使用 EBI 驱动同步信号 LCD 屏幕

NuMicro® 32 位系列微控制器范例程序代码介绍

文件信息

应用简述	本范例程序代码使用 EBI 驱动同步信号 LCD 屏幕
BSP 版本	M55M1_Series_BSP_CMSIS_V3.01.001
开发平台	NuMaker-M55M1 V1.0

The information described in this document is the exclusive intellectual property of
Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

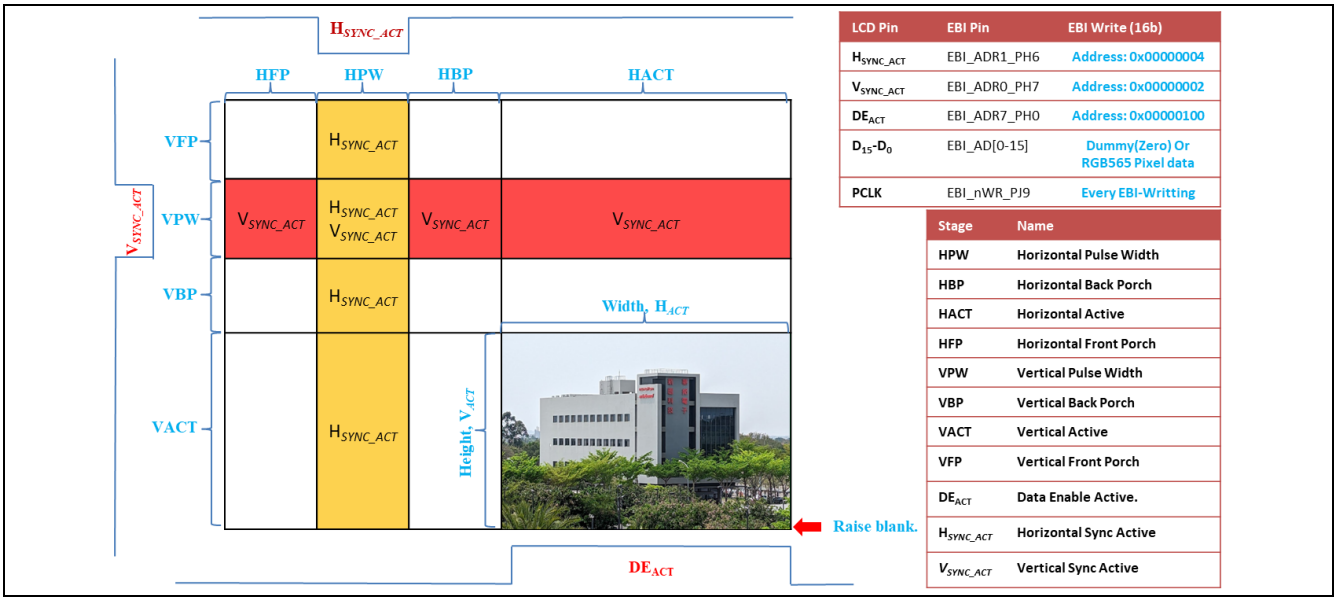
www.nuvoton.com

1. 概述

一般常用于微控制器的 LCD 屏幕接口为 MPU-type LCD，由于这类屏幕内建控制器与显示内存，以致于价格高于同步信号(Sync-type) LCD 屏幕 (可参阅表 1-1)。本范例程序代码使用 M55M1 系列支持的 EBI 传输接口，并搭配 PDMA 或 GDMA 控制器，实现自我刷新画面功能，以驱动同步信号 LCD 屏幕，如图 1-1 所示。

	MPU-type LCD	Sync-type LCD
传输方式	Host 透过传输接口将影像数据同步写入 LCD 显示内存，由 LCD 控制器负责直接输出 RGB 数据与同步信号至面板。	Host 端直接输出 RGB 与同步信号至屏幕，用于影像显示。
内存与带宽使用量	Host 可将少量图像数据分批同步至 LCD 显示内存，由 LCD 面板周期性地读取该内存并输出影像，达成画面显示。此架构可有效降低 Host 的内存与带宽使用量。	Host 需将完整的图像数据储存在自身内存中，并由内建的显示控制器读取这些资料后输出至 LCD 屏幕。控制器需以较高带宽周期性地存取内存以维持影像输出，因此 Host 的内存与带宽使用量相对较高。
传输接口	LCD 模块可透过 EBI i80（并行接口）或 SPI（串行接口）与 Host 连接。EBI 提供较高的数据传输速率，适合高分辨率或高更新率应用；而 SPI 则以低脚位数与简单连接为优势，适用于资源受限的系统设计。	24-bit RGB 接口（RGB888）采用 8-bit 的红、绿、蓝三信道数据进行显示输出，并搭配同步信号（如 HSYNC、VSYNC、DE 及 PCLK）实现实时画面驱动。此接口可提供高画质与低延迟的影像显示，但需占用较多的 I/O 脚位与传输带宽。
硬件成本	具备内建控制器与显示内存的屏幕可减轻 Host 侧的负担，然而由于额外硬件设计，其模块成本相对较高。	此类屏幕未内建显示控制器与显示内存，须由 Host 端负责影像输出处理，但由于省略了相关硬件设计，模块成本相对较低。

表 1-1 MPU-type 与 Sync-type LCD 屏幕比较表



LCD Pin	EBI Pin	EBI Write (16b)
H_{SYNC_ACT}	EBI_ADR1_PH6	Address: 0x00000004
V_{SYNC_ACT}	EBI_ADR0_PH7	Address: 0x00000002
DE_{ACT}	EBI_ADR7_PH0	Address: 0x00000100
D_{15}-D_0	EBI_AD[0-15]	Dummy(Zero) Or RGB565 Pixel data
PCLK	EBI_nWR_PJ9	Every EBI-Writting

Stage	Name
HPW	Horizontal Pulse Width
HBP	Horizontal Back Porch
HACT	Horizontal Active
HFP	Horizontal Front Porch
VPW	Vertical Pulse Width
VBP	Vertical Back Porch
VACT	Vertical Active
VFP	Vertical Front Porch
DE_{ACT}	Data Enable Active.
H_{SYNC_ACT}	Horizontal Sync Active
V_{SYNC_ACT}	Vertical Sync Active

图 1-1 EBI 驱动同步信号 LCD 屏幕

1.1 原理

本范例程序代码实现同步信号 LCD 控制器，透过 EBI 传输接口与 PDMA 或是 GDMA 控制器进行 Memory-to-memory 传输，将图片像素数据传送至同步信号 LCD 屏幕，其运作原理为屏幕画面建立 V Lines × H Stages DMA 描述符，其中：

- V 代表 LCD 时序的 Vertical Pulse Width (VPW)、Vertical Back Porch (VBP)、Vertical Active (VA) 和 Vertical Front Porch (VFP) 的扫描线数量。
- H 代表 Horizontal Pulse Width (HPW)、Horizontal Back Porch (HBP)、Horizontal Active (HA)和 Horizontal Front Porch (HFP)的这四个阶段。

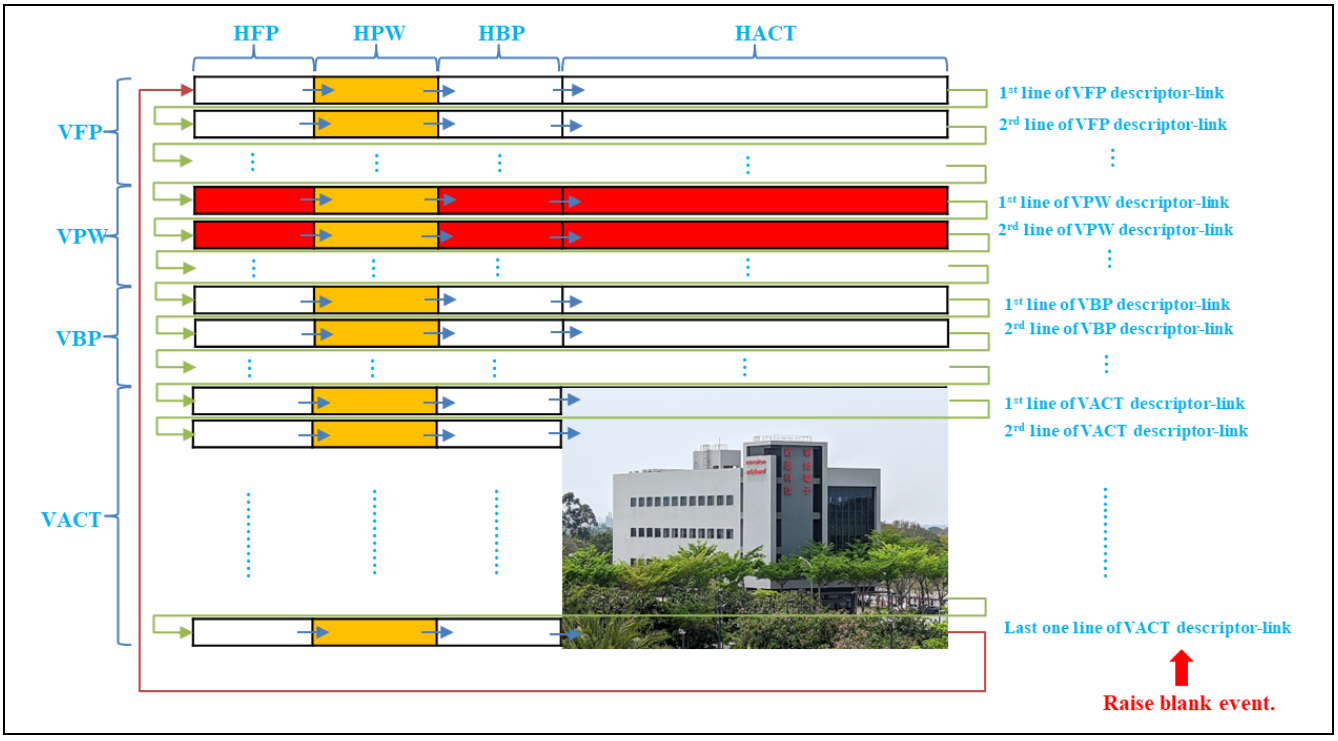


图 1-2 V Lines × H Stages DMA 描述符

如图 1-2 所示，将所有 DMA 描述符依序链接，并把最后一个描述符的 NEXT 参数设置为第一个描述符的地址，从而形成一个环状描述符集，并且将最后一个描述符启用传输完成中断响应，如表 1-2 所示。驱动程序码在中断服务程序（ISR）中更新 VACT.HACT 阶段的描述符数据传输起始地址，切换至新图像的每条图像内存起始地址，实现画面动态切换功能，以防止画面撕裂（Anti-tearing）。

V \ H	HFP	HPW	HBP	HACT
VFP	Set H_{SYNC_ACT} to Inactive Set V_{SYNC_ACT} to Inactive Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy	Set H_{SYNC_ACT} to Active Set V_{SYNC_ACT} to Inactive Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy	Set H_{SYNC_ACT} to Inactive Set V_{SYNC_ACT} to Inactive Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy	Set H_{SYNC_ACT} to Inactive Set V_{SYNC_ACT} to Inactive Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy
VPW	Set H_{SYNC_ACT} to Inactive Set V_{SYNC_ACT} to Active Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy	Set H_{SYNC_ACT} to Active Set V_{SYNC_ACT} to Active Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy	Set H_{SYNC_ACT} to Inactive Set V_{SYNC_ACT} to Active Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy	Set H_{SYNC_ACT} to Inactive Set V_{SYNC_ACT} to Active Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy
VBP	Set H_{SYNC_ACT} to Inactive Set V_{SYNC_ACT} to Inactive Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy	Set H_{SYNC_ACT} to Active Set V_{SYNC_ACT} to Inactive Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy	Set H_{SYNC_ACT} to Inactive Set V_{SYNC_ACT} to Inactive Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy	Set H_{SYNC_ACT} to Inactive Set V_{SYNC_ACT} to Inactive Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy
VACT	Set H_{SYNC_ACT} to Inactive Set V_{SYNC_ACT} to Inactive Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy	Set H_{SYNC_ACT} to Active Set V_{SYNC_ACT} to Inactive Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy	Set H_{SYNC_ACT} to Inactive Set V_{SYNC_ACT} to Inactive Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy	Set H_{SYNC_ACT} to Inactive Set V_{SYNC_ACT} to Inactive Set DE_{ACT} to Active Set $D_{[15-0]}$ to pixel data in every Line Raising blank event.

表 1-2 各 LCD 时序阶段与 HSYNC_ACT、VSYNC_ACT 和 DEACT 脚位对应表

由于不同类型屏幕对于同步信号支持程度不同，有些屏幕可支持 DE_{ACT}-ONLY 同步模式，表示它们仅需要 DE_{ACT} 信号输入；而无需 H_{SYNC_ACT} 和 V_{SYNC_ACT} 信号输入。在操作 DE_{ACT}-ONLY 屏幕时，可大幅度地减少 DMA 描述符的使用量，以节省系统内存空间与使用带宽。如表 1-2 与表 1-3 表 1-3 所列。

V \ H	HFP	HPW	HBP	HACT
VFP	Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy			
VPW				
VBP				
VACT	Set DE_{ACT} to Inactive Set $D_{[15-0]}$ to Dummy			Set DE_{ACT} to Active Set $D_{[15-0]}$ to pixel data in every Line Raising blank event.

表 1-3 各 LCD 时序阶段与 DEACT-Only 脚位对应表

如图 1-3 所示，LCD 时序的 H_{SYNC_ACT}、V_{SYNC_ACT} 和 DE_{ACT} 三个同步信号脚位分别透过 EBI 的 ADR1、ADR0 和 ADR7 地址脚位来仿真，透过指定不同地址译码以产生 LCD 同步信号、PCLK 信号连接至 EBI-nWR 脚位和 EBI 的 16 位数据脚位 $D_{[15-0]}$ 对应到 LCD 颜色信号：

- $R_{[7-3]}$(红色)，$G_{[7-2]}$(绿色) 和 $B_{[7-3]}$(篮色)。
- 额外的 $R_{[2-0]}$、$G_{[1-0]}$ 和 $B_{[2-0]}$ 可以使用 $R_{[7-5]}$、$G_{[7-6]}$ 和 $B_{[7-5]}$ 的数据进行 RGB888 补色，以呈现更高的颜色精度。

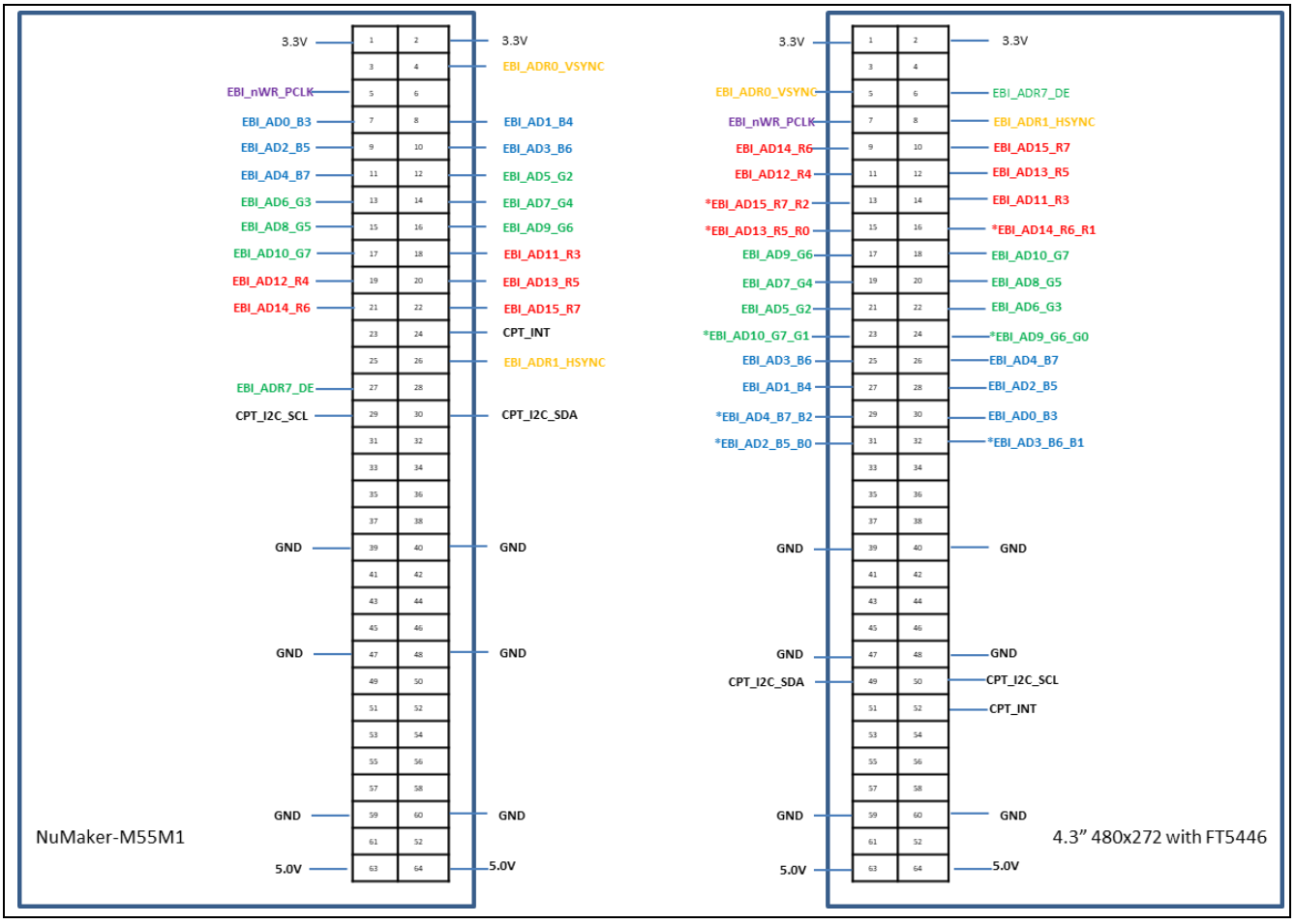


图 1-3 EBI16-to-RGB24 各信号脚位连接图

1.2 执行结果

本范例程序代码提供两种目标编译选项：WQVGA_GDMA 和 WQVGA_PDMA，分别对应使用 GDMA 或 PDMA 控制器进行自我刷新画面功能，如图 1-4 所示。执行时，可透过连接 Nu-Link2-Me 的 VCOM 功能，搭配计算机端终端机软件进行观测，默认通信参数为 115200N81。如图 1-5 与图 1-6 所示，呈现终端机执行结果和图像于屏幕呈现图片效果。

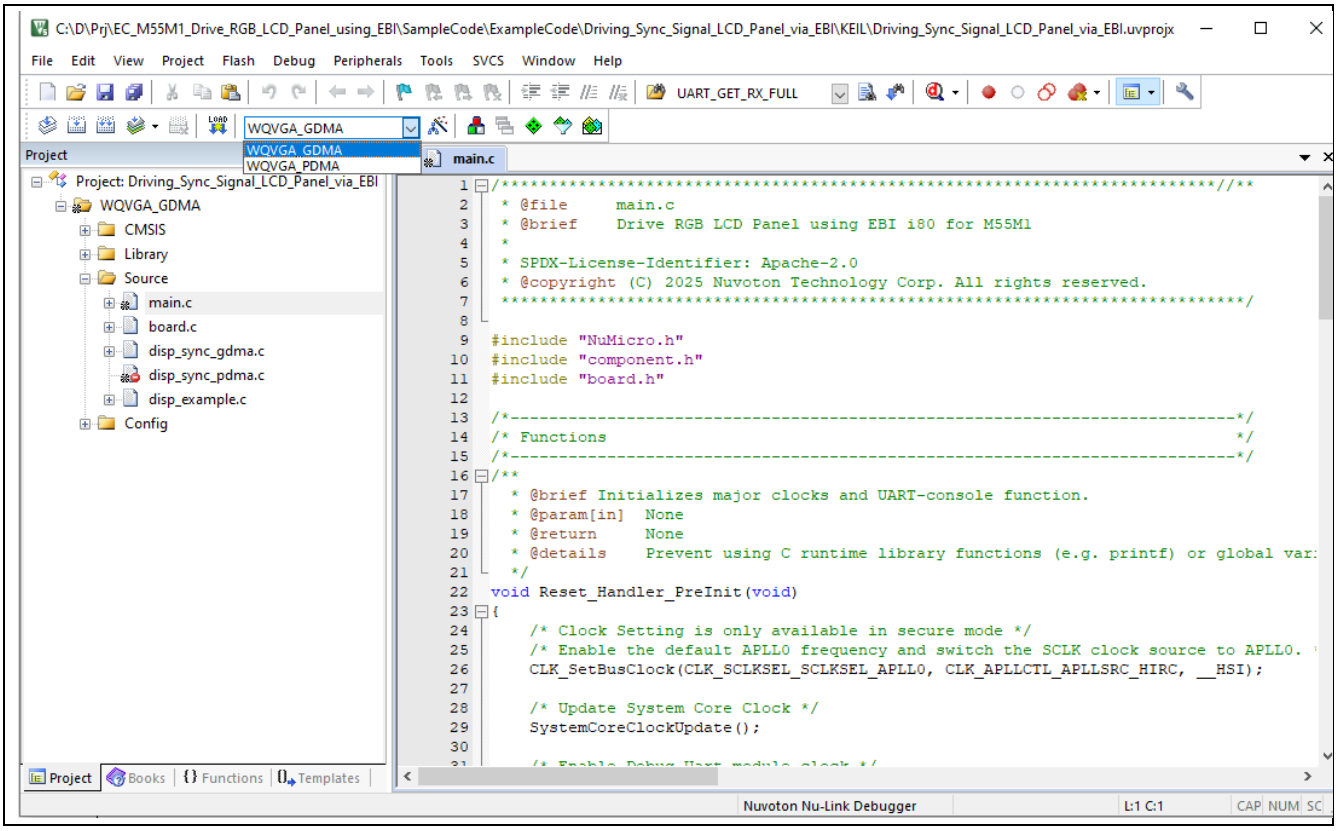


图 1-4 选择编译选项与执行编译

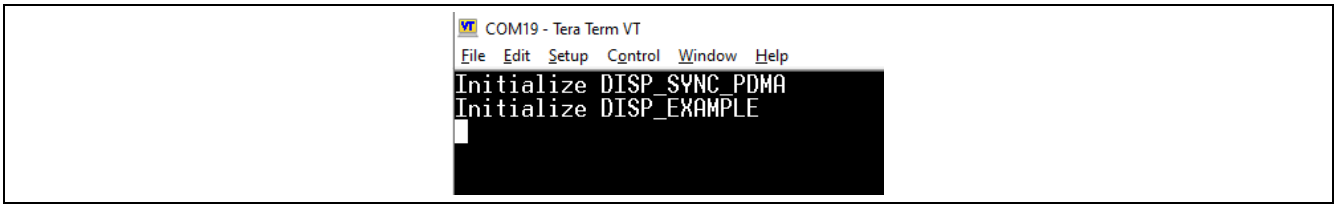


图 1-5 执行画面

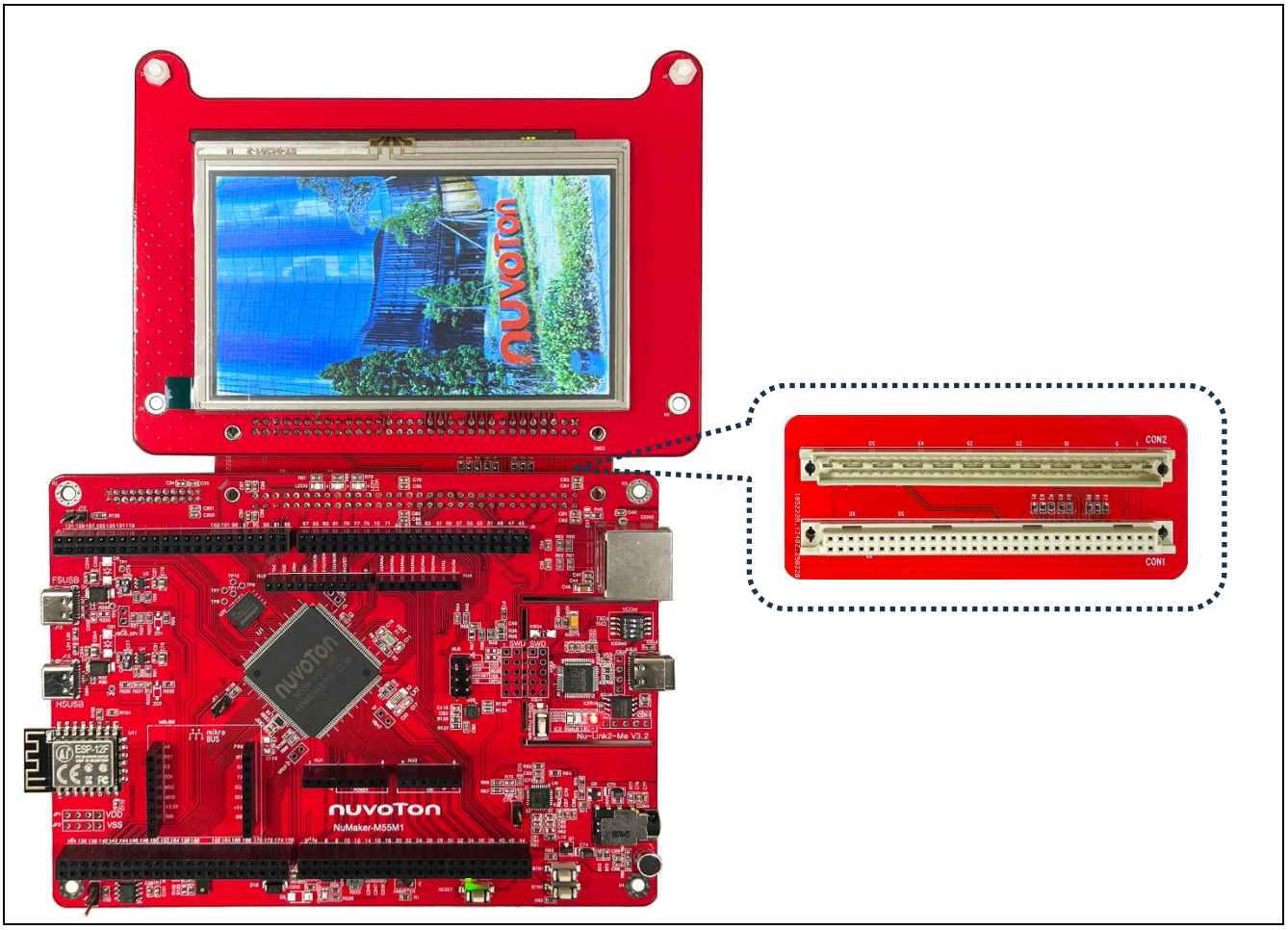


图 1-6 屏幕呈现图像

2. 程序代码介绍

本范例程序代码执行各组件初始化与结束流程，程序代码位于 main.c。各组件初始化、结束函数程序代码和配置文件位于 disp_sync_gdma.c、disp_sync_pdma.c、disp_example.c 和 disp.h 实作内。各组件实作使用 COMPONENT_EXPORT 宏注册组件名称(.name)、组件初始化回调函数(.initialize) 和组件结束回调函数(.finalize)，并将这些组件注册数据结收录至 CompInitTab 区域，供主程序调用执行。

```
/**
  * @brief Initializes the components.
  * @param[in]  None
  * @return     None
  */
static void components_initialize(void)
{
    int i;
    component_export_t asCompInitTbl = (component_export_t)&CompInitTab$$Base;
    uint32_t u32CompInitNum = (component_export_t)&CompInitTab$$Limit - asCompInitTbl;

    for (i = 0; i < u32CompInitNum; i++)
    {
        if (asCompInitTbl[i].initialize)
        {
            printf("Initial %s\n", asCompInitTbl[i].name);
            if ( asCompInitTbl[i].initialize() < 0 )
            {
                // Print message if return of initialize function is small than zero.
                printf("Initialize %s failure.\n", asCompInitTbl[i].name);
            }
        }
    }
}

/**
  * @brief      Finalizes the components.
  * @param[in]  None
  * @return     None
  */
static void components_finalize(void)
{
    int i;
    component_export_t asCompInitTbl = (component_export_t)&CompInitTab$$Base;
    uint32_t u32CompInitNum = (component_export_t)&CompInitTab$$Limit - asCompInitTbl;

    for (i = 0; i < u32CompInitNum; i++)
    {
        if (asCompInitTbl[i].finalize)
        {
            printf("Finalize %s\n", asCompInitTbl[i].name);
            if (asCompInitTbl[i].finalize() < 0)
            {
                // Print message if return of finalize function is small than zero.
                printf("Initialize %s failure.\n", asCompInitTbl[i].name);
```

```
            }
        }
    }
}

int main(void)
{
    // Module clocks and function pin setting initialization.
    board_init();

    /* Initialize all components */
    components_initialize();

    /* Just keep here, or put your code here. */
    while (1)
    {
        __WFI();
    }

    /* This will never execute due to the infinite loop. */
    /* Just keep for orthogonal implementation. */
    components_finalize();

    // Module clocks and function pin setting finalization.
    board_fini();

    return 0;
}
```

2.1 disp_sync_gdma.c

在 disp_sync_gdma.c 的实作中，在 disp_gdma_dsc_init 函数逐一地建立 V Lines × H Stages DMA 描述符，依不同 H/VLINE Stage 给予不同的 Memory-to-memory 操作属性和写出地址，输出 LCD 屏幕 HSYNC_ACT、VSYNC_ACT、DEACT 信号与 VRAM 缓冲区内的像素。这些描述符会储存在一块不可快取的内存区域，程序代码变量名称为 s_sDecLCD，属性宣告为 NVT_NONCACHEABLE，意味着当 CPU 向这些内存地址写入数据时，数据不会进入 CPU 的数据快取空间，而是直接写入内存。在中断服务函数内处理 Blank 事件时，程序会更新所有 VACT.HACT 区 DMA 描述符的起始地址参数，从而切换不同的 VRAM 缓冲区。这样的做法避免了重复操作 SCB_CleanDCache_by_Addr 函数来刷新起始地址参数，从而提高切换不同 VRAM 缓冲区的执行效率。

```
#if defined(NVT_NONCACHEABLE)
    /* V × H DMA Descriptors, stored in a non-cacheable memory region. */
    NVT_NONCACHEABLE static S_DSC_LCD s_sDscLCD;
#else
    static S_DSC_LCD s_sDscLCD;
#endif
// Function to initialize EBI sync GDMA
static int disp_ebi_sync_gdma_init(void)
{
    enum dma350_lib_error_t lib_err;
```

```
    /* Set the VRAM address by default. */
    s_pu16BufAddr = (uint16_t*)g_au8FrameBuf;

    /* Enable GDMA module clock and un-mask interrupt. */
    gdma_init();

    /* Initial all Lines descriptor-link. */
    disp_gdma_dsc_init();

    /* Link to external command */
    dma350_ch_enable_linkaddr(GDMA_CH_DEV_S[1]);
    dma350_ch_set_linkaddr32(GDMA_CH_DEV_S[1], (uint32_t) s_head);
    dma350_ch_disable_intr(GDMA_CH_DEV_S[1], DMA350_CH_INTREN_DONE);
    dma350_ch_cmd(GDMA_CH_DEV_S[1], DMA350_CH_CMD_ENABLECMD);

    return 0;
}

static int disp_ebi_sync_gdma_fini(void)
{
    /* Disable GDMA module clock and mask interrupt. */
    gdma_fini();

    return 0;
}
```

2.2 disp_sync_pdma.c

在 disp_sync_pdma.c 的实作中，在 disp_pdma_dsc_init 函数逐一地建立 V Lines × H Stages DMA 描述符，依不同 H/VLINE Stage 给予不同的 Memory-to-memory 操作属性和写出地址，输出 LCD 屏幕 HSYNC_ACT、VSYNC_ACT、DEACT 信号与 VRAM 缓冲区内的像素。这些描述符会储存在一块不可快取的内存区域，程序代码变量名称为 s_sDecLCD，属性宣告为 NVT_NONCACHEABLE，意味着当 CPU 向这些内存地址写入数据时，数据不会进入 CPU 的数据快取空间，而是直接写入内存。在中断服务函数内处理 Blank 事件时，程序会更新所有 VACT.HACT 区 DMA 描述符的起始地址参数，从而切换不同的 VRAM 缓冲区。这样的做法避免了重复操作 SCB_CleanDCache_by_Addr 函数来刷新起始地址参数，从而提高切换不同 VRAM 缓冲区的执行效率。

```
#if defined(NVT_NONCACHEABLE)
    /* V × H DMA Descriptors, stored in a non-cacheable memory region. */
    NVT_NONCACHEABLE static S_DSC_LCD s_sDscLCD;
#else
    static S_DSC_LCD s_sDscLCD;
#endif
// Function to initialize the EBI sync PDMA
static int disp_sync_pdma_init(void)
{
    struct nu_pdma_chn_cb sChnCB;

    /* Set the VRAM address by default. */
    s_pu16BufAddr = (uint16_t*)g_au8FrameBuf;
```

```
    pdma_init();

    if (s_i32Channel < 0)
    {
        /* Allocate a PDMA channel resource. */
        s_i32Channel = nu_pdma_channel_allocate(PDMA_MEM);
        if (s_i32Channel < 0)
            return -1;
    }

    /* Initial all Lines descriptor-link. */
    disp_pdma_dsc_init();

    /* Register ISR callback function */
    sChnCB.m_eCBType = eCBType_Event;
    sChnCB.m_pfnCBHandler = nu_pdma_memfun_cb;
    sChnCB.m_pvUserData = (void *)NULL;

    nu_pdma_filtering_set(s_i32Channel, NU_PDMA_EVENT_TRANSFER_DONE);
    nu_pdma_callback_register(s_i32Channel, &sChnCB);

    /* Trigger scatter-gather transferring. */
    return nu_pdma_sg_transfer(s_i32Channel, s_head, 0);
}

// Function to deinitialize the EBI sync PDMA
static int disp_sync_pdma_fini(void)
{
    if (s_i32Channel >= 0)
    {
        /* Free allocated PDMA channel resource. */
        nu_pdma_channel_free(s_i32Channel);
        s_i32Channel = -1;
    }

    pdma_fini();

    return 0;
}
```

2.3 disp_example.c

在 disp_example.c 的实作中，使用 .incbin 汇编语言指令将两张 RGB565 格式的影像嵌入程序中，直接存入 FLASH，以缩短编译时间。程序代码中，PATH_IMAGE1_BIN 与 PATH_IMAGE2_BIN 分别为两张图片档案路径定义值，存于 disp.h 头文件中。图像文件案可透过在线转换工具 Image to RGB565 Converter (https://longfangsong.github.io/en/image-to-rgb565/) 进行汇入全屏分辨率的 PNG 图文件并下载，取得 RGB565 格式档案。在 disp_example_init 函数中，首先注册 Blank 事件的回调函数 (disp_example_blankcb)。最后，将两张 RGB565 影像数据拷贝至影像缓冲区。由于 Cortex-M55 支持 Data-Cache，拷贝完成后需调用 SCB_CleanDCache_by_Addr 函数，以同步影像数据至 SRAM。在

disp_example_blankcb 中，透过 disp_set_vrambufaddr 函数并传入不同的 VRAM 地址，交替更新画面，以防止画面撕裂。然而，由于展示范例更新速率过快导致显示重影，因此设定条件为收到 16 次 Blank 事件后再进行画面切换。

```
#define STR2(x) #x
#define STR(x) STR2(x)
#define INCBIN(name, file) \
    __asm__(".section .rodata\n" \
            ".global incbin_" STR(name) "_start\n" \
            ".balign 16\n" \
            "incbin_" STR(name) "_start:\n" \
            ".incbin \"" file "\"\n" \
            \
            ".global incbin_" STR(name) "_end\n" \
            ".balign 1\n" \
            "incbin_" STR(name) "_end:\n" \
            ".byte 0\n" \
    ); \
    extern const __attribute__((aligned(32))) void* incbin_ ## name ## _start; \
    extern const void* incbin_ ## name ## _end; \

static uint8_t s_au8FrameBuf[CONFIG_VRAM_TOTAL_ALLOCATED_SIZE]
__attribute__((aligned(DCACHE_LINE_SIZE))); // Declare VRAM instance.
INCBIN(image1, PATH_IMAGE1_BIN);
INCBIN(image2, PATH_IMAGE2_BIN);

// Blank event callback function
void disp_example_blankcb(void *p)
{
    static uint32_t u32Counter = 0;

    /* Toggle different image showing after getting 16 event, */
    /* Just for avoid visual persistence ghosting. */
    #define DEF_TOGGLE_COND     (u32Counter & 0x10u)

    /* Toggle between image1 and image2 display based on u32Counter's value. */
    if (DEF_TOGGLE_COND)
    {
        /* If the condition is true, set VRAM buffer to image2 buffer address. */
        disp_set_vrambufaddr((void *)&s_au8FrameBuf[CONFIG_VRAM_BUF_SIZE]);
    }
    else
    {
        /* If the condition is false, set VRAM buffer to image1 buffer address. */
        disp_set_vrambufaddr((void *)s_au8FrameBuf);
    }

    // Increment the counter to alternate the display in the next callback
    u32Counter++;
}

// Initialize the display example
static int disp_example_init(void)
{
    /* Set VRAM buffer address. */
```

```
    disp_set_vrambufaddr((void *)s_au8FrameBuf);

    /* Set blank event callback function. */
    disp_set_blankcb(disp_example_blankcb);

    /* Copy image1 and image2 pixel data to VRAM buffer. */
    memcpy(s_au8FrameBuf, (const uint8_t *)&incbin_image1_start, CONFIG_VRAM_BUF_SIZE);
    memcpy(&s_au8FrameBuf[CONFIG_VRAM_BUF_SIZE], (const uint8_t *)&incbin_image2_start,
CONFIG_VRAM_BUF_SIZE);

    /* Flush all pixel data in DCache to memory. */
    SCB_CleanDCache_by_Addr(s_au8FrameBuf, 2 * CONFIG_VRAM_BUF_SIZE);

    return 0;
}
```

2.4 disp.h

disp.h 头文件用于定义同步信号 LCD 屏幕的同步信号脚位与时间驱动参数。这些参数需依据实际板子的引脚配置进行设置，例如：

- CONFIG_LCD_PANEL_USE_DE_ONLY 取消定义后，系统将启用 VSYNC_ACT、HSYNC_ACT 和 DEACT 三条信号输出至屏幕；反之，仅输出 DEACT 信号。
- CONFIG_DISP_*_ACTIVE_LOW 设为 0 时，表示对应信号为 Active-High 动作；反之，则为 Active-Low 动作。
- CONFIG_DISP_*_BITIDX 设定值为各 EBI_ADRx 脚位编号加 1 的值。
- CONFIG_TIMING_* 用于设定同步信号屏幕的时序参数，需与屏幕规格书对应调整。

```
#define CONFIG_LCD_PANEL_USE_DE_ONLY          /*!< DE-only mode, W/O H/VSync. */

#define CONFIG_DISP_DE_ACTIVE_LOW         0   /*!< Disable DE active low */
#define CONFIG_DISP_VPW_ACTIVE_LOW        1   /*!< Enable VPW active low */
#define CONFIG_DISP_HPW_ACTIVE_LOW        1   /*!< Enable HPW active low */

#define CONFIG_DISP_DE_BITIDX             8   /*!< Implies SET_EBI_ADR7_PH0 */
#define CONFIG_DISP_VSYNC_BITIDX          1   /*!< Implies SET_EBI_ADR0_PH7 */
#define CONFIG_DISP_HSYNC_BITIDX          2   /*!< Implies SET_EBI_ADR1_PH6 */

#define CONFIG_TIMING_HACT              480   /*!< Specify Width */
#define CONFIG_TIMING_VACT              272   /*!< Specify Height */
#define CONFIG_TIMING_HBP                30   /*!< Specify HBP (Horizontal Back Porch) */
#define CONFIG_TIMING_HFP                 5   /*!< Specify HFP (Horizontal Front Porch) */
#define CONFIG_TIMING_HPW                41   /*!< Specify HPW (HSYNC Pulse Width) */
#define CONFIG_TIMING_VBP                 2   /*!< Specify VBP (Vertical Back Porch) */
#define CONFIG_TIMING_VFP                27   /*!< Specify VFP (Vertical Front Porch) */
#define CONFIG_TIMING_VPW                10   /*!< Specify VPW (VSYNC width) */
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M55M1_Series_BSP_CMSIS_V3.01.001。
- IDE 版本
  - Keil uVision 5.40。

3.2 硬件需求

- 电路组件
  - NuMaker-M55M1 V1.0。
  - EBI16-RGB24 连接板。
  - RGB LCD Panel，分辨率为 WQVGA。
- 线路示意图
  - 将 NuMaker-M55M1 V1.0、EBI16-RGB24 连接板和 RGB LCD Panel 进行组装。
  - 使用 USB Type-C to Type-A 连接线，将 NuMaker-M55M1 V1.0 开发板连接至计算机。

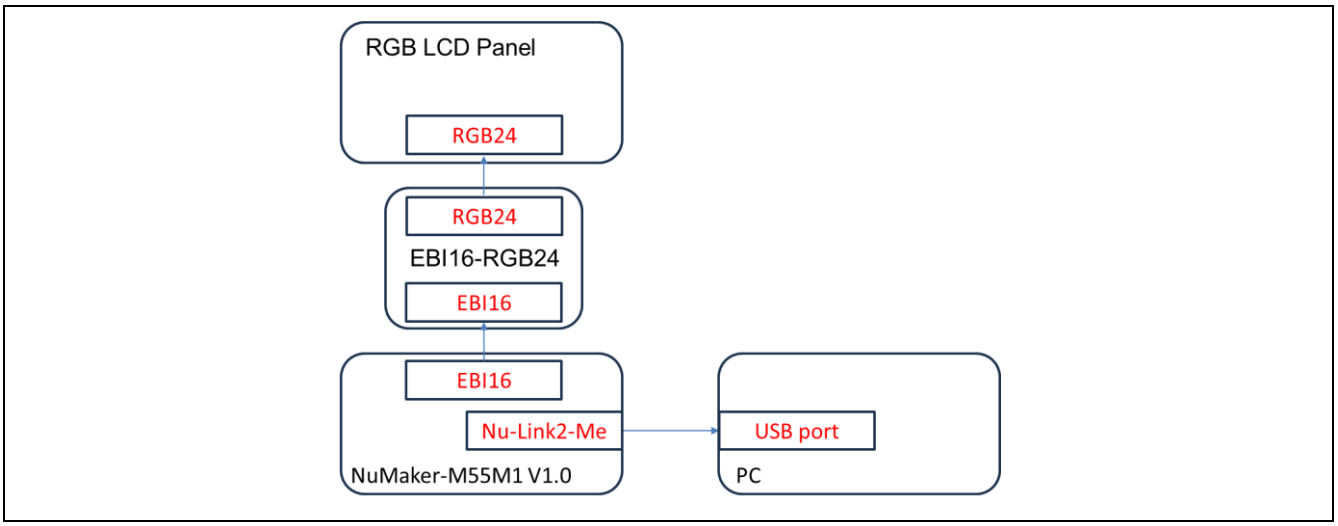


图 3-1 线路示意图

4. 目录信息

目录结构如下图所示。

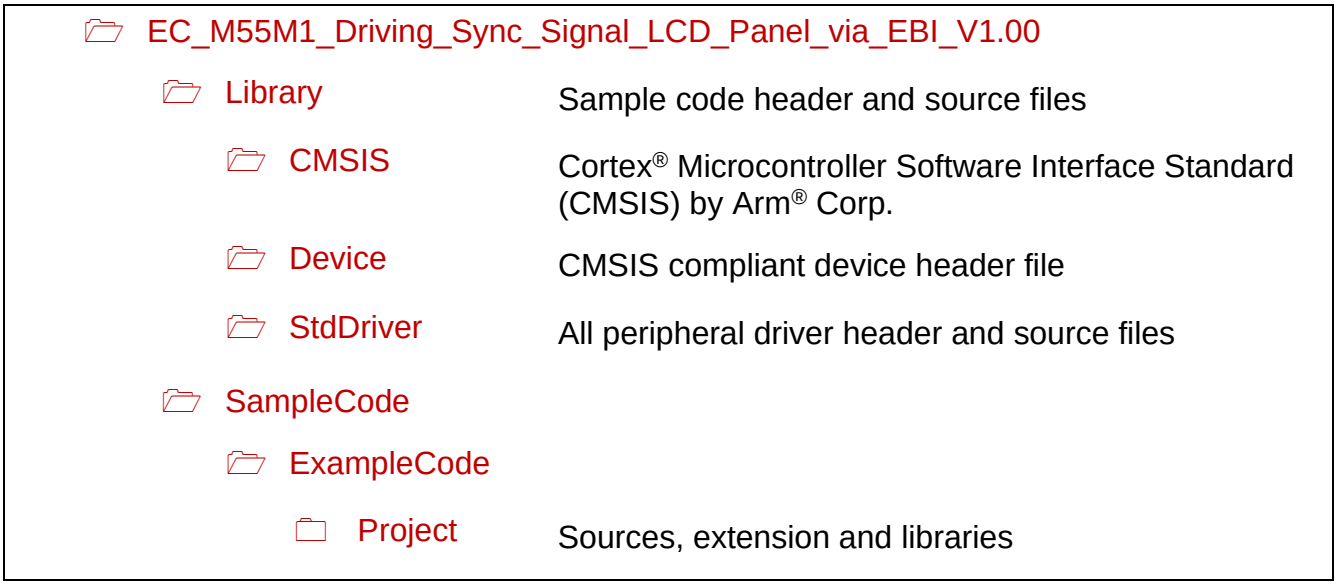


图 4-1 目录信息

5. 范例程序执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *Driving_Sync_Signal_LCD_Panel_via_EBI.uvprojx*。
2. 进入编译模式接口
   - 编译
   - 下载代码至内存
   - 进入 / 离开除错模式
3. 进入除错模式接口
   - 执行代码

6. 修订纪录

Date	Revision	Description
2025.03.21	1.00	初始发布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.