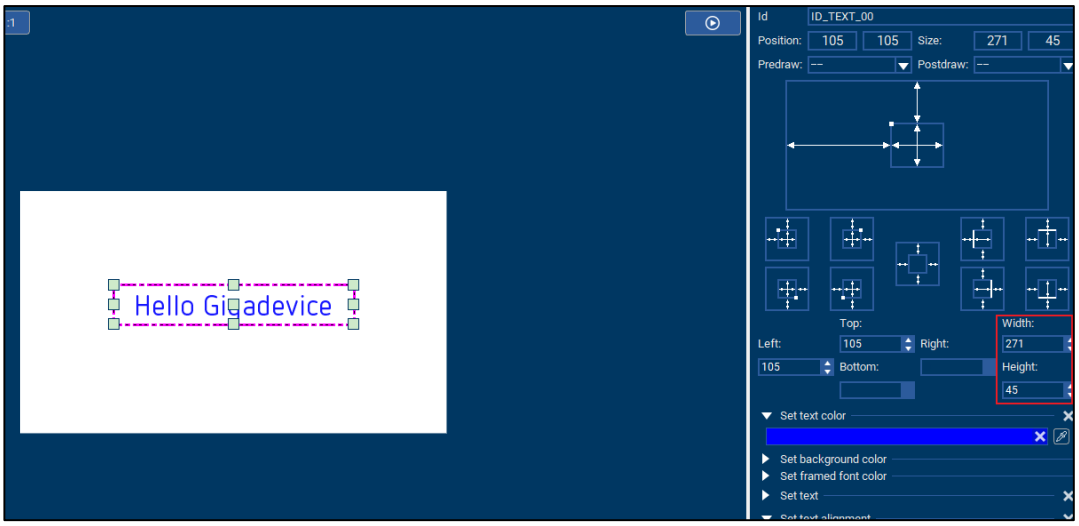
Different fonts can be selected as needed. " Choose font" in the Properties panel on the right sidebar can be clicked, the appropriate font chosen, and the "Select" button clicked to confirm the font, as shown below.

Figure 3-13. Choose font



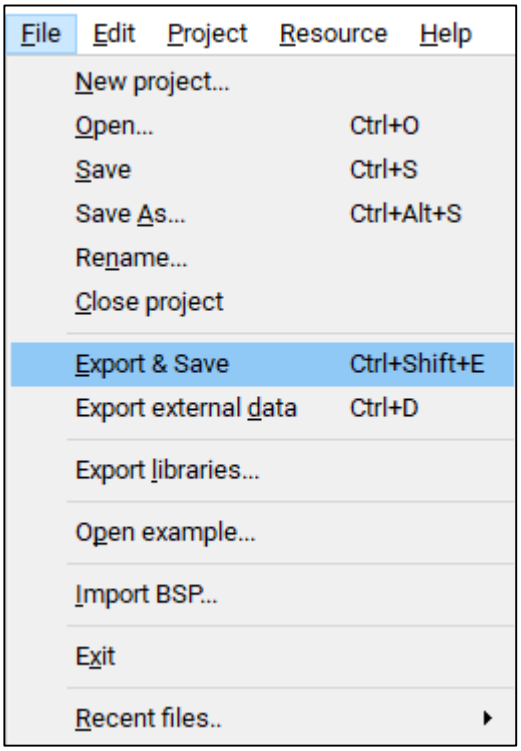
As the initial width and height of the Text are fixed, increasing the font size may result in incomplete display. The width and height attributes of the Text can be adjusted to enable proper display, as shown below.

Figure 3-14. Font attribute adjustment



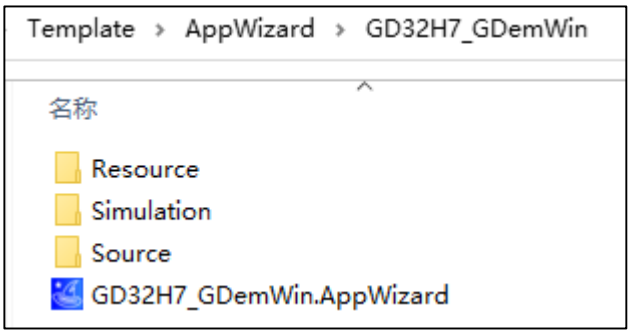
Select "Export & Save" to generate the files for the exported project, as shown in the figure below.

Figure 3-15. Export & Save



After exporting the project, the generated files are shown in the figure below.

Figure 3-16. Exported project files



3.3.2. Import AppWizard project

In the Keil project, create AppWizardSource and AppWizardResource folders, and add the files exported by AppWizard. Also, include the required paths in the Keil project, as shown in the figure below.

Figure 3-17. Add AppWizard Groups

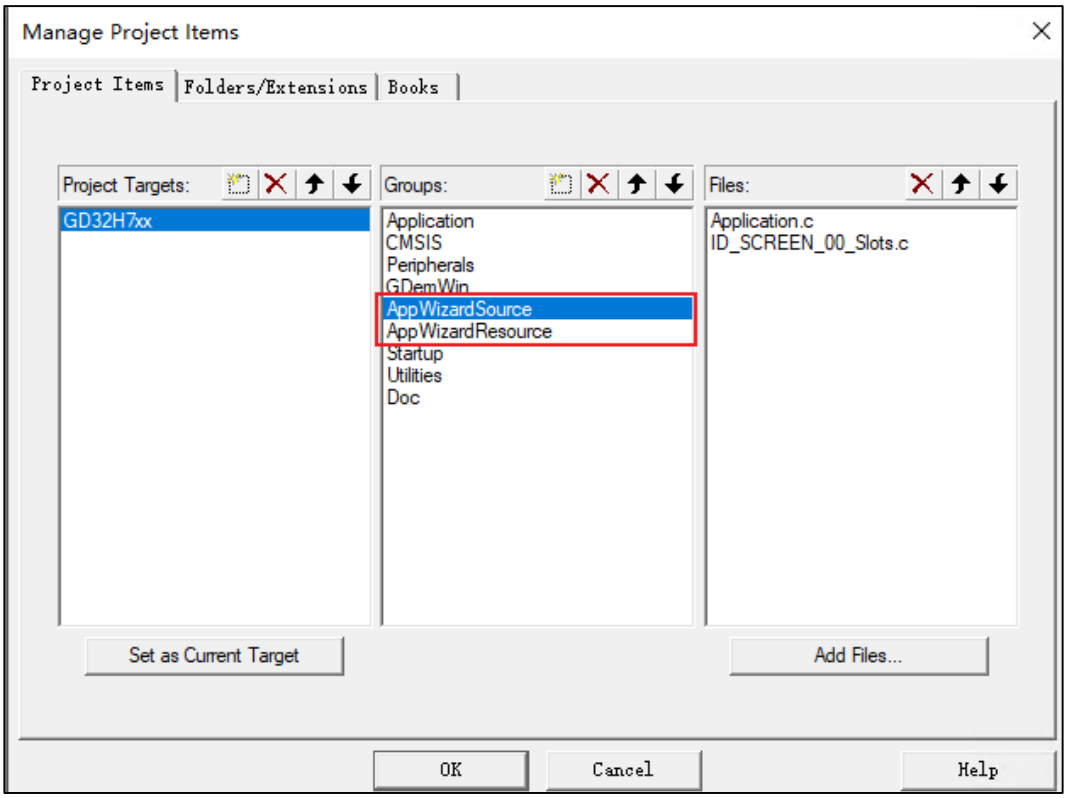
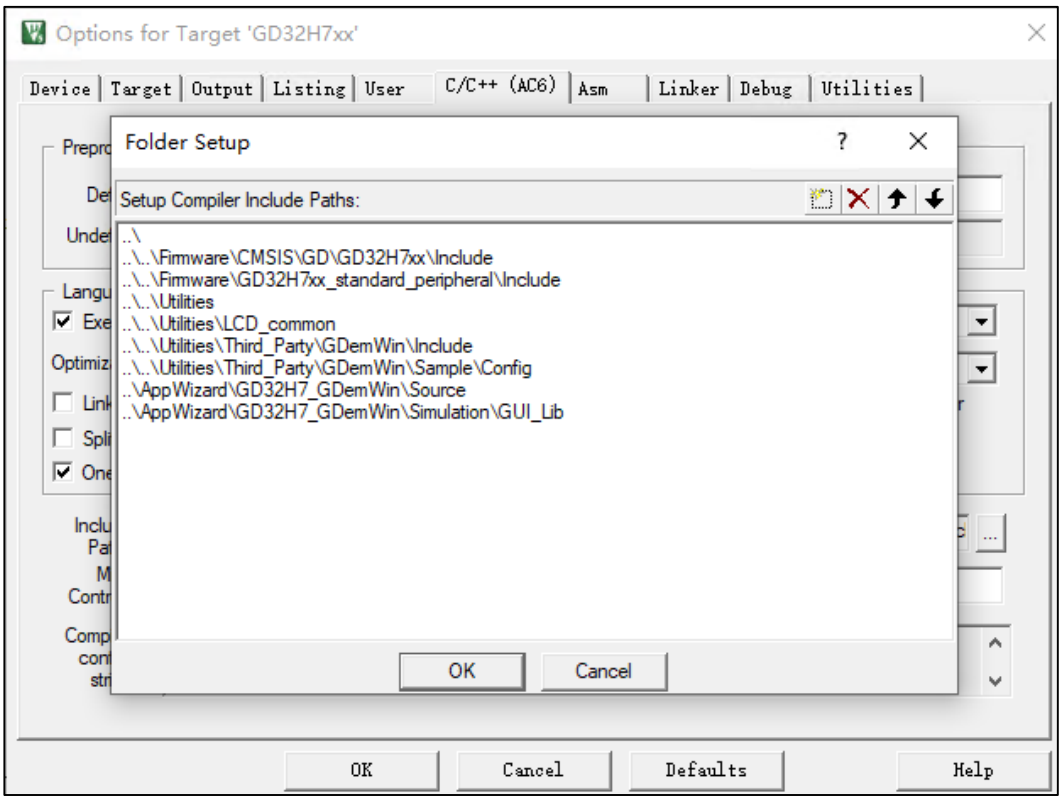


Figure 3-18. Add include paths



3.3.3. Modify Application.c

As this manual does not demonstrate content related to the file system, it is recommended to add the code shown in ***Table 3-1. Modify Application.c***, to the Application.c file to avoid compilation errors.

Table 3-1. Modify Application.c

```
#include "Application.h"

void APPLICATION(void);
void APPLICATION(void) {} // avoid empty object files

#ifndef WIN32
void APPW_X_FS_Init(void)
{

}
#endif
```

3.4. Modify LCDConf_Lin_Template

LCDConf_Lin_Template is the core part of the underlying driver and requires LCD configuration based on the screen parameters. Additionally, the chip's IPA module is used to accelerate and optimize certain functions. Code modifications are provided in ***Table 3-2. Modify LCDConf_Lin_Template***.

Table 3-2. Modify LCDConf_Lin_Template

```
#include "GUI.h"
#include "GUIDRV_Lin.h"
#include "GUI_Private.h"
#include "gd32h759i_lcd_eval.h"
#include "exmc_sdram.h"
#include "gd32h7xx.h"

/*********************************************************************
*
*          Layer configuration (to be modified)
*
**********************************************************************
*/
//
// Physical display size
//
```

```
#define XSIZE_PHYS 480
#define YSIZE_PHYS 272



#define   TOUCH_AD_LEFT              300
#define   TOUCH_AD_RIGHT              3850
#define   TOUCH_AD_TOP               220
#define   TOUCH_AD_BOTTOM              3850

//
// Buffers / VScreens
//
#define NUM_BUFFERS                  3 // Number of multiple buffers to be used
#define NUM_VSCREENS                  1 // Number of virtual screens to be used

#undef   GUI_NUM_LAYERS
#define GUI_NUM_LAYERS               1

#define LCD_LAYER0_FRAME_BUFFER   SDRAM_DEVICE0_ADDR
#define  LCD_LAYER1_FRAME_BUFFER    (LCD_LAYER0_FRAME_BUFFER  +  XSIZE_PHYS  *
YSIZE_PHYS * 4 * NUM_VSCREENS * NUM_BUFFERS)

#define COLOR_CONVERSION_0          GUICC_M565
#define DISPLAY_DRIVER_0            GUIDRV_LIN_16

#if GUI_NUM_LAYERS > 1
#define COLOR_CONVERSION_1          GUICC_M565
#define DISPLAY_DRIVER_1            GUIDRV_LIN_16
#endif

#undef XSIZE_0
#undef YSIZE_0
#define XSIZE_0                     XSIZE_PHYS
#define YSIZE_0                     YSIZE_PHYS

/*********************************************************************
*
*          Configuration checking
*
**********************************************************************
*/
#ifndef     XSIZE_PHYS
#error Physical X size of display is not defined!
```

```
#endif
#ifndef     YSIZE_PHYS
#error Physical Y size of display is not defined!

#endif
#ifndef     NUM_VSCREENS
#define NUM_VSCREENS 1
#else
#if (NUM_VSCREENS <= 0)
#error At least one screeen needs to be defined!
#endif
#endif
#if (NUM_VSCREENS > 1) && (NUM_BUFFERS > 1)
#error Virtual screens and multiple buffers are not allowed!
#endif

/*********************************************************************
*
*          Public code
*
**********************************************************************
*/
static U32              _a_layer[]              = {LAYER0, LAYER1};
static     const     U32          _a_layeraddr[]                    =     {LCD_LAYER0_FRAME_BUFFER,
LCD_LAYER1_FRAME_BUFFER};
static int              _a_pendbuf[2]         = {-1, -1};
static int              _a_bufidx[GUI_NUM_LAYERS];
static int              _a_xsize[GUI_NUM_LAYERS];
static int              _a_ysize[GUI_NUM_LAYERS];
static int              _a_pixelbytes[GUI_NUM_LAYERS];

static U32              _a_ipabuf[XSIZE_PHYS];
static U32              _a_fgbuf[XSIZE_PHYS];
static U32              _a_bgbuf[XSIZE_PHYS];

static volatile int ipa_done;

static uint16_t     _active_width = 480, _active_height = 272, _hsync_w = 40, _vsync_w = 9, _hbp =
2, _hfp = 2, _vbp = 2, _vfp = 2;

static const LCD_API_COLOR_CONV *_ap_color_conv[] = {
      COLOR_CONVERSION_0,
#if GUI_NUM_LAYERS > 1
```

```
    COLOR_CONVERSION_1,
#endif
};

static void ipa_handler(void)
{
    __IO U32 timeout;
    /* the ipa_done will be set in TLI_IRQHandler */
    ipa_done = 0;
    /* start ipa */
    IPA_CTL |= IPA_CTL_TEN;
    /* wait until ipa is done */
    timeout = 0xFFFFFFF;
    while((ipa_done == 0) && timeout) {
        timeout--;
    }
    /* stop ipa if timeout */
    if(timeout == 0) {
        IPA_CTL |= IPA_CTL_TST;
    }
}

static U32 _pixel_format_get(int LayerIndex)
{
    const LCD_API_COLOR_CONV *_p_color_conv;
    tli_layer_ppf_enum layer_ppf;

    if(LayerIndex >= GUI_COUNTOF(_ap_color_conv)) {
        while(1);
    }
    _p_color_conv = _ap_color_conv[LayerIndex];
    if(GUICC_M8888I == _p_color_conv) {
        layer_ppf = LAYER_PPF_ARGB8888;
    } else if((GUICC_M888 == _p_color_conv) | (GUICC_888 == _p_color_conv)) {
        layer_ppf = LAYER_PPF_RGB888;
    } else if((GUICC_M565 == _p_color_conv) | (GUICC_565 == _p_color_conv)) {
        layer_ppf = LAYER_PPF_RGB565;
    } else if(GUICC_M1555I == _p_color_conv) {
        layer_ppf = LAYER_PPF_ARGB1555;
    } else if(GUICC_M4444I == _p_color_conv) {
        layer_ppf = LAYER_PPF_ARGB4444;
    } else if(GUICC_8666 == _p_color_conv) {
        layer_ppf = LAYER_PPF_L8;
```

```
    } else if(GUICC_1616I == _p_color_conv) {
        layer_ppf = LAYER_PPF_AL44;
    } else if(GUICC_88666I == _p_color_conv) {
        layer_ppf = LAYER_PPF_AL88;
    } else {
        while(1);
    }
    return layer_ppf;
}

static U32 _ipa_pixel_format_get(int LayerIndex)
{
    const LCD_API_COLOR_CONV *_p_color_conv;
    ipa_dpf_enum ipa_ppf;

    if(LayerIndex >= GUI_COUNTOF(_ap_color_conv)) {
        while(1);
    }
    _p_color_conv = _ap_color_conv[LayerIndex];
    if(GUICC_M8888I == _p_color_conv) {
        ipa_ppf = IPA_DPF_ARGB8888;
    } else if((GUICC_M888 == _p_color_conv) | (GUICC_888 == _p_color_conv)) {
        ipa_ppf = IPA_DPF_RGB888;
    } else if((GUICC_M565 == _p_color_conv) | (GUICC_565 == _p_color_conv)) {
        ipa_ppf = IPA_DPF_RGB565;
    } else if(GUICC_M1555I == _p_color_conv) {
        ipa_ppf = IPA_DPF_ARGB1555;
    } else if(GUICC_M4444I == _p_color_conv) {
        ipa_ppf = IPA_DPF_ARGB4444;
    } else {
        while(1);
    }
    return ipa_ppf;
}

static int _line_bytes_get(int LayerIndex, int xSize)
{
    int pixel_bits, line_bytes;
#ifdef __CM7_REV
    SCB_CleanInvalidateDCache();
#endif
    pixel_bits  = LCD_GetBitsPerPixelEx(LayerIndex);
    line_bytes = (pixel_bits * xSize + 7) / 8;
```

```
    return line_bytes;
}

static void _alpha_invert_redblue_swap(LCD_COLOR *pSrc, LCD_COLOR *pDst, U32 Num)
{
    U32 color;

    do {
        color = *pSrc++;
        *pDst++ = ((color & 0x000000FF) << 16)                  /* Red <--> Blue */
                | (color & 0x0000FF00)                           /* Green */
                | ((color & 0x00FF0000) >> 16)                  /* Red <--> blue */
                | ((color & 0xFF000000) ^ 0xFF000000); /* Alpha invert */
    } while(--Num);
}

static void _alpha_invert(LCD_COLOR *pSrc, LCD_COLOR *pDst, U32 Num)
{
    U32 color;

    do {
        color = *pSrc++;
        *pDst++ = color ^ 0xFF000000; /* Alpha invert */
    } while(--Num);
}

static void _ipa_alpha_blend(LCD_COLOR *pFG, LCD_COLOR *pBG, LCD_COLOR *pDst, U32
Num)
{
#ifdef __CM7_REV
    SCB_CleanInvalidateDCache();
#endif

    IPA_CTL       = 0x00020000UL | (1 << 9);
    IPA_FMADDR    = (U32)pFG;
    IPA_BMADDR    = (U32)pBG;
    IPA_DMADDR    = (U32)pDst;
    IPA_FLOFF     = 0;
    IPA_BLOFF     = 0;
    IPA_DLOFF     = 0;
    IPA_FPCTL     = LAYER_PPF_ARGB8888;
    IPA_BPCTL     = LAYER_PPF_ARGB8888;
    IPA_DPCTL     = LAYER_PPF_ARGB8888;
```

```
    IPA_IMS        = (U32)(Num << 16) | 1;

    ipa_handler();
}

static void _ipa_mix_colors_bulk(U32 *pFG, U32 *pBG, U32 *pDst, unsigned OffFG, unsigned OffBG,
unsigned OffDest, unsigned xSize, unsigned ySize,
                                        U8 Intens)
{
    int y;

#ifdef __CM7_REV
    SCB_CleanInvalidateDCache();
#endif

    for(y = 0; y < ySize; y++) {
        IPA_CTL          = 0x00020000UL | (1 << 9);
        IPA_FMADDR     = (U32)pFG;
        IPA_BMADDR     = (U32)pBG;
        IPA_DMADDR     = (U32)pDst;

        IPA_FPCTL = LAYER_PPF_ARGB8888
                  | (1UL << 16)
                  | ((U32)Intens << 24);

        IPA_BPCTL = LAYER_PPF_ARGB8888
                  | (0UL << 16)
                  | ((U32)(255 - Intens) << 24);

        IPA_DPCTL    = LAYER_PPF_ARGB8888;

        IPA_IMS    = (U32)(xSize << 16) | 1;

        ipa_handler();


        pFG    += xSize + OffFG;
        pBG    += xSize + OffBG;
        pDst += xSize + OffDest;
    }
}

static void _ipa_color_convert(void *pSrc, void *pDst,    U32 PFSrc, U32 PFDst, U32 Num)
```

```
{
    IPA_CTL          = 0x00010000UL | (1 << 9);

    IPA_FMADDR     = (U32)pSrc;
    IPA_DMADDR     = (U32)pDst;

    IPA_FLOFF       = 0;
    IPA_DLOFF       = 0;

    IPA_FPCTL    = PFSrc;
    IPA_DPCTL    = PFDst;

    IPA_IMS        = (U32)(Num << 16) | 1;

    ipa_handler();
}

static LCD_PIXELINDEX *_ipa_get_pal_conv_table(const LCD_LOGPALETTE GUI_UNI_PTR
*pLogPal, const GUI_BITMAP GUI_UNI_PTR *pBitmap, int LayerIndex)
{
    void (* pFunc)(void);
    int exec_default_flag = 0;

    if(8 == pBitmap->BitsPerPixel) {
        pFunc = LCD_GetDevFunc(LayerIndex, LCD_DEVFUNC_DRAWBMP_8BPP);
        if(pFunc) {
            if(pBitmap->pPal) {
                if(pBitmap->pPal->HasTrans) {
                    exec_default_flag = 0xF;
                }
            } else {
                exec_default_flag = 0xF;
            }
        } else {
            exec_default_flag = 0xF;
        }
    } else {
        exec_default_flag = 0xF;
    }

    if(0 != exec_default_flag) {
        return LCD_GetpPalConvTable(pLogPal);
    }
```

```
    _alpha_invert_redblue_swap((U32 *)pLogPal->pPalEntries, _a_ipabuf, pLogPal->NumEntries);

    IPA_FLMADDR    = (U32)_a_ipabuf;
    IPA_FPCTL      = LAYER_PPF_RGB888 | ((pLogPal->NumEntries - 1) & 0xFF) << 8;
    IPA_FPCTL    |= (1 << 5);

    return _a_ipabuf;
}

static void _tli_layer_color_key_config(U32 TLI_Layerx, int NewState)
{
    if(DISABLE != NewState) {
        TLI_LXCTL(TLI_Layerx) |= (U32)TLI_LXCTL_CKEYEN;
    } else {
        TLI_LXCTL(TLI_Layerx) &= ~(U32)TLI_LXCTL_CKEYEN;
    }
    TLI_RL = TLI_RL_RQR;
}

static void _tli_layer_lut_config(U32 TLI_Layerx, int NewState)
{
    if(DISABLE != NewState) {
        TLI_LXCTL(TLI_Layerx) |= (U32)TLI_LXCTL_LUTEN;
    } else {
        TLI_LXCTL(TLI_Layerx) &= ~(U32)TLI_LXCTL_LUTEN;
    }
    TLI_RL = TLI_RL_RQR;
}

static void _tli_layer_position_set(int LayerIndex, int xPos, int yPos)
{
    int xSize, ySize;
    U32 HStart, HStop, VStart, VStop;

    xSize = LCD_GetXSizeEx(LayerIndex);
    ySize = LCD_GetYSizeEx(LayerIndex);
    HStart = xPos + _hbp + 1;
    HStop    = xPos + _hbp + xSize;
    VStart     = yPos + _vbp + 1;
    VStop       = yPos + _vbp + ySize;

    TLI_LXHPOS(_a_layer[LayerIndex]) &= ~(TLI_LXHPOS_WLP | TLI_LXHPOS_WRP);
    TLI_LXHPOS(_a_layer[LayerIndex]) = (HStart | (HStop << 16));
```

```
    TLI_LXVPOS(_a_layer[LayerIndex]) &= ~(TLI_LXVPOS_WTP | TLI_LXVPOS_WBP);
    TLI_LXVPOS(_a_layer[LayerIndex])  = (VStart | (VStop << 16));

    tli_reload_config(TLI_FRAME_BLANK_RELOAD_EN/*TLI_REQUEST_RELOAD_EN*/);
}

static void _tli_layer_alpha_config(int LayerIndex, int Alpha)
{
    TLI_LXSA(_a_layer[LayerIndex]) &= ~(TLI_LXSA_SA);
    TLI_LXSA(_a_layer[LayerIndex])  = (255 - Alpha) % 256;

    tli_reload_config(TLI_REQUEST_RELOAD_EN);
}

static void _tli_layer_lut_entry_config(int LayerIndex, U32 Color, int Pos)
{
    U32 r, g, b, a;

    r = (Color           & 0xff) << 16;
    g = ((Color >>  8) & 0xff) <<  8;
    b = ((Color >> 16) & 0xff);
    a = Pos << 24;
    TLI_LXLUT(_a_layer[LayerIndex]) &= ~(TLI_LXLUT_TB | TLI_LXLUT_TG | TLI_LXLUT_TR |
TLI_LXLUT_TADD);
    TLI_LXLUT(_a_layer[LayerIndex])  = r | g | b | a;

    tli_reload_config(TLI_REQUEST_RELOAD_EN);
}

static void _ipa_copy(int LayerIndex, void *pSrc, void *pDst, int xSize, int ySize, int src_offline, int
dst_offline)
{
    U32 pixel_format;

    //add if necessary
#ifdef __CM7_REV
    SCB_InvalidateDCache();
#endif

    pixel_format = _ipa_pixel_format_get(LayerIndex);
    IPA_CTL       = 0x00000000UL | (1 << 9);
    IPA_FMADDR    = (U32)pSrc;
```

```
    IPA_DMADDR    = (U32)pDst;
    IPA_FLOFF     = src_offline;
    IPA_DLOFF     = dst_offline;
    IPA_FPCTL     = pixel_format;
    IPA_IMS       = (U32)(xSize << 16) | (U16)ySize;

    ipa_handler();
}

static void _ipa_fill(int LayerIndex, void *pDst, int xSize, int ySize, int offline, U32 ColorIndex)
{
    U32 pixel_format;

    pixel_format = _ipa_pixel_format_get(LayerIndex);
    IPA_CTL       = 0x00030000UL | (1 << 9);
    IPA_DPV       = ColorIndex;
    IPA_DMADDR    = (U32)pDst;
    IPA_DLOFF     = offline;
    IPA_DPCTL     = pixel_format;
    IPA_IMS       = (U32)(xSize << 16) | (U16)ySize;

    ipa_handler();
}

static void _lcd_buf_copy(int LayerIndex, int SrcIndex, int DstIndex)
{
    U32 buf_size, scr_addr, dst_addr;

    buf_size = _a_xsize[LayerIndex] * _a_ysize[LayerIndex] * _a_pixelbytes[LayerIndex];
    scr_addr = _a_layeraddr[LayerIndex] + buf_size * SrcIndex;
    dst_addr = _a_layeraddr[LayerIndex] + buf_size * DstIndex;

    _ipa_copy(LayerIndex, (void *)scr_addr, (void *)dst_addr, _a_xsize[LayerIndex],
_a_ysize[LayerIndex], 0, 0);
    _a_bufidx[LayerIndex] = DstIndex;
}

static void _lcd_rect_copy(int LayerIndex, int x0, int y0, int x1, int y1, int xSize, int ySize)
{
    U32 buf_size, scr_addr, dst_addr;
    int offline;

    buf_size = _a_xsize[LayerIndex] * _a_ysize[LayerIndex] * _a_pixelbytes[LayerIndex];
```

```
    scr_addr = _a_layeraddr[LayerIndex] + buf_size * _a_bufidx[LayerIndex] + (y0 * _a_xsize[LayerIndex] + x0) * _a_pixelbytes[LayerIndex];
    dst_addr = _a_layeraddr[LayerIndex] + buf_size * _a_bufidx[LayerIndex] + (y1 * _a_xsize[LayerIndex] + x1) * _a_pixelbytes[LayerIndex];
    offline = _a_xsize[LayerIndex] - xSize;
    _ipa_copy(LayerIndex, (void *)scr_addr, (void *)dst_addr, xSize, ySize, offline, offline);
}

static void _lcd_rect_fill(int LayerIndex, int x0, int y0, int x1, int y1, U32 PixelIndex)
{
    U32 buf_size, dst_addr;
    int xSize, ySize;

    if(GUI_DM_XOR == GUI_GetDrawMode()) {
        LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_FILLRECT, NULL);
        LCD_FillRect(x0, y0, x1, y1);
        LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_FILLRECT, (void(*)(void))_lcd_rect_fill);
    } else {
        xSize = x1 - x0 + 1;
        ySize = y1 - y0 + 1;
        buf_size = _a_xsize[LayerIndex] * _a_ysize[LayerIndex] * _a_pixelbytes[LayerIndex];
        dst_addr = _a_layeraddr[LayerIndex] + buf_size * _a_bufidx[LayerIndex] + (y0 * _a_xsize[LayerIndex] + x0) * _a_pixelbytes[LayerIndex];
        _ipa_fill(LayerIndex, (void *)dst_addr, xSize, ySize, _a_xsize[LayerIndex] - xSize, PixelIndex);
    }
}

static void _lcd_bitmap_16bpp_draw(int LayerIndex, int x, int y, U16 const *p, int xSize, int ySize, int line_bytes)
{
    U32 buf_size, dst_addr;
    int src_offline, dst_offline;

    buf_size = _a_xsize[LayerIndex] * _a_ysize[LayerIndex] * _a_pixelbytes[LayerIndex];
    dst_addr = _a_layeraddr[LayerIndex] + buf_size * _a_bufidx[LayerIndex] + (y * _a_xsize[LayerIndex] + x) * _a_pixelbytes[LayerIndex];
    src_offline = (line_bytes / 2) - xSize;
    dst_offline = _a_xsize[LayerIndex] - xSize;
    _ipa_copy(LayerIndex, (void *)p, (void *)dst_addr, xSize, ySize, src_offline, dst_offline);
}

static void _lcd_bitmap_32bpp_draw(int LayerIndex, int x, int y, U8 const *p, int xSize, int ySize, int
```

```
line_bytes)
{
    U32 buf_size, dst_addr;
    int src_offline, dst_offline;

    buf_size = _a_xsize[LayerIndex] * _a_ysize[LayerIndex] * _a_pixelbytes[LayerIndex];
    dst_addr = _a_layeraddr[LayerIndex] + buf_size * _a_bufidx[LayerIndex] + (y * _a_xsize[LayerIndex] + x) * _a_pixelbytes[LayerIndex];
    src_offline = (line_bytes / 4) - xSize;
    dst_offline = _a_xsize[LayerIndex] - xSize;
    _ipa_copy(LayerIndex, (void *)p, (void *)dst_addr, xSize, ySize, src_offline, dst_offline);
}

void IPA_IRQHandler(void)
{
    if(RESET != (IPA_INTF & IPA_INT_FLAG_TAE)) {
        IPA_INTC |= (IPA_INT_FLAG_TAE);
    }
    if(RESET != (IPA_INTF & IPA_INT_FLAG_FTF)) {
        IPA_INTC |= (IPA_INT_FLAG_FTF);
    }
    if(RESET != (IPA_INTF & IPA_INT_FLAG_TLM)) {
        IPA_INTC |= (IPA_INT_FLAG_TLM);
    }
    if(RESET != (IPA_INTF & IPA_INT_FLAG_LAC)) {
        IPA_INTC |= (IPA_INT_FLAG_LAC);
    }
    if(RESET != (IPA_INTF & IPA_INT_FLAG_LLF)) {
        IPA_INTC |= (IPA_INT_FLAG_LLF);
    }
    if(RESET != (IPA_INTF & IPA_INT_FLAG_WCF)) {
        IPA_INTC |= (IPA_INT_FLAG_WCF);
    }
    ipa_done = 1;
}

void TLI_IRQHandler(void)
{
    U32 layer_addr;
    int i;

    TLI_INTC = (U32)TLI_INT_FLAG_LM;
    for(i = 0; i < GUI_NUM_LAYERS; i++) {
```

```
            if(_a_pendbuf[i] >= 0) {
                layer_addr = _a_layeraddr[i] + _a_xsize[i] * _a_ysize[i] * _a_pendbuf[i] *
_a_pixelbytes[i];

                TLI_LXFBADDR(_a_layer[i]) = layer_addr;
                tli_reload_config(TLI_REQUEST_RELOAD_EN);

                GUI_MULTIBUF_ConfirmEx(i, _a_pendbuf[i]);

                _a_pendbuf[i] = -1;
            }
        }
}

static void _lcd_controller_init(int LayerIndex)
{
    tli_layer_parameter_struct          tli_layer_init_struct;

    int xSize, ySize, line_bytes, pixel_bits, i;
    U32 Pixelformat, Color;
    static int init_flag;

    if(LayerIndex >= GUI_COUNTOF(_a_layer)) {
        return;
    }

    if(init_flag == 0) {
        init_flag = 1;

        tli_interrupt_enable(TLI_INT_LM);
        nvic_irq_enable(TLI_IRQn, 0, 0);
        rcu_periph_clock_enable(RCU_IPA);
        nvic_irq_enable(IPA_IRQn, 1, 0);
    }

    xSize = LCD_GetXSizeEx(LayerIndex);
    ySize = LCD_GetYSizeEx(LayerIndex);

    tli_layer_init_struct.layer_window_leftpos     = _hsync_w + _hbp + 1;
    tli_layer_init_struct.layer_window_rightpos                    =          (_active_width          +
tli_layer_init_struct.layer_window_leftpos - 1);
    tli_layer_init_struct.layer_window_toppos      = _vsync_w + _vbp + 1;
    tli_layer_init_struct.layer_window_bottompos             =          (_active_height          +
```

```
tli_layer_init_struct.layer_window_toppos - 1);

    Pixelformat = _pixel_format_get(LayerIndex);
    tli_layer_init_struct.layer_ppf = Pixelformat;

    tli_layer_init_struct.layer_sa = 0xFF;

    tli_layer_init_struct.layer_default_blue  = 0x0;
    tli_layer_init_struct.layer_default_green = 0x0;
    tli_layer_init_struct.layer_default_red   = 0x0;
    tli_layer_init_struct.layer_default_alpha = 0x0;

    line_bytes = _line_bytes_get(LayerIndex, xSize);
    tli_layer_init_struct.layer_acf1 = LAYER_ACF1_PASA;
    tli_layer_init_struct.layer_acf2 = LAYER_ACF2_PASA;
    tli_layer_init_struct.layer_frame_line_length = line_bytes + 7;
    tli_layer_init_struct.layer_frame_buf_stride_offset = line_bytes;
    tli_layer_init_struct.layer_frame_total_line_number = ySize;

    tli_layer_init_struct.layer_frame_bufaddr = _a_layeraddr[LayerIndex];
    tli_layer_init(_a_layer[LayerIndex], &tli_layer_init_struct);

    pixel_bits = LCD_GetBitsPerPixelEx(LayerIndex);
    if(pixel_bits <= 8) {
        _tli_layer_lut_config(_a_layer[LayerIndex], ENABLE);

        if(_ap_color_conv[LayerIndex] == GUICC_1616I) {
            for(i = 0; i < 16; i++) {
                Color = LCD_API_ColorConv_1616I.pfIndex2Color(i);
                _tli_layer_lut_entry_config(LayerIndex, Color, i);
            }
        }

        if(_ap_color_conv[LayerIndex] == GUICC_8666) {
            for(i = 0; i < 16; i++) {
                Color = LCD_API_ColorConv_8666.pfIndex2Color(i);
                _tli_layer_lut_entry_config(LayerIndex, Color, i);
            }
        }
    } else {
        if(_ap_color_conv[LayerIndex] == GUICC_88666I) {
            _tli_layer_lut_config(_a_layer[LayerIndex], ENABLE);
            for(i = 0; i < 256; i++) {
```

```
                    Color = LCD_API_ColorConv_8666.pfIndex2Color(i);
                    _tli_layer_lut_entry_config(LayerIndex, Color, i);
                }
            }
        }

        tli_layer_enable(_a_layer[LayerIndex]);
        tli_reload_config(TLI_REQUEST_RELOAD_EN);
}


/*********************************************************************
*
*          LCD_X_Config
*
* Purpose:
*     Called during the initialization process in order to set up the
*     display driver configuration.
*
*/
void LCD_X_Config(void)
{
    int i;

    //
    // At first initialize use of multiple buffers on demand
    //
#if (NUM_BUFFERS > 1)
    for(i = 0; i < GUI_NUM_LAYERS; i++) {
        GUI_MULTIBUF_ConfigEx(i, NUM_BUFFERS);
    }
#endif

    //
    // Set display driver and color conversion for 1st layer
    //
    GUI_DEVICE_CreateAndLink(DISPLAY_DRIVER_0, COLOR_CONVERSION_0, 0, 0);

    //
    // Set size of 1st layer
    //
    LCD_SetSizeEx(0, XSIZE_0, YSIZE_0);
    LCD_SetVSizeEx(0, XSIZE_0, YSIZE_0 * NUM_VSCREENS);
```

```
#if (GUI_NUM_LAYERS > 1)
    //
    // Set display driver and color conversion for 2nd layer
    //
    GUI_DEVICE_CreateAndLink(DISPLAY_DRIVER_1, COLOR_CONVERSION_1, 0, 1);

    //
    // Set size of 2nd layer
    //
    LCD_SetSizeEx(1, XSIZE_1, YSIZE_1);
    LCD_SetVSizeEx(1, XSIZE_1, YSIZE_1 * NUM_VSCREENS);
#endif

    //
    // Setting up VRam address and custom functions for CopyBuffer-, CopyRect- and FillRect operations
    //
    for(i = 0; i < GUI_NUM_LAYERS; i++) {
        _a_pendbuf[i] = -1;

        //
        // Set VRAM address
        //
        LCD_SetVRAMAddrEx(i, (void *)(_a_layeraddr[i]));

        //
        // Remember color depth for further operations
        //
        _a_pixelbytes[i] = LCD_GetBitsPerPixelEx(i) >> 3;

        //
        // Set custom functions for several operations
        //
        LCD_SetDevFunc(i, LCD_DEVFUNC_COPYBUFFER, (void(*)(void))_lcd_buf_copy);
        LCD_SetDevFunc(i, LCD_DEVFUNC_COPYRECT, (void(*)(void))_lcd_rect_copy);
        LCD_SetDevFunc(i, LCD_DEVFUNC_FILLRECT, (void(*)(void))_lcd_rect_fill);

        GUI_SetFuncAlphaBlending(_ipa_alpha_blend);
        GUI_SetFuncGetpPalConvTable(_ipa_get_pal_conv_table);
        GUI_SetFuncMixColorsBulk(_ipa_mix_colors_bulk);
    }

    GUI_TOUCH_Calibrate(GUI_COORD_X, 0, XSIZE_PHYS - 1, TOUCH_AD_TOP,
```

```
TOUCH_AD_BOTTOM);
    GUI_TOUCH_Calibrate(GUI_COORD_Y, 0, YSIZE_PHYS - 1, TOUCH_AD_LEFT,
TOUCH_AD_RIGHT);

#if VerticalScreen == 1
    GUI_TOUCH_SetOrientation(GUI_SWAP_XY | GUI_MIRROR_X);
#endif
}

/*********************************************************************
*
*       LCD_X_DisplayDriver
*
* Purpose:
*   This function is called by the display driver for several purposes.
*   To support the according task the routine needs to be adapted to
*   the display controller. Please note that the commands marked with
*   'optional' are not cogently required and should only be adapted if
*   the display controller supports these features.
*
* Parameter:
*   LayerIndex - Index of layer to be configured
*   Cmd        - Please refer to the details in the switch statement below
*   pData      - Pointer to a LCD_X_DATA structure
*
* Return Value:
*   < -1 - Error
*     -1 - Command not handled
*      0 - Ok
*/
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void *pData)
{
    int r = 0;

    switch(Cmd) {
    case LCD_X_INITCONTROLLER: {
        //
        // Called during the initialization process in order to set up the display controller and put it
into operation.
        //
        _lcd_controller_init(LayerIndex);
        break;
    }
```

```
    case LCD_X_SETORG: {
        //
        // Required for setting the display origin which is passed in the 'xPos' and 'yPos' element of p
        //
        LCD_X_SETORG_INFO *p;

        p = (LCD_X_SETORG_INFO *)pData;
        TLI_LXFBADDR(_a_layer[LayerIndex]) = _a_layeraddr[LayerIndex] + p->yPos * _a_xsize[LayerIndex] * _a_pixelbytes[LayerIndex];
        tli_reload_config(TLI_REQUEST_RELOAD_EN);
        break;
    }

    case LCD_X_SHOWBUFFER: {
        //
        // Required if multiple buffers are used. The 'Index' element of p contains the buffer index.
        //
        LCD_X_SHOWBUFFER_INFO *p;

        p = (LCD_X_SHOWBUFFER_INFO *)pData;
        _a_pendbuf[LayerIndex] = p->Index;
        break;
    }

    case LCD_X_SETLUTENTRY: {
        //
        // Required for setting a lookup table entry which is passed in the 'Pos' and 'Color' element of p
        //
        LCD_X_SETLUTENTRY_INFO *p;

        p = (LCD_X_SETLUTENTRY_INFO *)pData;
        _tli_layer_lut_entry_config(LayerIndex, p->Color, p->Pos);
        break;
    }
    case LCD_X_ON: {
        //
        // Required if the display controller should support switching on and off
        //
        tli_enable();
        break;
```

```
    }

    case LCD_X_OFF: {
        //
        // Required if the display controller should support switching on and off
        //
        tli_disable();
        break;
    }

    case LCD_X_SETVIS: {
        //
        // Required for setting the layer visibility which is passed in the 'OnOff' element of pData
        //
        LCD_X_SETVIS_INFO *p;

        p = (LCD_X_SETVIS_INFO *)pData;
        if(ENABLE == p->OnOff) {
            tli_layer_enable(_a_layer[LayerIndex]);
        } else {
            tli_layer_disable(_a_layer[LayerIndex]);
        }

        /* Reload shadow register */
        tli_reload_config(TLI_REQUEST_RELOAD_EN);
        break;
    }

    case LCD_X_SETPOS: {
        //
        // Required for setting the layer position which is passed in the 'xPos' and 'yPos' element of
pData
        //
        LCD_X_SETPOS_INFO *p;

        p = (LCD_X_SETPOS_INFO *)pData;
        _tli_layer_position_set(LayerIndex, p->xPos, p->yPos);
        break;
    }

    case LCD_X_SETSIZE: {
        //
        // Required for setting the layer position which is passed in the 'xPos' and 'yPos' element of
```

```
pData
            //
            LCD_X_SETSIZE_INFO *p;
            int xPos, yPos;

            GUI_GetLayerPosEx(LayerIndex, &xPos, &yPos);
            p = (LCD_X_SETSIZE_INFO *)pData;
            _a_xsize[LayerIndex] = p->xSize;
            _a_ysize[LayerIndex] = p->ySize;
            _tli_layer_position_set(LayerIndex, xPos, yPos);
            break;
        }

        case LCD_X_SETALPHA: {
            //
            // Required for setting the alpha value which is passed in the 'Alpha' element of pData
            //
            LCD_X_SETALPHA_INFO *p;

            p = (LCD_X_SETALPHA_INFO *)pData;
            _tli_layer_alpha_config(LayerIndex, p->Alpha);
            break;
        }

        case LCD_X_SETCHROMAMODE: {
            //
            // Required for setting the chroma mode which is passed in the 'ChromaMode' element of
pData
            //
            LCD_X_SETCHROMAMODE_INFO *p;

            p = (LCD_X_SETCHROMAMODE_INFO *)pData;
            _tli_layer_color_key_config(_a_layer[LayerIndex], (p->ChromaMode != 0) ? ENABLE :
DISABLE);
            break;
        }

        case LCD_X_SETCHROMA: {
            //
            // Required for setting the chroma value which is passed in the 'ChromaMin' and
'ChromaMax' element of pData
            //
            LCD_X_SETCHROMA_INFO *p;
```

```
        U32 color;

        p = (LCD_X_SETCHROMA_INFO *)pData;
        color = ((p->ChromaMin & 0xFF0000) >> 16) | (p->ChromaMin & 0x00FF00) |
((p->ChromaMin & 0x0000FF) << 16);
        TLI_LXCKEY(_a_layer[LayerIndex]) = color;
        tli_reload_config(TLI_REQUEST_RELOAD_EN);
        break;
    }

    default:
        r = -1;
    }

    return r;
}
```

3.5. Modify GUIConf.c

The GUIConf.c file defines the usable memory size for the GUI, which can be customized based on actual application scenarios and hardware resources.

3.6. Modify GUI_X_Touch_Analog.c

The GUI_X_Touch_Analog.c file defines the X-axis and Y-axis AD values required by the resistive touchscreen to be transmitted to the upper layer. The appropriate interface can be called to return the correct values.

3.7. Modify main.c

First, cache needs to be configured by enabling I-Cache and D-Cache.

Then, configure the MPU. Detailed configuration can be referenced in ***Table 3-3. MPU Configuration***.

Table 3-3. MPU Configuration

```
void mpu_config(void)
{
    mpu_region_init_struct mpu_init_struct;
    mpu_region_struct_para_init(&mpu_init_struct);

    /* disable the MPU */
```

```
    ARM_MPU_Disable();
    ARM_MPU_SetRegion(0, 0);

    /* configure the MPU attributes for Reserved, no access */
    mpu_init_struct.region_base_address    = 0x0;
    mpu_init_struct.region_size             = MPU_REGION_SIZE_4GB;
    mpu_init_struct.access_permission       = MPU_AP_NO_ACCESS;
    mpu_init_struct.access_bufferable       = MPU_ACCESS_NON_BUFFERABLE;
    mpu_init_struct.access_cacheable        = MPU_ACCESS_NON_CACHEABLE;
    mpu_init_struct.access_shareable        = MPU_ACCESS_SHAREABLE;
    mpu_init_struct.region_number           = MPU_REGION_NUMBER0;
    mpu_init_struct.subregion_disable       = 0x87;
    mpu_init_struct.instruction_exec        = MPU_INSTRUCTION_EXEC_NOT_PERMIT;
    mpu_init_struct.tex_type                = MPU_TEX_TYPE0;
    mpu_region_config(&mpu_init_struct);
    mpu_region_enable();

    /* configure the MPU attributes for AXI-SRAM and AXI-SRAMS Write through, no write allocate,
*/
    mpu_init_struct.region_base_address    = 0x24000000;
    mpu_init_struct.region_size             = MPU_REGION_SIZE_1MB;
    mpu_init_struct.access_permission       = MPU_AP_FULL_ACCESS;
    mpu_init_struct.access_bufferable       = MPU_ACCESS_NON_BUFFERABLE;
    mpu_init_struct.access_cacheable        = MPU_ACCESS_CACHEABLE;
    mpu_init_struct.access_shareable        = MPU_ACCESS_NON_SHAREABLE;
    mpu_init_struct.region_number           = MPU_REGION_NUMBER1;
    mpu_init_struct.subregion_disable       = 0x0;
    mpu_init_struct.instruction_exec        = MPU_INSTRUCTION_EXEC_PERMIT;
    mpu_init_struct.tex_type                = MPU_TEX_TYPE0;
    mpu_region_config(&mpu_init_struct);
    mpu_region_enable();

    /* configure the MPU attributes for SDRAM Write through, no write allocate,*/
    mpu_init_struct.region_base_address    = 0xC0000000;
    mpu_init_struct.region_size             = MPU_REGION_SIZE_32MB;
    mpu_init_struct.access_permission       = MPU_AP_FULL_ACCESS;
    mpu_init_struct.access_bufferable       = MPU_ACCESS_NON_BUFFERABLE;
    mpu_init_struct.access_cacheable        = MPU_ACCESS_CACHEABLE;
    mpu_init_struct.access_shareable        = MPU_ACCESS_NON_SHAREABLE;
    mpu_init_struct.region_number           = MPU_REGION_NUMBER2;
    mpu_init_struct.subregion_disable       = 0x0;
    mpu_init_struct.instruction_exec        = MPU_INSTRUCTION_EXEC_NOT_PERMIT;
    mpu_init_struct.tex_type                = MPU_TEX_TYPE0;
```

```
    mpu_region_config(&mpu_init_struct);
    mpu_region_enable();

    /* enable the MPU */
    ARM_MPU_Enable(MPU_MODE_PRIV_DEFAULT);
}
```

Next, initialize SDRAM, LCD, touch functionality, etc. Detailed instructions can be referenced in ***Table 3-4.*** 。

Table 3-4. Initialization code

```
int main(void)
{
    ......
    rcu_periph_clock_enable(RCU_IPA);
    /* configure the EXMC access mode */
    exmc_synchronous_dynamic_ram_init(EXMC_SDRAM_DEVICE0);

    /* configure touch panel port */
    touch_panel_gpio_config();
    /* initialize LCD */
    lcd_init();
    /* configure TLI and enable TLI interrupt */
    tli_line_mark_set(0);
    tli_interrupt_enable(TLI_INT_LM | TLI_INT_FE | TLI_INT_TE);
    nvic_irq_enable(TLI_IRQn, 1, 0);
    nvic_irq_enable(TLI_ER_IRQn, 1, 0);
    ......
}
```

Finally, add the clear screen test code and execute the MainTask GUI task. Detailed instructions can be referenced.

Next, initialize SDRAM, LCD, touch functionality, etc. For details, refer to ***Table 3-5. GUI code***.

Table 3-5. GUI code

```
int main(void)
{
    ......
    /* display test */
    GUI_Init();
    GUI_SetBkColor(GUI_LIGHTBLUE);
    GUI_Clear();

    MainTask();
    ......
```

```
}
```

3.8. Modify gd32h7xx_it.c

In the 1ms Systick interrupt service routine, emWin needs to be provided with a heartbeat. Detailed modifications to the Systick and TLI interrupt service routines are shown in ***Table 3-6.*** .

Table 3-6. Interrupt service routine

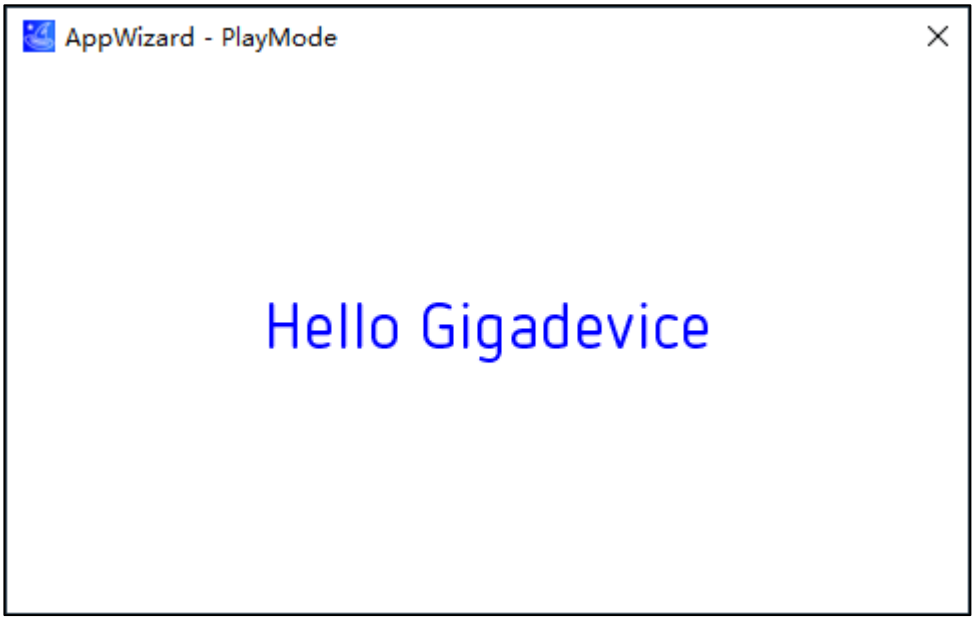
```
extern __IO int32_t OS_TimeMS;
void SysTick_Handler(void)
{
    static uint16_t count_time = 0;
    delay_decrement();
    ++OS_TimeMS;
    ++count_time;
    if (count_time == 10)
    {
        GUI_TOUCH_Exec();
        count_time = 0;
    }
}

void TLI_ER_IRQHandler(void)
{
    if(SET == tli_interrupt_flag_get(TLI_INT_FLAG_FE)){
        tli_interrupt_flag_clear(TLI_INT_FLAG_FE);
    }
    if(SET == tli_interrupt_flag_get(TLI_INT_FLAG_TE)){
        tli_interrupt_flag_clear(TLI_INT_FLAG_TE);
    }
}
```

4. Demonstration

In AppWizard, click  to run, and the simulation effect is shown in ***Figure 4-1. AppWizard Simulator***.

Figure 4-1. AppWizard Simulator



Export the output from the AppWizard project, compile it, and then flash the program to the development board. The specific effect is shown in ***Figure 4-2. Evaluation board demonstration***. Subsequent development can begin with page design in AppWizard, followed by exporting intermediate files, and finally compiling successfully in the IDE and downloading to the chip.

Figure 4-2. Evaluation board demonstration

5. Revision history

Table 5-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Mar.30 2025
1.1	Add ***Table 1-1. Applicable product***	Feb.12 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.