GigaDevice Semiconductor Inc.

GDemWin GD32H7xx 系列移植指南

应用笔记

AN212

1.1 版本

（2026 年 2 月）

目录

目录 **2**
图索引 **3**
表索引 **4**
1. 简介 **5**
2. 开发环境 **6**
2.1. 硬件环境 **6**
2.2. 软件环境 **6**
3. 移植 **7**
3.1. 工具安装 **7**
3.2. 驱动准备 **7**
3.2.1. GDemWin 软件库 7
3.2.2. TLI LCD 驱动 8
3.2.3. SDRAM 驱动 8
3.2.4. 触摸驱动 8
3.3. 工程搭建 **8**
3.3.1. 新建 AppWizard 工程 9
3.3.2. 导入 AppWizard 工程 15
3.3.3. 修改 Application.c 17
3.4. 修改 LCDConf_Lin_Template **17**
3.5. 修改 GUIConf.c **38**
3.6. 修改 GUI_X_Touch_Analog.c **38**
3.7. 修改 main.c **38**
3.8. 修改 gd32h7xx_it.c **41**
4. 演示效果 **42**
5. 版本历史 **43**

图索引

图 2-1. 从官网获得 GDemWin 资料 6
图 3-1. Tool 安装包 7
图 3-2. 新建 GDemWin 文件夹 7
图 3-3. 拷贝数据到 GDemWin 文件夹下 8
图 3-4. 初始工程模板 8
图 3-5. 添加 Groups 和 Files 9
图 3-6. 添加驱动文件 9
图 3-7. AppWizard 支持的组件 10
图 3-8. 新建 AppWizard 工程 11
图 3-9. 新建 Screen 12
图 3-10. 设置 Box 颜色 12
图 3-11. 设置文字显示 13
图 3-12. 添加并选择文字 13
图 3-13. 设置字体 14
图 3-14.字体属性调整 14
图 3-15.工程导出并保存 15
图 3-16.导出后的工程文件 15
图 3-17.添加 AppWizard Groups 16
图 3-18.添加包含路径 16
图 4-1. AppWizard 模拟器效果 42
图 4-2. 开发板演示效果 42

表索引

表 1-1. 适用产品 5
表 3-1. Application.c 修改 17
表 3-2. LCDConf_Lin_Template 修改 17
表 3-3. MPU 配置 38
表 3-4. 初始化代码 40
表 3-5. GUI 代码 40
表 3-6. 中断服务程序 41
表 5-1. 版本历史 43

1. 简介

本文主要介绍如何在GD32H7xx系列使用GDemWin作为GUI中间件来开发UI项目。

GD32H737/757/759系列为基于Arm® Cortex®-M7内核的超高性能微控制器。GD32H7系列MCU具备卓越的处理能效、丰富连接特性及多重安全机制，可广泛应用于数据信号处理、电机变频、电源、储能系统、音频视频、图形图像处理等各类应用。

emWin是由德国SEGGER公司开发，可为图形LCD设计提供高级支持，极大地简化了LCD设计。借助emWin这类图形库以及配套提供的工具，如AppWizard，能很方便地进行模块化设计，既能提高设计界面图形质量，减少产品迭代更新成本，还可以大大的减少开发时间。

GDemWin为GigaDevice向SEGGER获得许可之后更名而来。用户在GD32 MCU Arm® Cortex®-M的芯片上使用GDemWin软件库进行项目开发，可以免费商用。本文档适用型号参考*表1-1. 适用产品*。

表 1-1. 适用产品

产品系列	型号
GD32H77x	GD32H779, GD32H77D, GD32H77E系列
GD32H78x	GD32H789, GD32H78D, GD32H78E系列
GD32H73x	GD32H737系列
GD32H75x	GD32H757, GD32H759, GD32H75E系列

2. 开发环境

2.1. 硬件环境

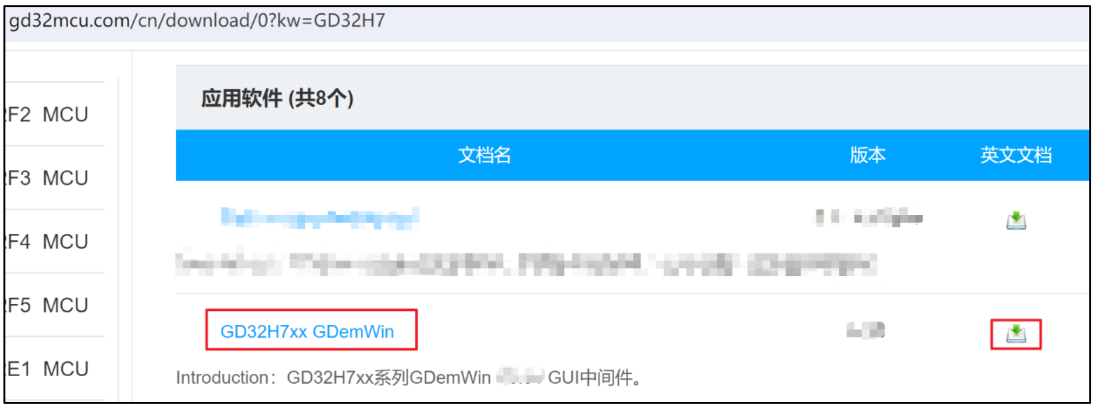
- 硬件开发板：GD32H759I-EVAL 开发板
- 烧录工具：板载 GD-Link

2.2. 软件环境

- 操作系统：Win10-64 位
- 开发环境：KEIL 5.29
- 固件库：GD32H7xx_Firmware_Library
- GUI：GDemWin

GDemWin 及配套工具的获取链接为：www.gd32mcu.com/cn/download/0?kw=GD32H7，然后找到 GD32H7xx GDemWin，具体为下图所示。

图 2-1. 从官网获得 GDemWin 资料


gd32mcu.com/cn/download/0?kw=GD32H7

文档名	版本	英文文档
GD32H7xx GDemWin		

Introduction：GD32H7xx系列GDemWin GUI中间件。

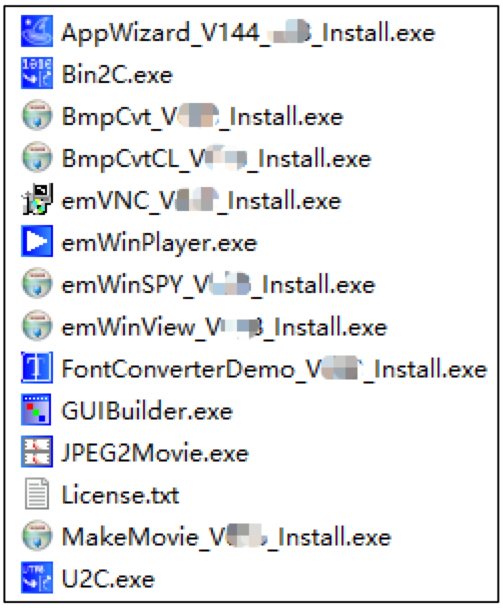
GD32H7xx_Firmware_Library 的获取与 GDemWin 及配套工具的获取类似，均可以从网站链接为：www.gd32mcu.com 下载获得。

3. 移植

3.1. 工具安装

根据需求选择安装\Tool 目录下的安装包，具体如下图所示。

图 3-1. Tool 安装包


AppWizard_V144_[illegible]_Install.exe
Bin2C.exe
BmpCvt_V[illegible]_Install.exe
BmpCvtCL_V[illegible]_Install.exe
emVNC_V[illegible]_Install.exe
emWinPlayer.exe
emWinSPY_V[illegible]_Install.exe
emWinView_V[illegible]_Install.exe
FontConverterDemo_V[illegible]_Install.exe
GUIBuilder.exe
JPEG2Movie.exe
License.txt
MakeMovie_V[illegible]_Install.exe
U2C.exe

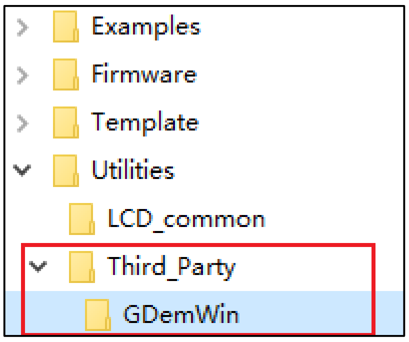
3.2. 驱动准备

先将下载的 GD32H7xx Firmware Library 固件包解压。

3.2.1. GDemWin 软件库

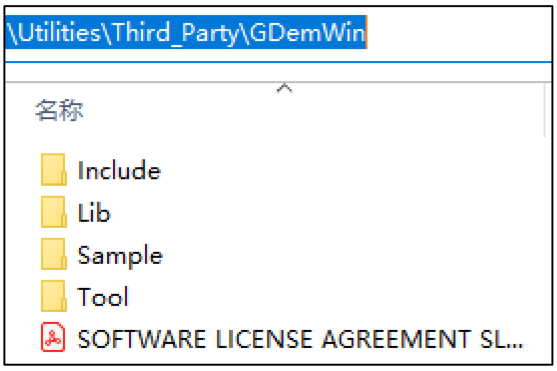
在 Utilities 文件夹新建 Third_Party 文件夹，进入\Utilities\Third_Party 目录，新建 GDemWin 文件夹。

图 3-2. 新建 GDemWin 文件夹


Examples
Firmware
Template
Utilities
LCD_common
Third_Party
GDemWin

将从 www.gd32mcu.com/cn/download/0?kw=GD32H7 下载得到的 GD32H7xx GDemWin 内容拷贝到 GDemWin 文件夹下。

图 3-3. 拷贝数据到 GDemWin 文件夹下



\Utilities\Third_Party\GDemWin

名称

- Include
- Lib
- Sample
- Tool
- SOFTWARE LICENSE AGREEMENT SL...

3.2.2. TLI LCD 驱动

GD32H759I-EVAL 开发板上搭载的是 RGB 接口的 LCD，驱动可以直接使用 Utilities 文件夹下的 gd32h759i_lcd_eval.c/.h 文件。

3.2.3. SDRAM 驱动

SDRAM 驱动可以参考\Examples\EXMC\SDRAM 下的 exmc_sdram.c/.h。并将其拷贝到 Utilities 文件夹下。

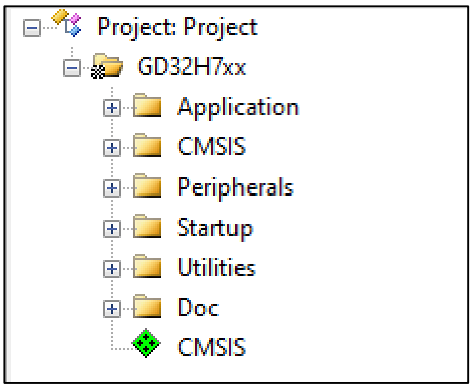
3.2.4. 触摸驱动

GD32H759I-EVAL 开发板上搭载的屏幕触摸芯片使用的是 SPI 接口与 MCU 通信，参考 touch_panel.c/.h。

3.3. 工程搭建

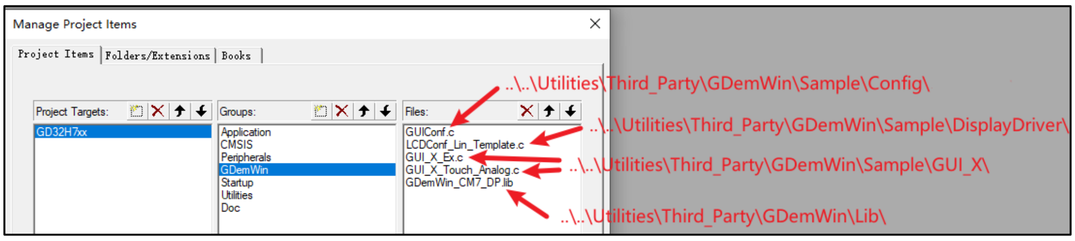
打开\Template\Keil_project 的 Keil 工程，模板工程如下图所示。

图 3-4. 初始工程模板



```
Project: Project
  GD32H7xx
    Application
    CMSIS
    Peripherals
    Startup
    Utilities
    Doc
    CMSIS
```

图 3-5. 添加 Groups 和 Files



Project Targets: GD32H7xx

Groups: Application, CMSIS, Peripherals, GDemWin, Startup, Utilities, Doc

Files: GUIConf.c, LCDConf_Lin_Template.c, GUI_X_Ex.c, GUI_X_Touch_Analog.c, GDemWin_CM7_DP.lib

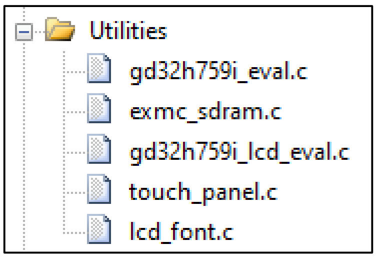
..\..\Utilities\Third_Party\GDemWin\Sample\Config\

..\..\Utilities\Third_Party\GDemWin\Sample\DisplayDriver\

..\..\Utilities\Third_Party\GDemWin\Sample\GUI_X\

..\..\Utilities\Third_Party\GDemWin\Lib\

图 3-6. 添加驱动文件



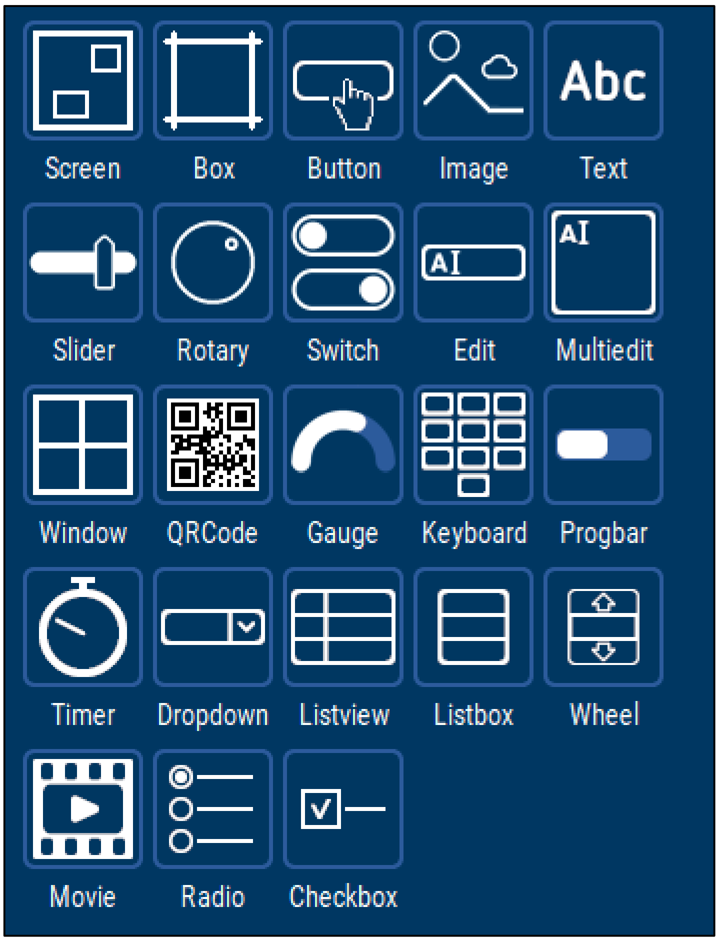
Utilities
- gd32h759i_eval.c
- exmc_sdram.c
- gd32h759i_lcd_eval.c
- touch_panel.c
- lcd_font.c

3.3.1. 新建 AppWizard 工程

AppWizard 是 emWin 的界面开发工具，AppWizard 能以最简单的方式在任何嵌入式系统上创建高效、高质量的图形用户界面，用于创建完整且随时可运行的应用程式，使得复杂的图形应用程序变得简单，降低 UI 开发难度并且提升效率。

安装好 AppWizard 后打开 AppWizard 软件，可以看到目前 AppWizard 支持的组件如下所示，用户指南请参考《UM03003 AppWizard User Guide & Reference Manual》。

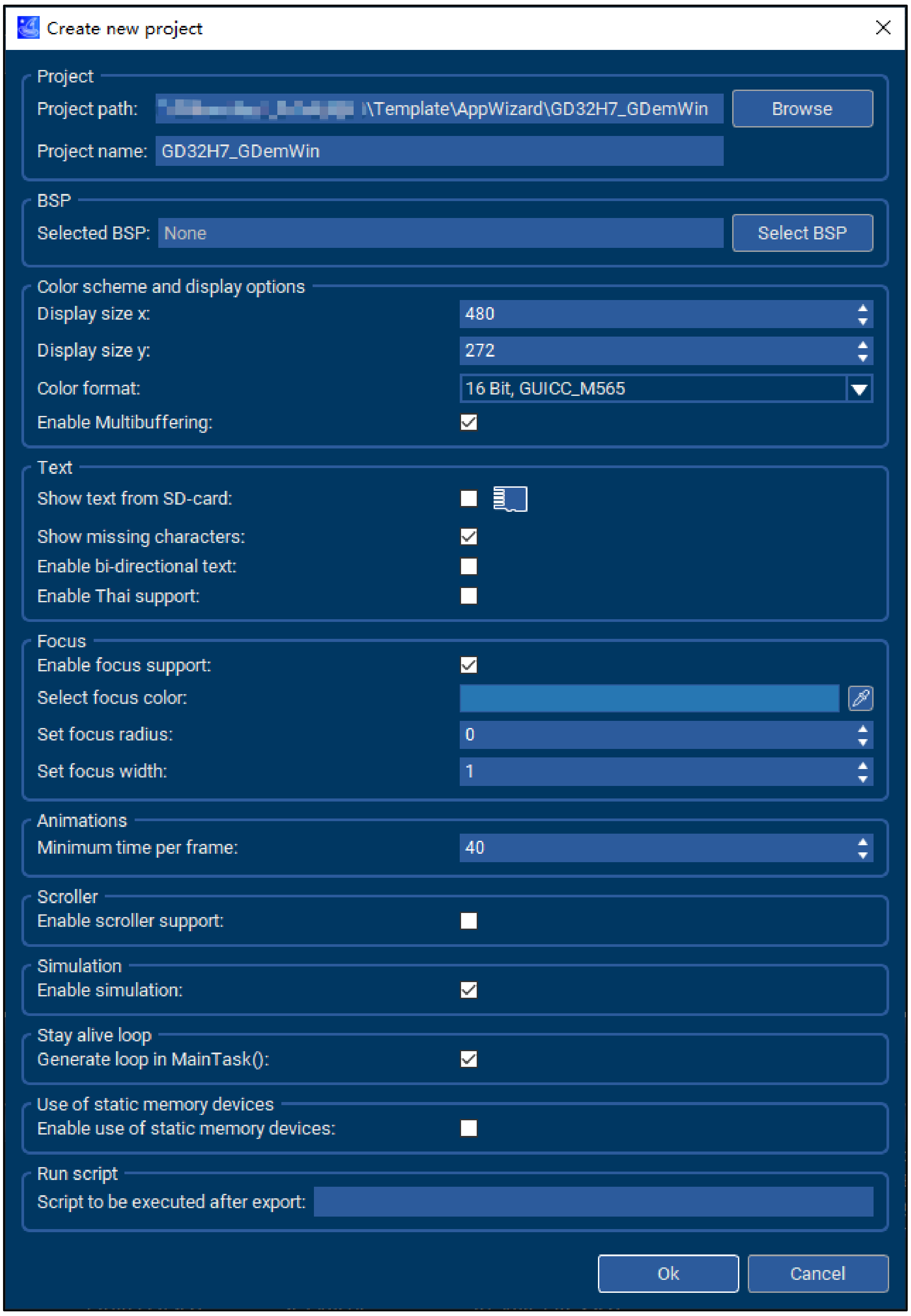
图 3-7. AppWizard 支持的组件



为了方便 AppWizard 生成的页面可以很方便地在开发板上验证显示效果，在 Template 目录下，新建 AppWizard 文件夹用于存放 AppWizard 工程以及生成的输出文件。

首先，新建 AppWizard 工程，工程配置如下所示。

图 3-8. 新建 AppWizard 工程


Create new project

Project
Project path: [redacted]\Template\AppWizard\GD32H7_GDemWin
Project name: GD32H7_GDemWin

BSP
Selected BSP: None

Color scheme and display options
Display size x: 480
Display size y: 272
Color format: 16 Bit, GUICC_M565
Enable Multibuffering: ✓

Text
Show text from SD-card: ☐
Show missing characters: ✓
Enable bi-directional text: ☐
Enable Thai support: ☐

Focus
Enable focus support: ✓
Select focus color:
Set focus radius: 0
Set focus width: 1

Animations
Minimum time per frame: 40

Scroller
Enable scroller support: ☐

Simulation
Enable simulation: ✓

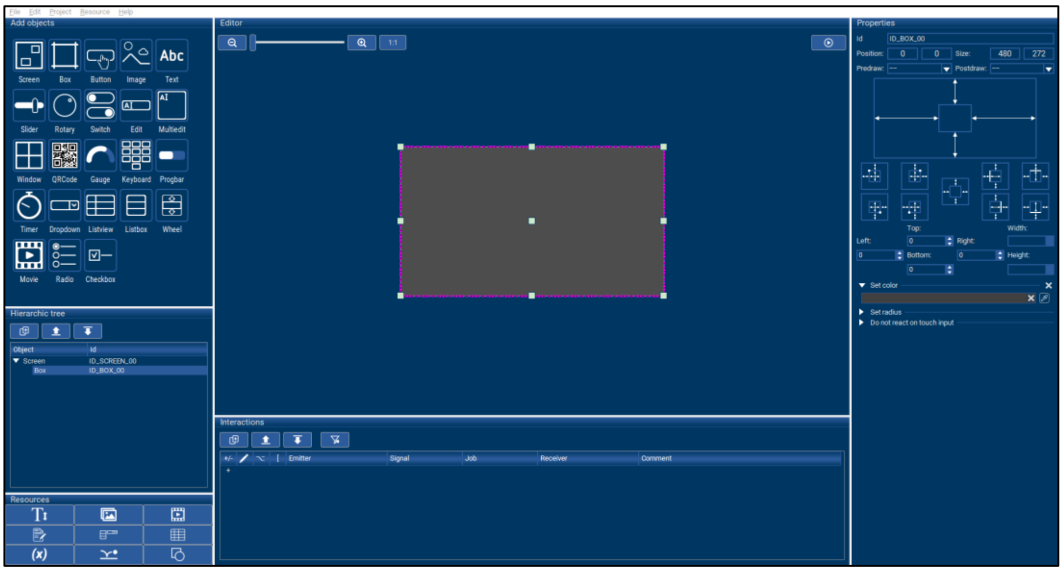
Stay alive loop
Generate loop in MainTask(): ✓

Use of static memory devices
Enable use of static memory devices: ☐

Run script
Script to be executed after export:

在 AppWizard 工程中新建一个新的 Screen 组件，然后新建 Box 组件，效果如下。

图 3-9. 新建 Screen

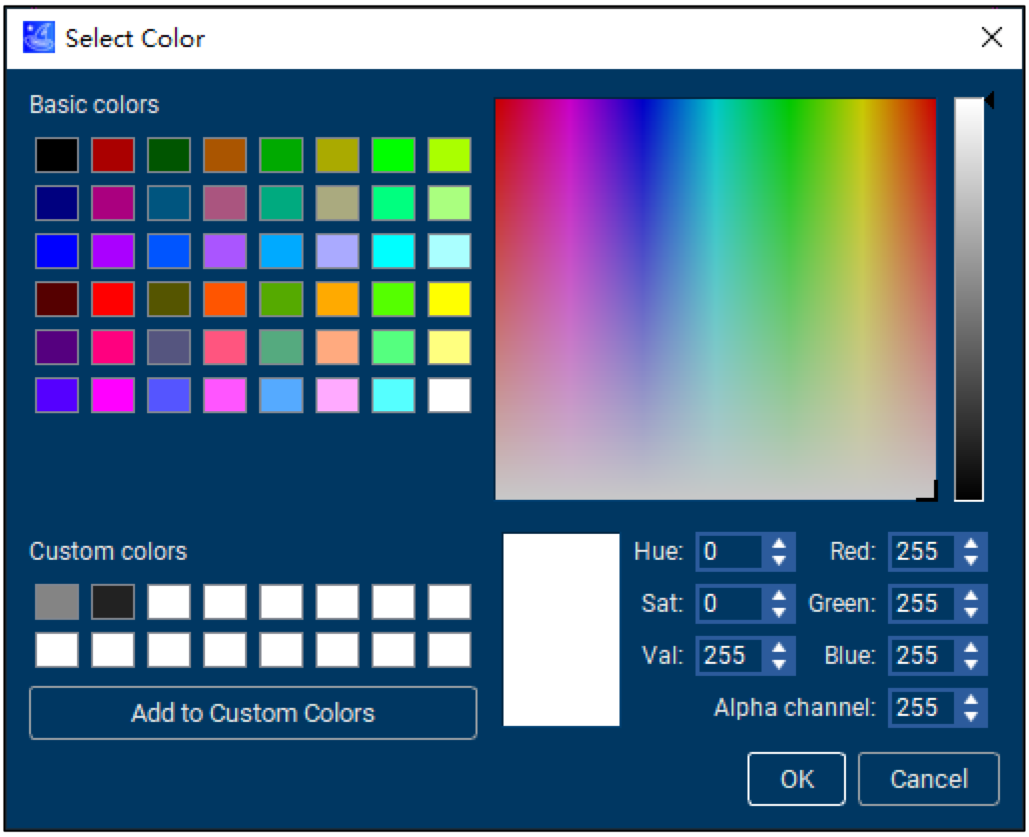


Object	Id
Screen	ID_SCREEN_00
Box	ID_BOX_00

Id: ID_BOX_00
Position: 0 0 Size: 480 272

在右边栏 Properities 中设置 Box 的颜色，改为白色（255,255,255），具体如下。

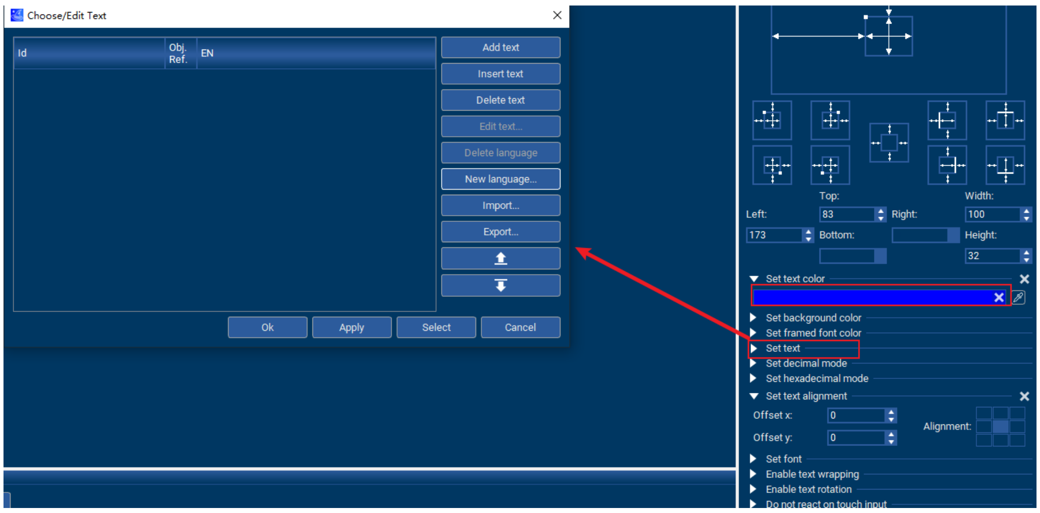
图 3-10. 设置 Box 颜色



Hue: 0
Sat: 0
Val: 255
Red: 255
Green: 255
Blue: 255
Alpha channel: 255

新建 Text 组件，在右边栏 Properities 中设置 Text 颜色为蓝色（0,0,255），再点击 Set Text 设置文字显示，并选择“New language...”，定义文字名称，具体如下所示。

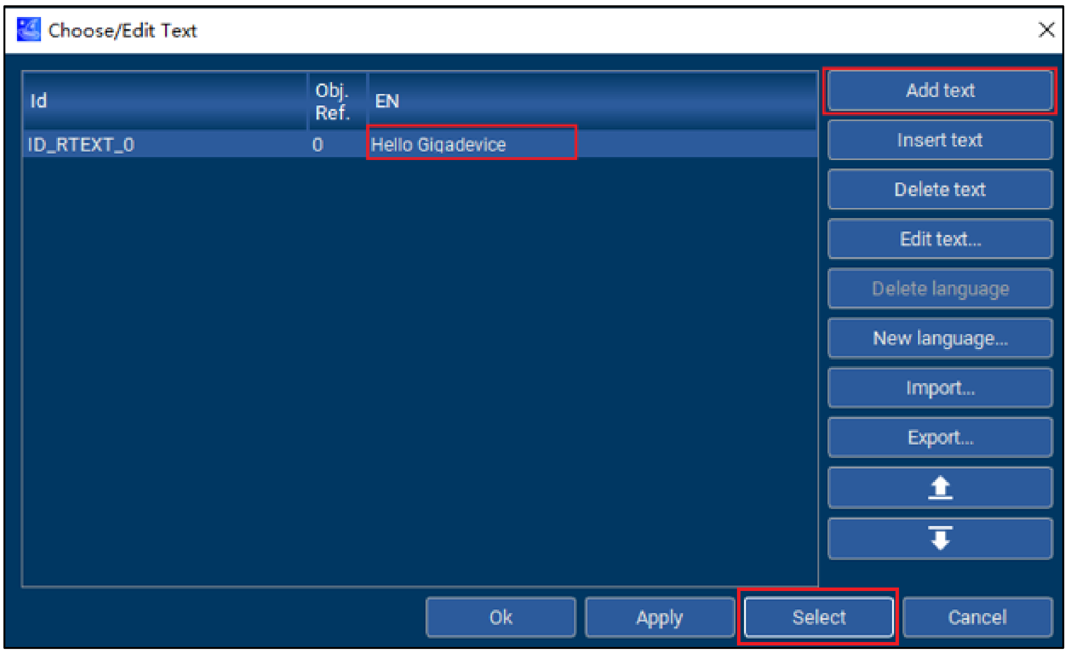
图 3-11. 设置文字显示



Left: 173
Top: 83
Right:
Bottom:
Width: 100
Height: 32
Offset x: 0
Offset y: 0

选择“Add text”，并为 ID_RTEXT_0 键入需要显示的文字，并点击 Select 按钮选择文字，具体如下所示。

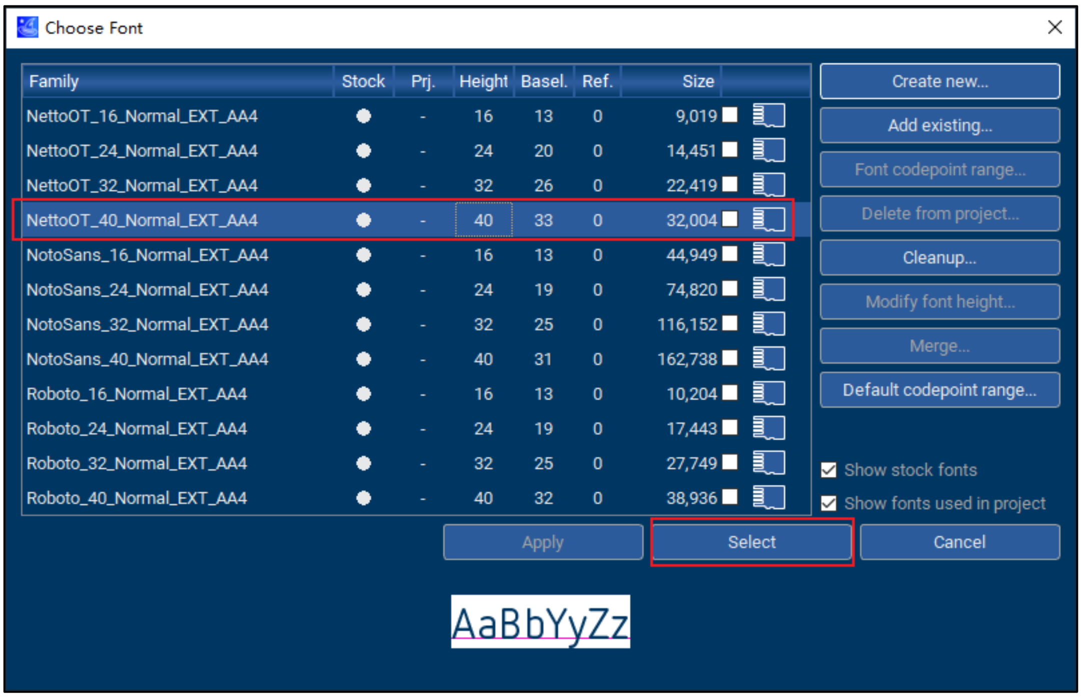
图 3-12. 添加并选择文字



Id	Obj. Ref.	EN
ID_RTEXT_0	0	Hello Gigadevice

可以根据需要选择不同的字体，点击右边栏 Properities 中的“Set font”，选择合适的字体，最后点击 Select 按键确认字体，具体如下所示。

图 3-13. 设置字体



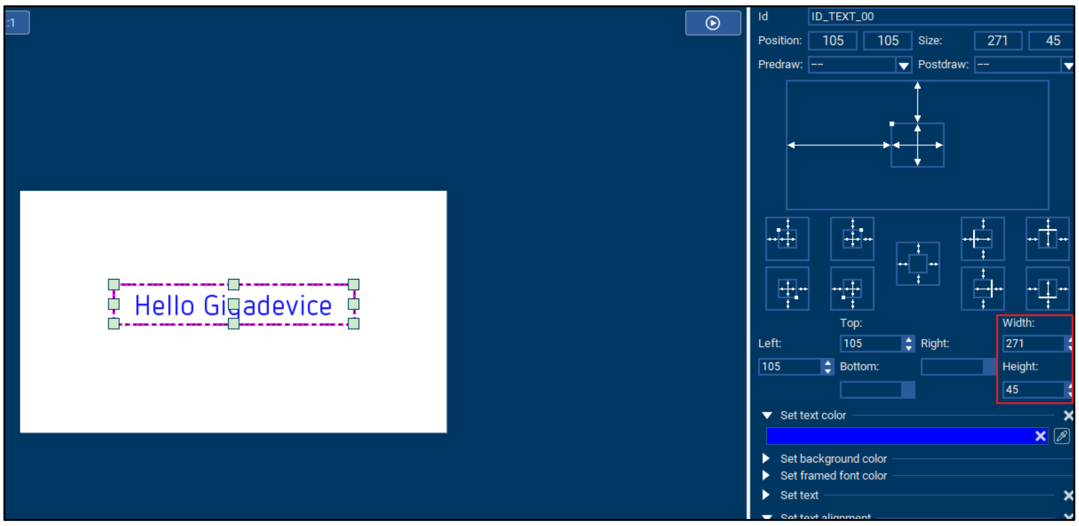
Choose Font

Family	Stock	Prj.	Height	Basel.	Ref.	Size
NettoOT_16_Normal_EXT_AA4	●	-	16	13	0	9,019
NettoOT_24_Normal_EXT_AA4	●	-	24	20	0	14,451
NettoOT_32_Normal_EXT_AA4	●	-	32	26	0	22,419
NettoOT_40_Normal_EXT_AA4	●	-	40	33	0	32,004
NotoSans_16_Normal_EXT_AA4	●	-	16	13	0	44,949
NotoSans_24_Normal_EXT_AA4	●	-	24	19	0	74,820
NotoSans_32_Normal_EXT_AA4	●	-	32	25	0	116,152
NotoSans_40_Normal_EXT_AA4	●	-	40	31	0	162,738
Roboto_16_Normal_EXT_AA4	●	-	16	13	0	10,204
Roboto_24_Normal_EXT_AA4	●	-	24	19	0	17,443
Roboto_32_Normal_EXT_AA4	●	-	32	25	0	27,749
Roboto_40_Normal_EXT_AA4	●	-	40	32	0	38,936

☑ Show stock fonts
☑ Show fonts used in project

由于初始 Text 的宽度和高度是固定的，当字体字号被改大了之后会出现显示不全的现象，可以通过设定 Text 的宽度和高度属性来让其正常显示，具体如下所示。

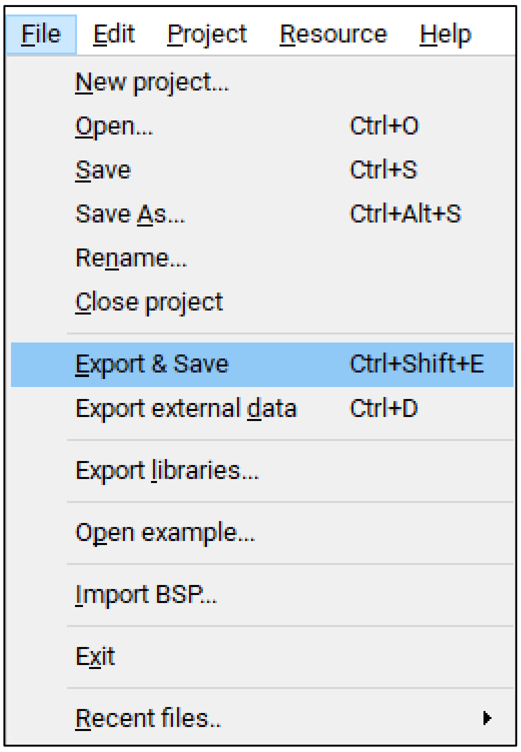
图 3-14.字体属性调整



Id: ID_TEXT_00
Position: 105 105 Size: 271 45
Predraw: -- Postdraw: --
Top: 105
Left: 105
Right:
Bottom:
Width: 271
Height: 45

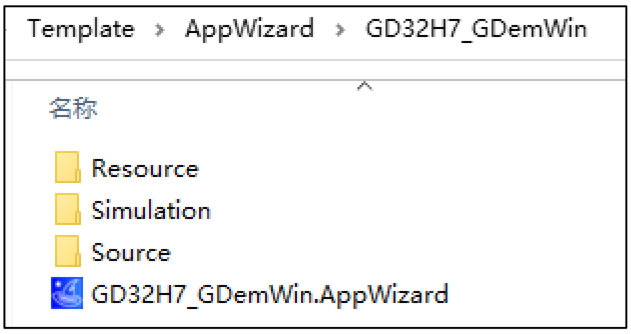
选择 Export & Save，将导出工程生成的文件，具体如下图所示。

图 3-15.工程导出并保存



导出工程之后生成的文件如下图所示。

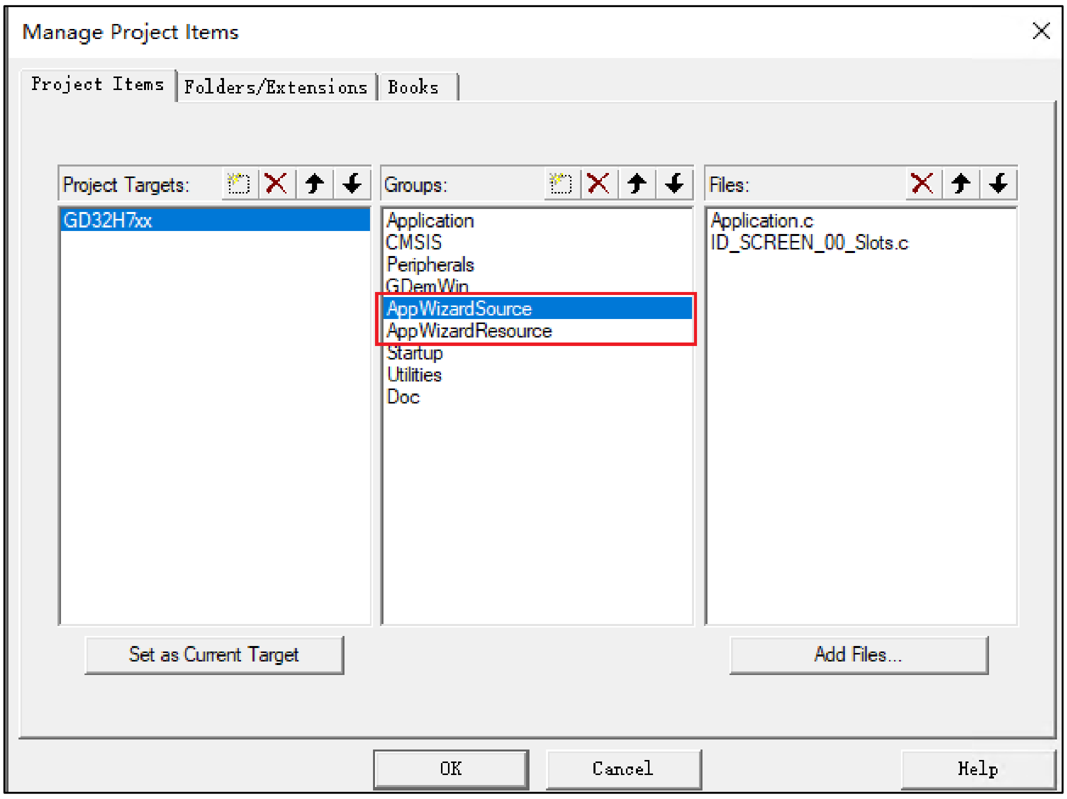
图 3-16.导出后的工程文件



3.3.2. 导入 AppWizard 工程

在 Keil 工程中新建 AppWizardSource 和 AppWizardResource，添加 AppWizard 导出的文件，同时在 Keil 工程里添加包含路径，具体如下图所示。

图 3-17.添加 AppWizard Groups

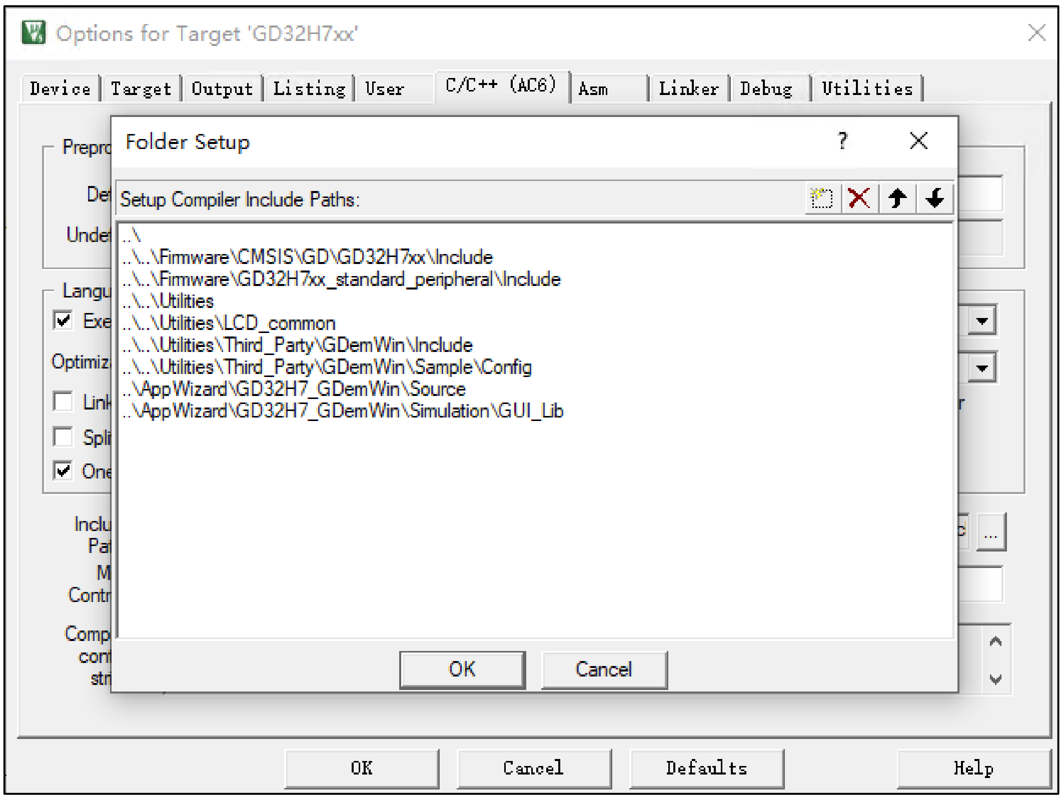


Project Targets: GD32H7xx

Groups:
- Application
- CMSIS
- Peripherals
- GDemWin
- AppWizardSource
- AppWizardResource
- Startup
- Utilities
- Doc

Files:
- Application.c
- ID_SCREEN_00_Slots.c

图 3-18.添加包含路径



Setup Compiler Include Paths:

```
..\
..\..\Firmware\CMSIS\GD\GD32H7xx\Include
..\..\Firmware\GD32H7xx_standard_peripheral\Include
..\..\Utilities
..\..\Utilities\LCD_common
..\..\Utilities\Third_Party\GDemWin\Include
..\..\Utilities\Third_Party\GDemWin\Sample\Config
..\AppWizard\GD32H7_GDemWin\Source
..\AppWizard\GD32H7_GDemWin\Simulation\GUI_Lib
```

3.3.3. 修改 Application.c

由于本手册并未演示文件系统相关内容，建议在 Application.c 中添加如*表 3-1. Application.c 修改*所示代码避免编译报错。

表 3-1. Application.c 修改

```
#include "Application.h"

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

#ifndef WIN32
void APPW_X_FS_Init(void)
{

}
#endif
```

3.4. 修改 LCDConf_Lin_Template

LCDConf_Lin_Template 为底层驱动的核心部分，需要根据屏的参数进行 LCD 的配置，同时使用芯片的 IPA 模块对部分函数进行加速优化，代码的修改如*表 3-2. LCDConf_Lin_Template 修改*所示。

表 3-2. LCDConf_Lin_Template 修改

```
#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 = 0xFFFFFFFF;
    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. 修改 GUIConf.c

GUIConf.c 文件定义了 GUI 可用的空间大小，可根据实际应用场景以及硬件资源进行自定义地设定。

3.6. 修改 GUI_X_Touch_Analog.c

GUI_X_Touch_Analog.c 文件定义了电阻屏所需要传到上层的 X 轴和 Y 轴的 AD 值，可调用相应接口将正确值返回。

3.7. 修改 main.c

首先需要先配置 cache，打开 I-Cache 和 D-Cache。

然后配置 MPU，具体配置可参考*表 3-3. MPU 配置*所示。

表 3-3. MPU 配置

```
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);
}
```

接下来需要初始化 SDRAM、LCD、触摸等，具体可以参考*表 3-4. 初始化代码*。

表 3-4. 初始化代码

```
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);
    ......
}
```

最后新增清屏测试代码，以及执行 MainTask GUI 任务，具体可以参考。

接下来需要初始化 SDRAM、LCD、触摸等，具体可以参考*表 3-5. GUI 代码*。

表 3-5. GUI 代码

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

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

3.8. 修改 gd32h7xx_it.c

在 Systick 1ms 中断服务程序里，需要给 emWin 提供心跳，详细的 Systick 和 TLI 的中断服务程序的修改如*表 3-6. 中断服务程序*所示。

表 3-6. 中断服务程序

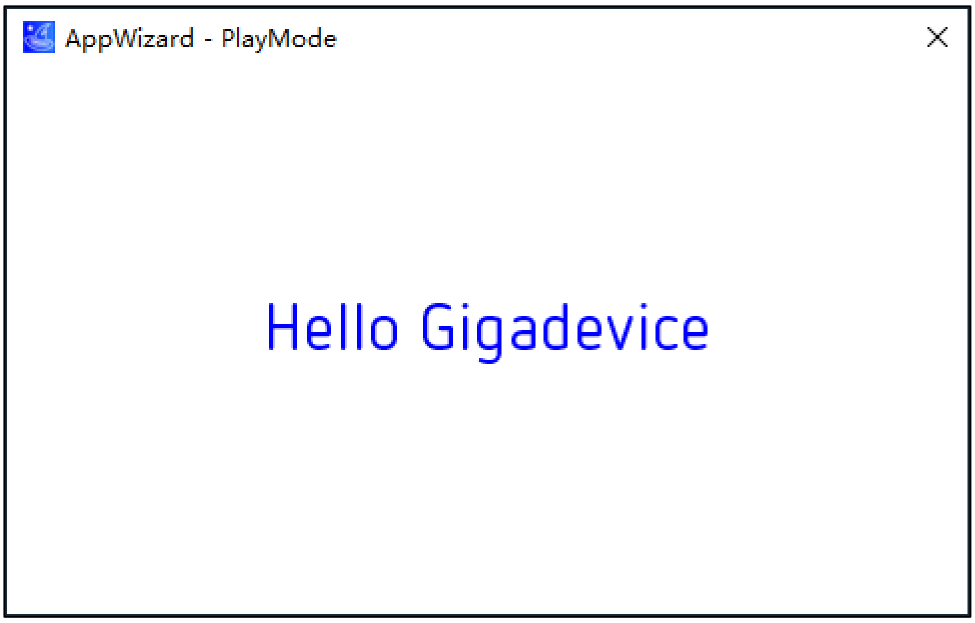
```
extern __IO int32_t OS_TimeMS;
void SysTick_Handler(void)
{
    static uint16_t count_time = 0;
    delay_decrement();
    ++OS_TimeMS;
    ++count_time;
    if (count_time == 10)
    {
        GUI_TOUCH_Exec();
        count_time = 0;
    }
}

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. 演示效果

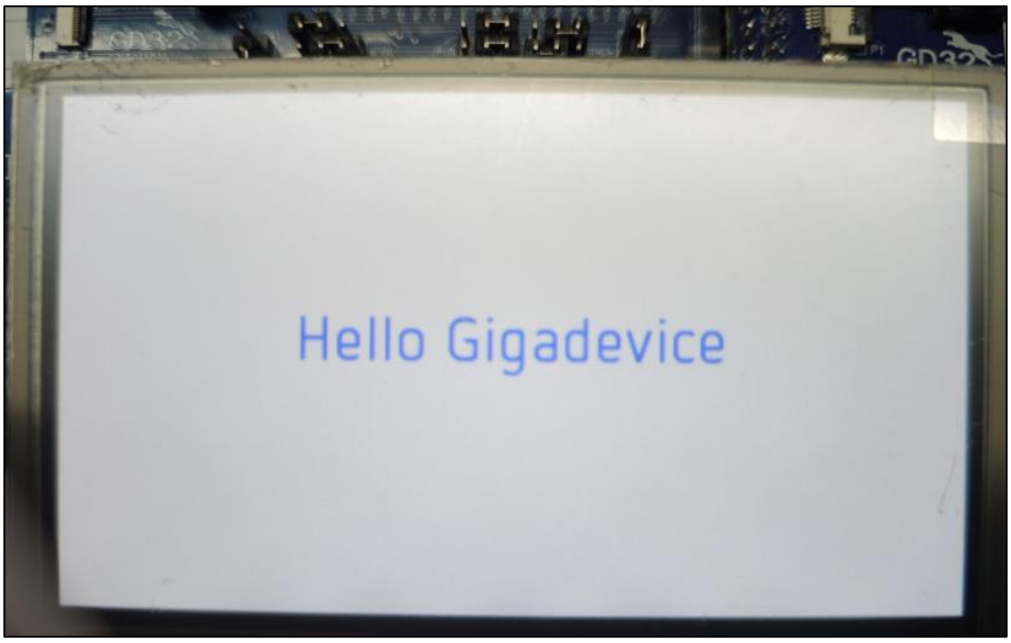
在 AppWizard 中，点击 ⊙ 运行效果如*图 4-1. AppWizard 模拟器效果*所示。

图 4-1. AppWizard 模拟器效果



将 AppWizard 工程里的输出导出，编译之后，烧录程序到开发板上，具体效果如*图 4-2. 开发板演示效果*所示，后续开发可以先在 AppWizard 上进行页面设计，然后输出中间文件，最后在 IDE 中编译成功后烧录到芯片中。

图 4-2. 开发板演示效果



5. 版本历史

表 5-1. 版本历史

版本号	说明	日期
1.0	首次发布	2024 年 5 月 28 日
1.1	新增*表 1-1. 适用产品*	2026 年 2 月 12 日

Important Notice