使用 EBI 驅動同步訊號 LCD 屏幕

NuMicro® 32 位元系列微控制器範例程式碼介紹

文件資訊

應用簡述	本範例程式碼使用 EBI 驅動同步訊號 LCD 屏幕
BSP 版本	M55M1_Series_BSP_CMSIS_V3.01.001
開發平台	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.

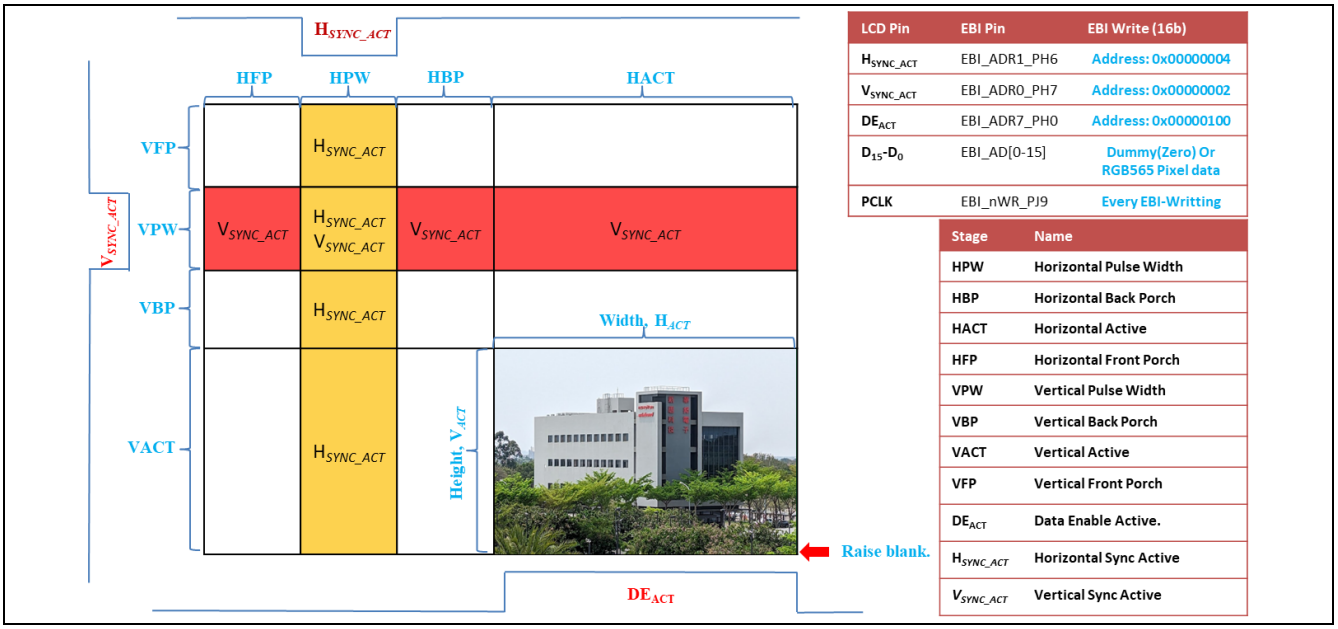
www.nuvoton.com

1. 概述

一般常用於微控制器的 LCD 屏幕介面為 MPU-type LCD，由於這類屏幕內建控制器與顯示記憶體，以致於價格高於同步信號(Sync-Type) LCD 屏幕 (可參閱表 1-1)。本範例程式碼使用 M55M1 系列支援的 EBI 傳輸介面，並搭配 PDMA 或 GDMA 控制器，實現自我刷新畫面功能，以驅動同步信號 LCD 屏幕，如圖 1-1 所示。

	MPU-type LCD	Sync-type LCD
傳輸方式	Host 透過傳輸介面將影像資料同步寫入 LCD 顯示記憶體，由 LCD 控制器負責直接輸出 RGB 資料與同步信號至面板。	Host 端直接輸出 RGB 與同步信號至螢幕，用於影像顯示。
內存與頻寬使用量	Host 可將少量圖像資料分批同步至 LCD 顯示記憶體，由 LCD 面板週期性地讀取該記憶體並輸出影像，達成畫面顯示。此架構可有效降低 Host 的記憶體與頻寬使用量。	Host 需將完整的圖像資料儲存在自身記憶體中，並由內建的顯示控制器讀取這些資料後輸出至 LCD 屏幕。控制器需以較高頻寬週期性地存取記憶體以維持影像輸出，因此 Host 的記憶體與頻寬使用量相對較高。
傳輸介面	LCD 模組可透過 EBI i80（平行介面）或 SPI（序列介面）與 Host 連接。EBI 提供較高的資料傳輸速率，適合高解析度或高更新率應用；而 SPI 則以低腳位數與簡單連接為優勢，適用於資源受限的系統設計。	24-bit RGB 介面（RGB888）採用 8-bit 的紅、綠、藍三通道資料進行顯示輸出，並搭配同步訊號（如 HSYNC、VSYNC、DE 及 PCLK）實現即時畫面驅動。此介面可提供高畫質與低延遲的影像顯示，但需佔用較多的 I/O 腳位與傳輸頻寬。
硬體成本	具備內建控制器與顯示記憶體的屏幕可減輕 Host 側的負擔，然而由於額外硬體設計，其模組成本相對較高。	此類屏幕未內建顯示控制器與顯示記憶體，須由 Host 端負責影像輸出處理，但由於省略了相關硬體設計，模組成本相對較低。

表 1-1 MPU-type 與 Sync-type LCD 屏幕比較表



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

圖 1-1 EBI 驅動同步信號 LCD 屏幕

1.1 原理

本範例程式碼實現同步信號 LCD 控制器，透過 EBI 傳輸介面與 PDMA 或是 GDMA 控制器進行 Memory-to-memory 傳輸，將圖片像素資料傳送至同步信號 LCD 屏幕，其運作原理為屏幕畫面建立 V Lines × H Stages DMA 描述符，其中：

- V 代表 LCD 時序的 Vertical Pulse Width (VPW)、Vertical Back Porch (VBP)、Vertical Active (VA) 和 Vertical Front Porch (VFP) 的掃描線數量。
- H 代表 Horizontal Pulse Width (HPW)、Horizontal Back Porch (HBP)、Horizontal Active (HA)和 Horizontal Front Porch (HFP)的這四個階段。

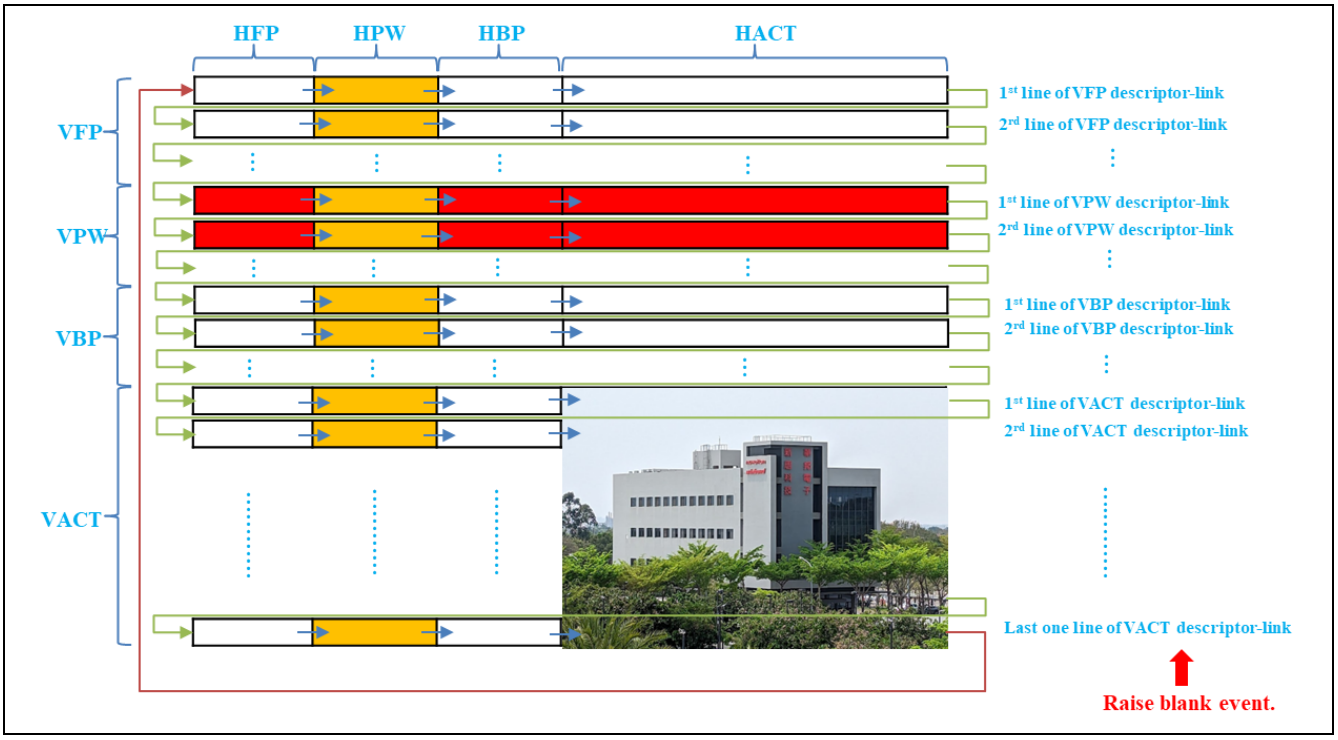


圖 1-2 V Lines × H Stages DMA 描述符

如圖 1-2 所示，將所有 DMA 描述符依序鏈接，並把最後一個描述符的 NEXT 參數設置為第一個描述符的地址，從而形成一個環狀描述符集，並且將最後一個描述符啟用傳輸完成中斷響應，如表 1-2 所示。驅動程式碼在中斷服務程序（ISR）中更新 VACT.HACT 階段的描述符資料傳輸起始地址，切換至新圖像的每條圖像記憶體起始地址，實現畫面動態切換功能，以防止畫面撕裂（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 時序階段與 H_{SYNC_ACT}、V_{SYNC_ACT} 和 DE_{ACT} 腳位對應表

由於不同類型屏幕對於同步信號支援程度不同，有些屏幕可支援 DE_{ACT}-ONLY 同步模式，表示它們僅需要 DE_{ACT} 信號輸入；而無需 H_{SYNC_ACT} 和 V_{SYNC_ACT} 信號輸入。在操作 DE_{ACT}-ONLY 屏幕時，可大幅度地減少 DMA 描述符的使用量，以節省系統記憶體空間與使用頻寬。如表 1-2 與表 1-3 表 1-3 各 LCD 時序階段與 DE_{ACT}-Only 腳位對應表所列。

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 時序階段與 DE_{ACT}-Only 腳位對應表

如圖 1-3 所示，LCD 時序的 H_{SYNC_ACT}、V_{SYNC_ACT} 和 DE_{ACT} 三個同步訊號腳位分別透過 EBI 的 ADR1、ADR0 和 ADR7 位址腳位來模擬，透過指定不同位址解碼以產生 LCD 同步訊號、PCLK 訊號連接至 EBI-nWR 腳位和 EBI 的 16 位元資料腳位 $D_{[15-0]}$ 對應到 LCD 顏色信號：

- $R_{[7-3]}$(紅色)，$G_{[7-2]}$(綠色) 和 $B_{[7-3]}$(藍色)。
- 額外的 $R_{[2-0]}$、$G_{[1-0]}$ 和 $B_{[2-0]}$ 可以使用 $R_{[7-5]}$、$G_{[7-6]}$ 和 $B_{[7-5]}$ 的資料進行 RGB888 補色，以呈現更高的顏色精度。

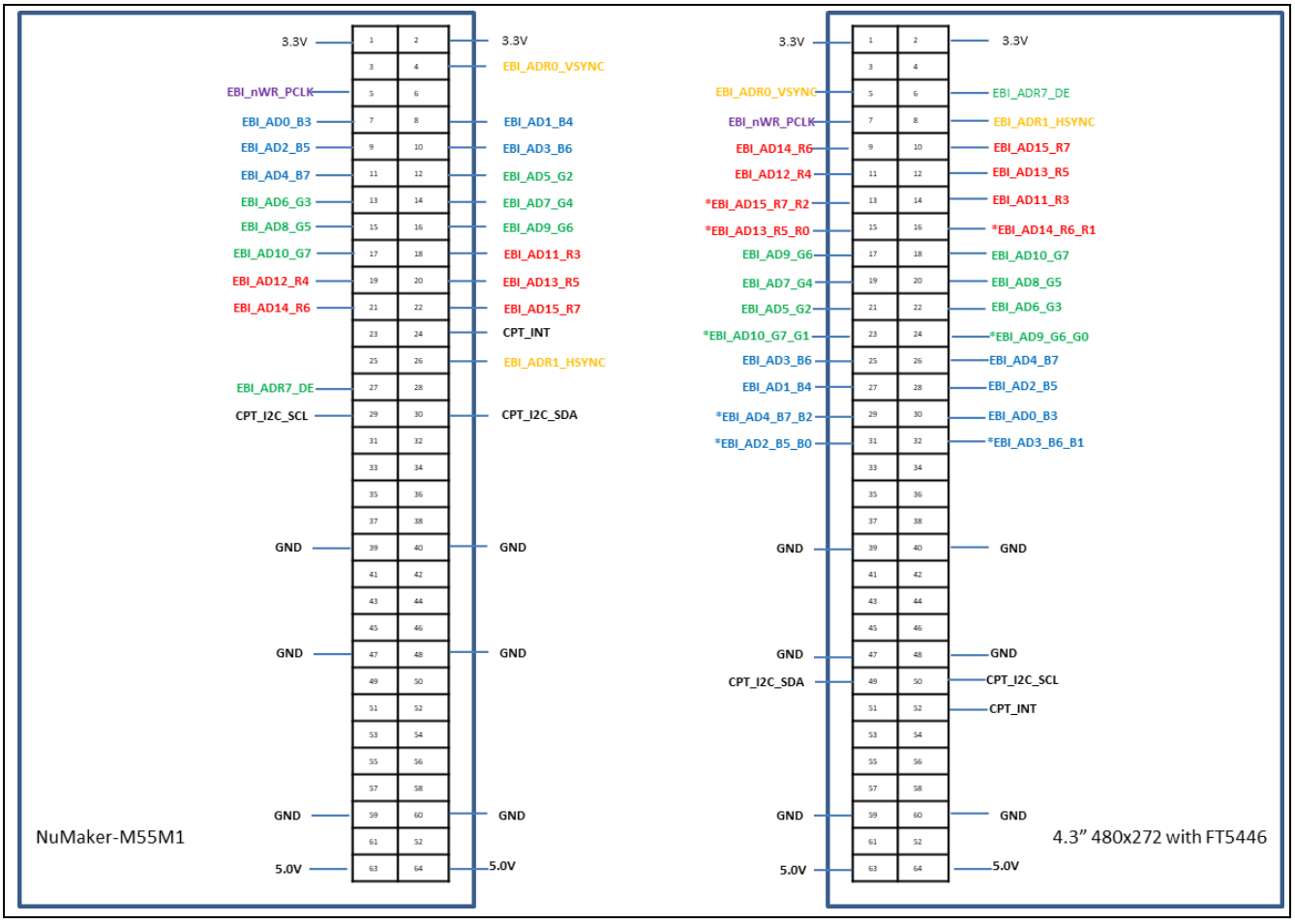


圖 1-3 EBI16-to-RGB24 各訊號腳位連接圖

1.2 執行結果

本範例程式碼提供兩種目標編譯選項：WQVGA_GDMA 和 WQVGA_PDMA，分別對應使用 GDMA 或 PDMA 控制器進行自我刷新畫面功能，如圖 1-4 所示。執行時，可透過連接 Nu-Link2-Me 的 VCOM 功能，搭配電腦端終端機軟體進行觀測，預設通訊參數為 115200N81。如圖 1-5 與圖 1-6 所示，呈現終端機執行結果和圖像於屏幕呈現圖片效果。

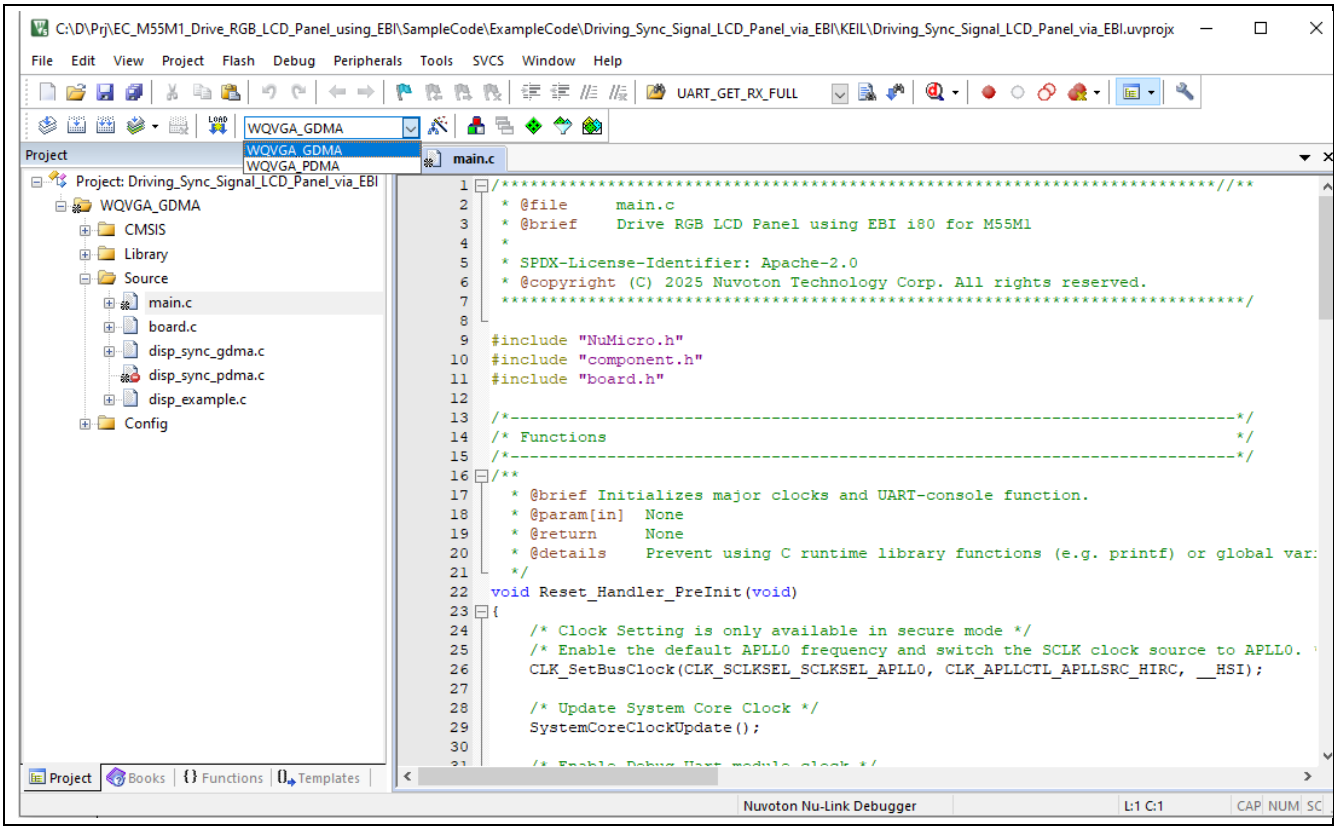


圖 1-4 選擇編譯選項與執行編譯

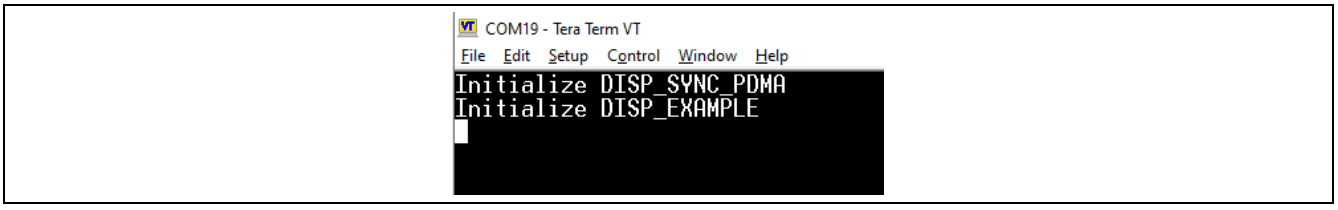


圖 1-5 執行畫面

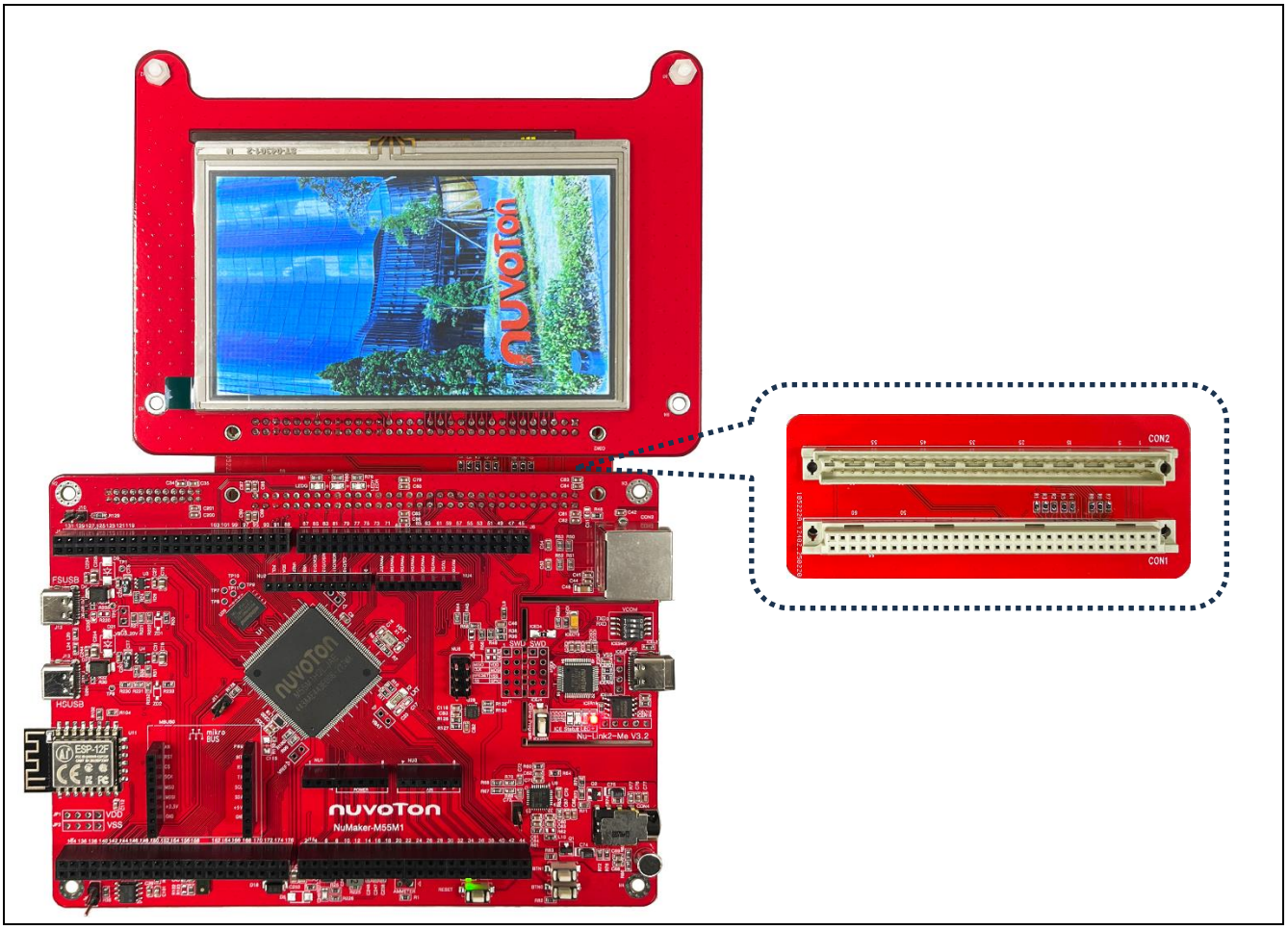

CON2
CON1
nuvoTon
NuMaker-M55M1
Nu-Link2-Me V3.2
ESP-12F
FSUSB
HOUSB

圖 1-6 屏幕呈現圖像

2. 程式碼介紹

本範例程式碼執行各元件初始化與結束流程，程式碼位於 main.c。各組件初始化、結束函式程式碼和設定檔位於 disp_sync_gdma.c、disp_sync_pdma.c、disp_example.c 和 disp.h 實作內。各組件實作使用 COMPONENT_EXPORT 巨集註冊組件名稱(.name)、組件初始化回調函式(.initialize) 和組件結束回調函式(.finalize)，並將這些組件註冊資料結收錄至 CompInitTab 區域，供主程式呼叫執行。

```
/**
  * @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 的實作中，在 disp_gdma_dsc_init 函式逐一地建立 V Lines × H Stages DMA 描述符，依不同 H/VLINE Stage 給予不同的 Memory-to-memory 操作屬性和寫出位址，輸出 LCD 屏幕 H_{SYNC_ACT}、V_{SYNC_ACT}、DE_{ACT} 訊號與 VRAM 緩衝區內的像素。這些描述符會儲存在一塊不可快取的記憶體區域，程式碼變數名稱為 s_sDecLCD，屬性宣告為 NVT_NONCACHEABLE，意味著當 CPU 向這些記憶體位址寫入資料時，資料不會進入 CPU 的資料快取空間，而是直接寫入記憶體。在中斷服務函式內處理 Blank 事件時，程式會更新所有 VACT.HACT 區 DMA 描述符的起始位址參數，從而切換不同的 VRAM 緩衝區。這樣的做法避免了重複操作 SCB_CleanDCache_by_Addr 函式來刷新起始位址參數，從而提高切換不同 VRAM 緩衝區的執行效率。

```
#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 的實作中，在 disp_pdma_dsc_init 函式逐一地建立 V Lines × H Stages DMA 描述符，依不同 H/VLINE Stage 給予不同的 Memory-to-memory 操作屬性和寫出位址，輸出 LCD 屏幕 H_{SYNC_ACT}、V_{SYNC_ACT}、DE_{ACT} 訊號與 VRAM 緩衝區內的像素。這些描述符會儲存在一塊不可快取的記憶體區域，程式碼變數名稱為 s_sDecLCD，屬性宣告為 NVT_NONCACHEABLE，意味著當 CPU 向這些記憶體位址寫入資料時，資料不會進入 CPU 的資料快取空間，而是直接寫入記憶體。在中斷服務函式內處理 Blank 事件時，程式會更新所有 VACT.HACT 區 DMA 描述符的起始位址參數，從而切換不同的 VRAM 緩衝區。這樣的做法避免了重複操作 SCB_CleanDCache_by_Addr 函式來刷新起始位址參數，從而提高切換不同 VRAM 緩衝區的執行效率。

```
#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 的實作中，使用 .incbin 組合語言指令將兩張 RGB565 格式的影像嵌入程式中，直接存入 FLASH，以縮短編譯時間。程式碼中，PATH_IMAGE1_BIN 與 PATH_IMAGE2_BIN 分別為兩張圖片檔案路徑定義值，存於 disp.h 標頭檔中。影像檔案可透過線上轉換工具 Image to RGB565 Converter (https://longfangsong.github.io/en/image-to-rgb565/) 進行匯入全屏解析度的 PNG 圖檔並下載，取得 RGB565 格式檔案。在 disp_example_init 函式中，首先註冊 blank 事件的回調函式 (disp_example_blankcb)。最後，

將兩張 RGB565 影像資料拷貝至影像緩衝區。由於 Cortex-M55 支援 Data-Cache，拷貝完成後需呼叫 SCB_CleanDCache_by_Addr 函式，以同步影像資料至 SRAM。在 disp_example_blankcb 中，透過 disp_set_vrambufaddr 函式並傳入不同的 VRAM 位址，交替更新畫面，以防止畫面撕裂。然而，由於展示範例更新速率過快導致顯示重影，因此設定條件為收到 16 次 Blank 事件後再進行畫面切換。

```
#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 標頭檔用於定義同步信號 LCD 屏幕的同步訊號腳位與時間驅動參數。這些參數需依據實際板子的引腳配置進行設置，例如：

- CONFIG_LCD_PANEL_USE_DE_ONLY 取消定義後，系統將啟用 V_{SYNC_ACT}、H_{SYNC_ACT} 和 DE_{ACT} 三條訊號輸出至屏幕；反之，僅輸出 DE_{ACT} 訊號。
- CONFIG_DISP_*_ACTIVE_LOW 設為 0 時，表示對應訊號為 Active-High 動作；反之，則為 Active-Low 動作。
- CONFIG_DISP_*_BITIDX 設定值為各 EBI_ADRx 腳位編號加 1 的值。
- CONFIG_TIMING_* 用於設定同步訊號屏幕的時序參數，需與屏幕規格書對應調整。

```
#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. 軟體與硬體需求

3.1 軟體需求

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

3.2 硬體需求

- 電路元件
  - NuMaker-M55M1 V1.0。
  - EBI16-RGB24 連接板。
  - RGB LCD Panel，解析度為 WQVGA。
- 線路示意圖
  - 將 NuMaker-M55M1 V1.0、EBI16-RGB24 連接板和 RGB LCD Panel 進行組裝。
  - 使用 USB Type-C to Type-A 連接線，將 NuMaker-M55M1 V1.0 開發板連接至電腦。

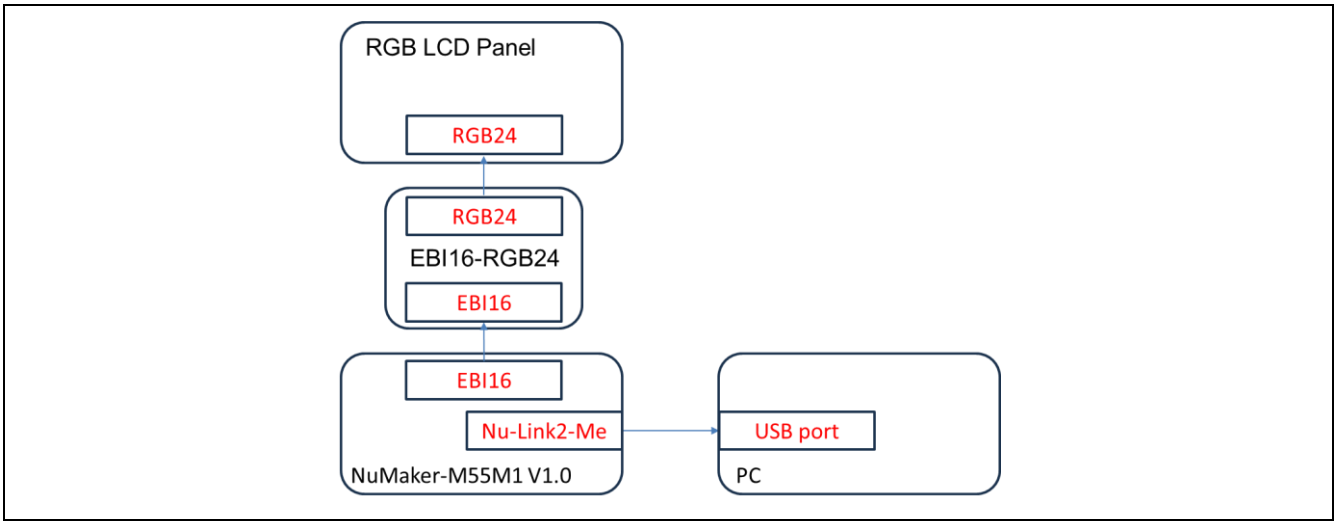


圖 3-1 線路示意圖

4. 目錄資訊

目錄結構如下圖所示。

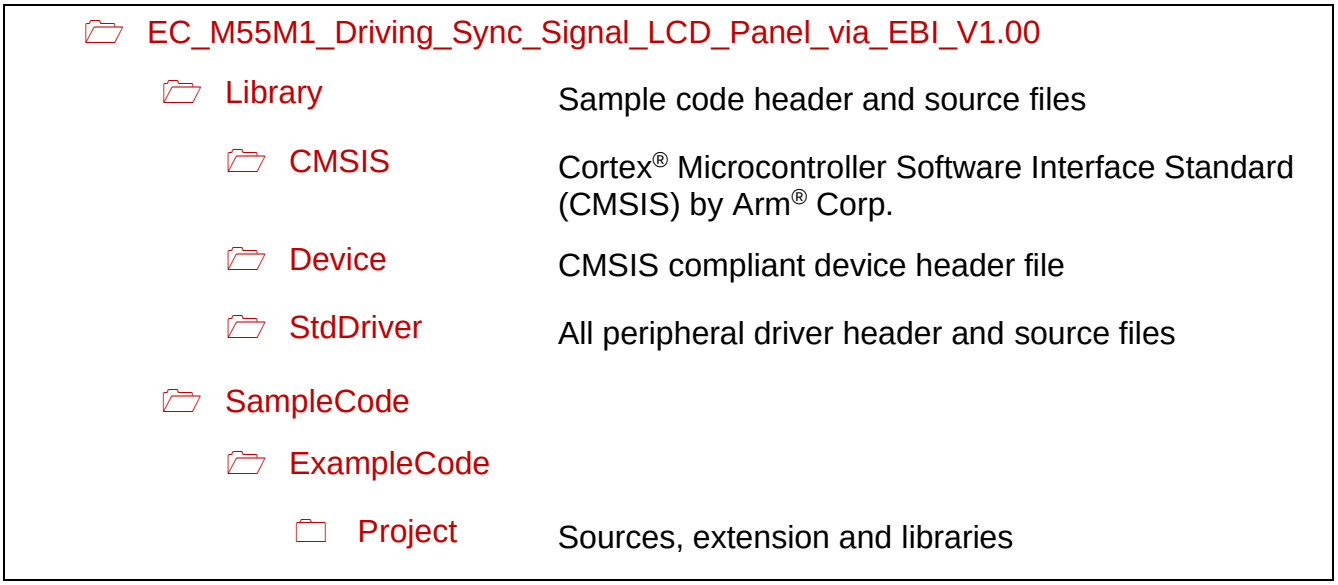


圖 4-1 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 *Driving_Sync_Signal_LCD_Panel_via_EBI.uvprojx*。

1. 進入編譯模式介面
   - 編譯
   - 下載代碼至記憶體
   - 進入 / 離開除錯模式
2. 進入除錯模式介面
   - 執行代碼

6. 修訂紀錄

Date	Revision	Description
2025.03.21	1.00	初始發佈。

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.