Driving Sync Signal LCD Panel via EBI

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses the EBI to drive synchronous signal LCD panel.
BSP Version	M55M1_Series_BSP_CMSIS_V3.01.001
Hardware	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. Overview

The commonly used LCD interface for microcontrollers (MCUs) is the MPU-type LCD. Since the MPU-type LCDs integrate a built-in controller and display memory, they are generally more expensive than Sync-type LCDs (Table 1-1).This example code utilizes the EBI (External Bus Interface) supported by the M55M1 series MCUs, combined with a PDMA or GDMA controller, to implement a self-refreshing display function for driving a Sync-type LCD, as illustrated in Figure 1-1.

	MPU-type LCD	Sync-type LCD
Transmission Method	The host writes image data to the LCD memory through a transmission interface, and the LCD controller handles direct RGB and sync signal output to the panel.	The host outputs RGB and sync signals directly to the display panel for rendering images.
Memory and Bandwidth Usage	The host transfers small image data batches to the LCD memory, where the LCD panel periodically reads and outputs the image, minimizing memory and bandwidth usage on the host.	The host must store the complete image data in its own memory, which is then read by the built-in display controller and output to the LCD panel. The controller requires high bandwidth to periodically access the memory to maintain image output, resulting in relatively high memory and bandwidth usage on the host.
Transmission Interface	The LCD module can connect to the host via EBI i80 (parallel interface) or SPI (serial interface). EBI provides higher data transfer rates, making it suitable for high-resolution or high-refresh-rate applications, while SPI offers advantages in terms of fewer pins and simpler connections, making it ideal for resource-constrained system designs.	The 24-bit RGB interface (RGB888) uses 8-bit data for each of the red, green, and blue channels for display output, accompanied by synchronization signals (such as HSYNC, VSYNC, DE, and PCLK) to achieve real-time image rendering. This interface provides high-quality and low-latency image display, but requires more I/O pins and transmission bandwidth.
Hardware Cost	Panels with built-in controllers and display memory can reduce the burden on the host, but due to the additional hardware design, the module cost is relatively higher.	The panels do not have built-in display controllers or display memory, and require the host to handle image output processing. However, by omitting the related hardware design, the module cost is relatively lower.

Table 1-1 Comparison Table of MPU-type and Sync-type LCD Panel

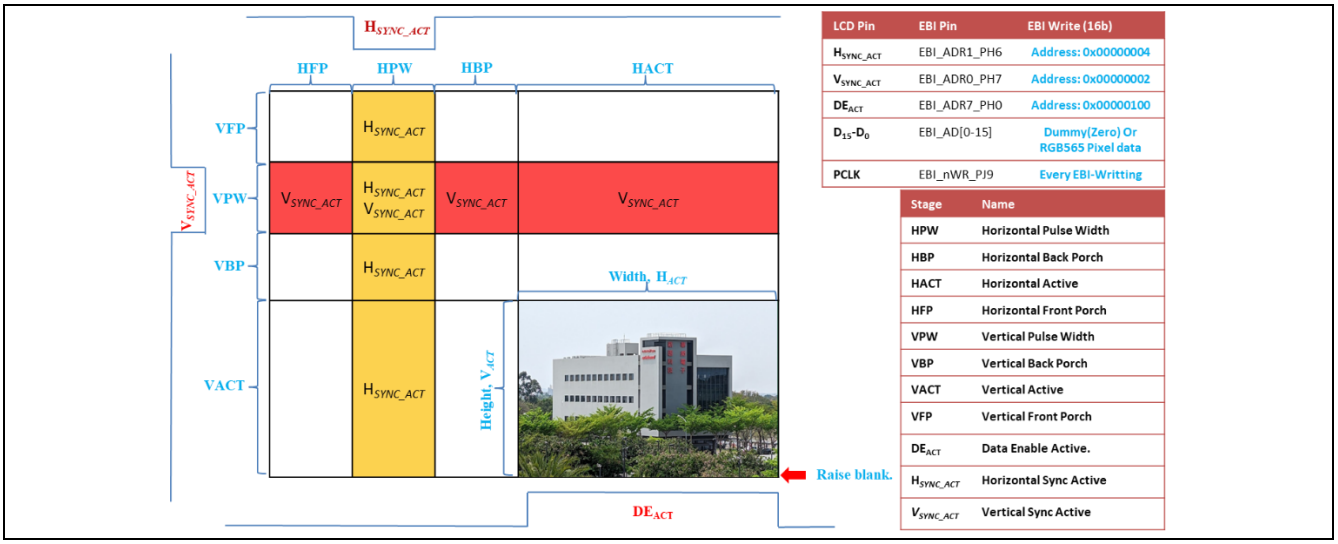

Figure 1-1 EBI Driving Sync Signal LCD Panel

1.1 Principle

This example code implements a Sync-type LCD controller, utilizing the EBI (External Bus Interface) along with a PDMA or GDMA controller to perform memory-to-memory transfers. It transmits image pixel data to the Sync-type LCD Panel. The operating principle involves constructing DMA descriptors for V Lines × H Stages, where:

- V (Vertical) represents the number of scan lines corresponding to the LCD timing parameters: Vertical Pulse Width (VPW), Vertical Back Porch (VBP), Vertical Active (VA), and Vertical Front Porch (VFP).
- H (Horizontal) represents the four stages of horizontal timing: Horizontal Pulse Width (HPW), Horizontal Back Porch (HBP), Horizontal Active (HA), and Horizontal Front Porch (HFP).

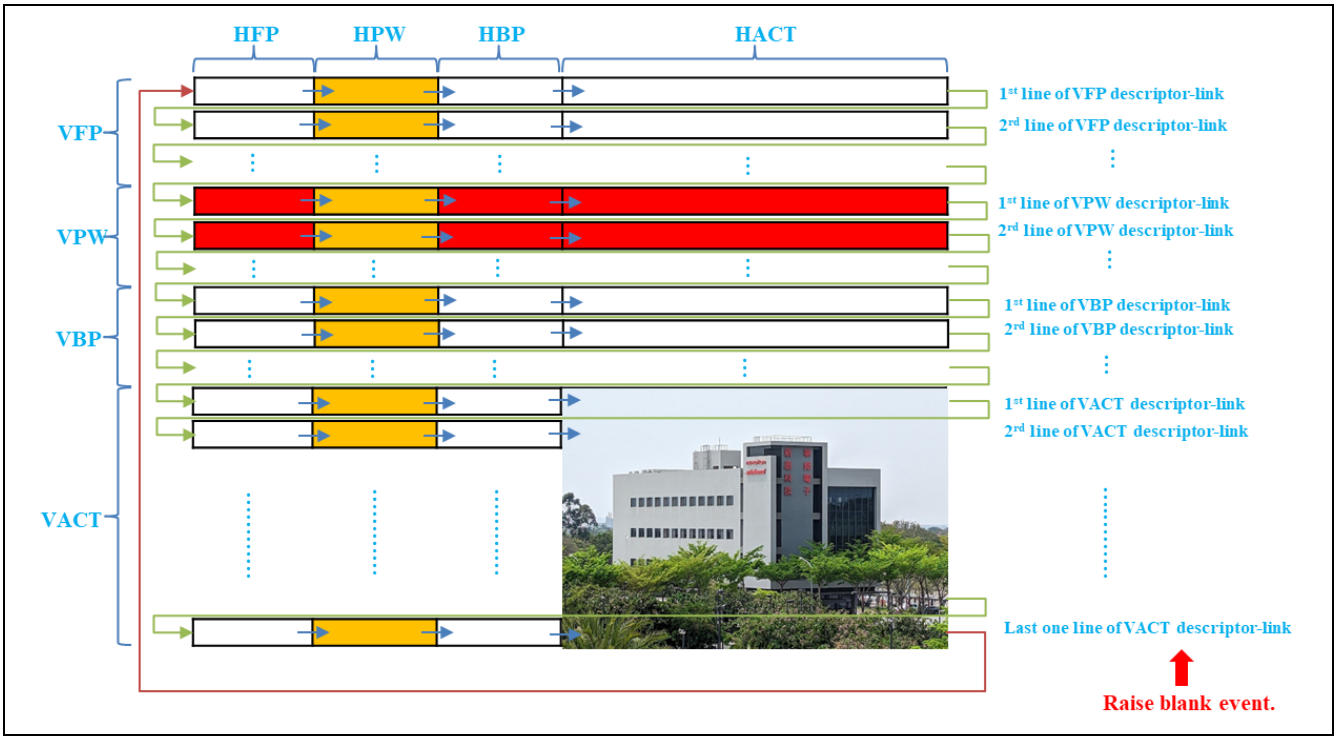

Figure 1-2 V Lines × H Stages DMA Description

As shown in Figure 1-2, all DMA descriptors are linked sequentially, with the NEXT parameter of the last descriptor set to the address of the first descriptor, forming a circular descriptor chain. Additionally, the last descriptor is configured to trigger a transfer completion interrupt response, as detailed in Table 1-2. In the interrupt service routine (ISR), the driver updates the starting address of the descriptor data transfer for the VACT and HACT stages. This ensures that each frame switches to the starting address of the new image memory, enabling smooth dynamic screen transitions and preventing screen tearing (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.

Table 1-2 Mapping of Each LCD Timing Stage and H_{SYNC_ACT}, V_{SYNC_ACT} and DE_{ACT} Pins

Since different types of displays have varying levels of support for sync signals, some panels support the DE_{ACT}-ONLY sync mode, meaning that they only require the DE_{ACT} signal input and do not need H_{SYNC_ACT} or V_{SYNC_ACT} signals. When operating a DE_{ACT}-ONLY display, the number of DMA descriptors can be significantly reduced, thereby conserving system memory and bandwidth usage, as detailed in Table 1-2 and Table 1-3.

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.

Table 1-3 Mapping of Each LCD Timing Stage and DE_{ACT}-Only Pin

As shown in Figure 1-3, the three LCD timing sync signals — H_{SYNC_ACT}, V_{SYNC_ACT}, and DE_{ACT} — are emulated using the EBI address pins ADR1, ADR0, and ADR7, respectively. By assigning different address decodings, the LCD sync signals are generated, and the PCLK signal is connected to the EBI-nWR pin. The 16-bit EBI data bus $D_{[15:0]}$ is mapped to the LCD color signals as follows:

- R[7:3] (Red), G[7:2] (Green), and B[7:3] (Blue).
- The additional R[2:0], G[1:0], and B[2:0] can be derived from R[7:5], G[7:6], and B[7:5], respectively, to perform RGB888 dithering, enhancing color accuracy.

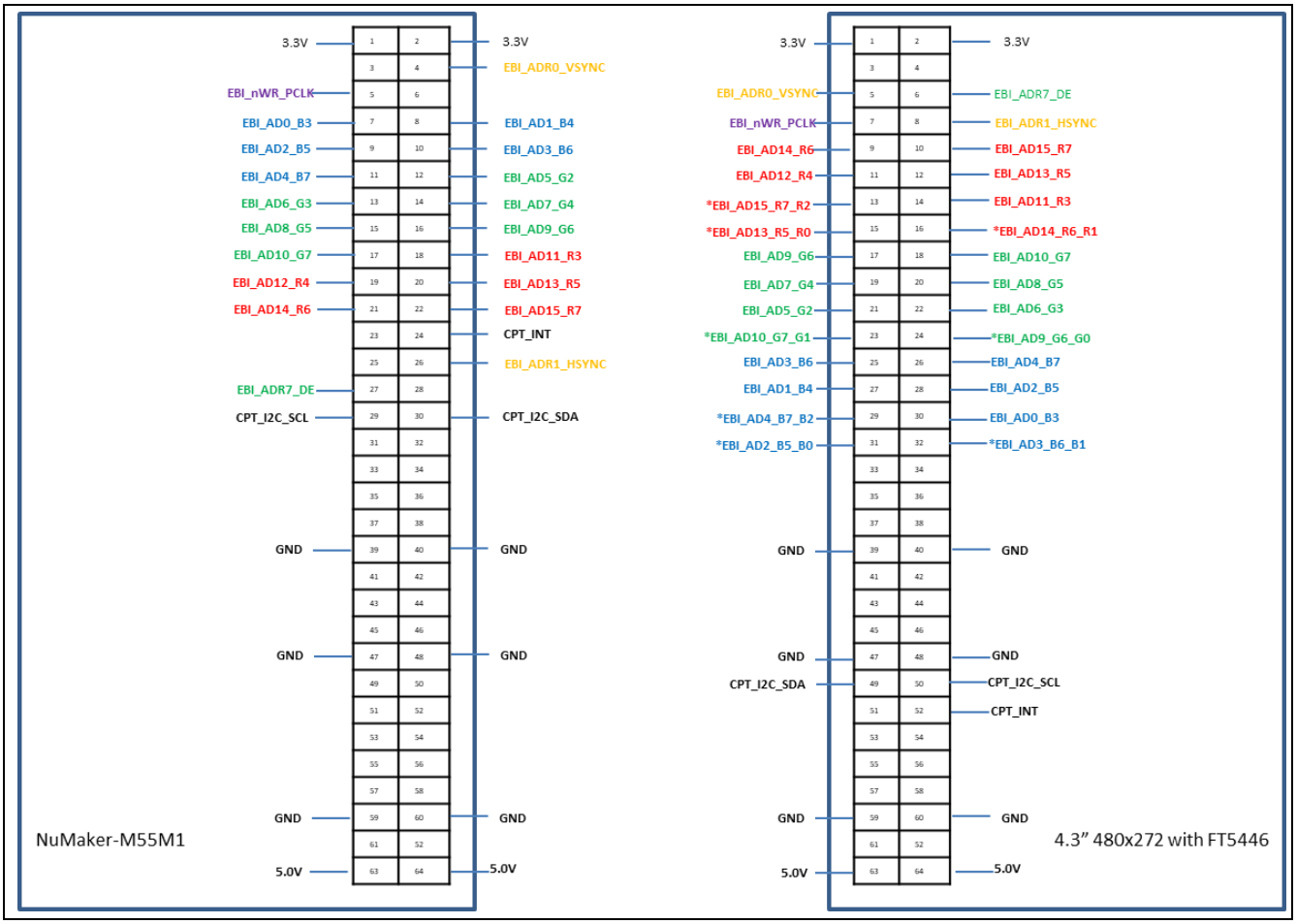

Figure 1-3 EBI16-to-RGB24 Signal Pinout Diagram

1.2 Execution Result

This example code provides two target build options: WQVGA_GDMA and WQVGA_PDMA, which utilize the GDMA and PDMA controllers, respectively, to implement the self-refreshing display function, as illustrated in Figure 1-4. During execution, the system can be monitored using the VCOM function of Nu-Link2-Me, in combination with a terminal software on a PC. The default communication parameters are 115200N81. As shown in Figure 1-5 and Figure 1-6, the terminal output and the rendered image on the screen are displayed.

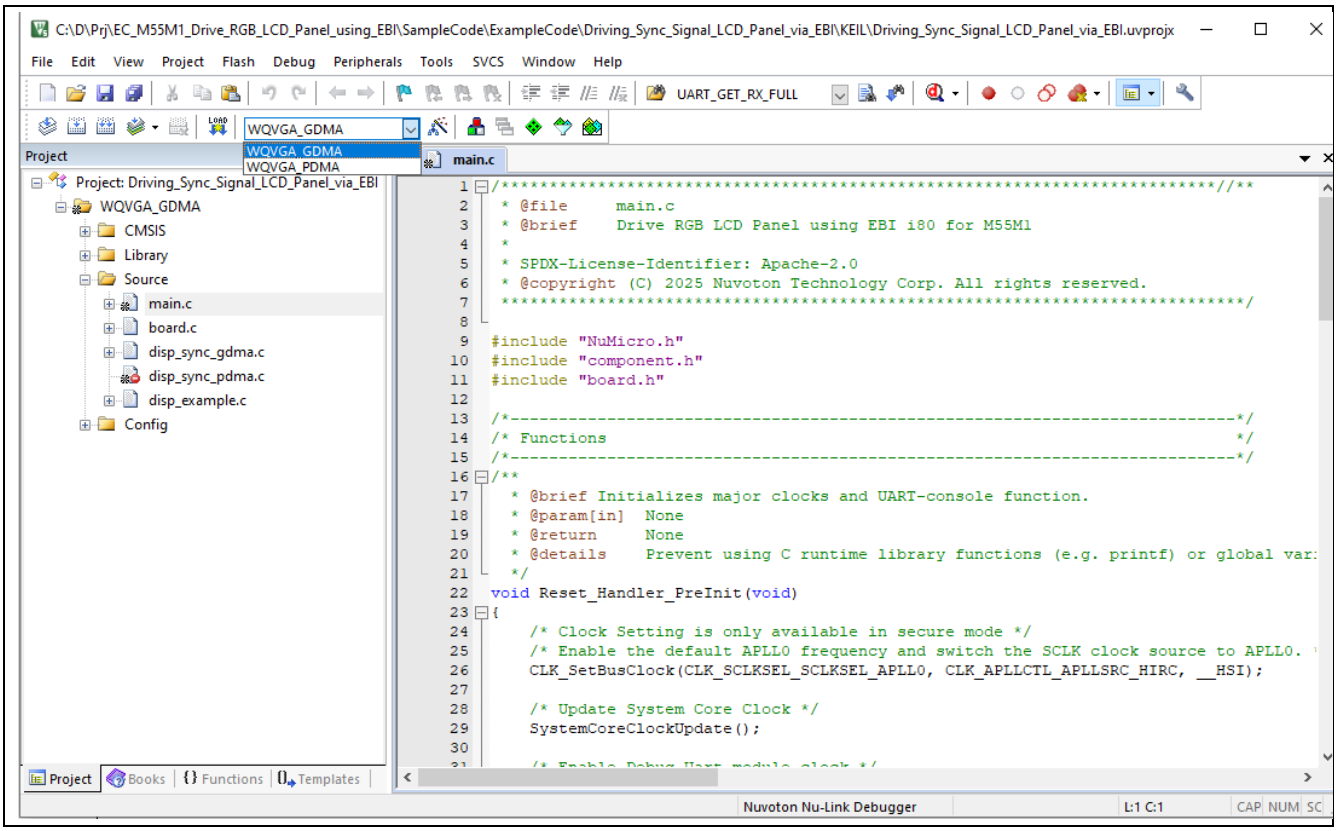

Figure 1-4 Selecting Build Options and Compiling

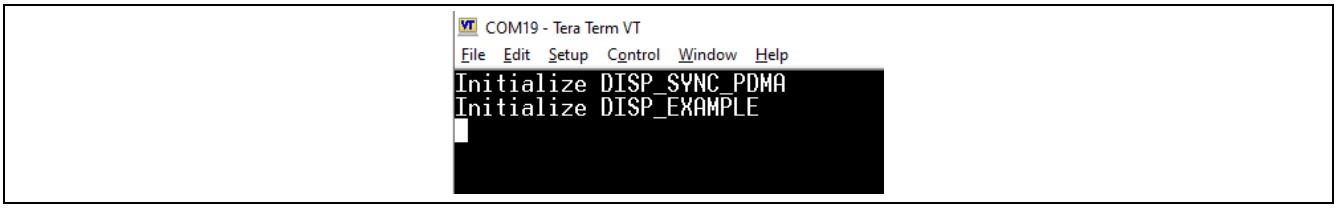

Figure 1-5 Execution Screen

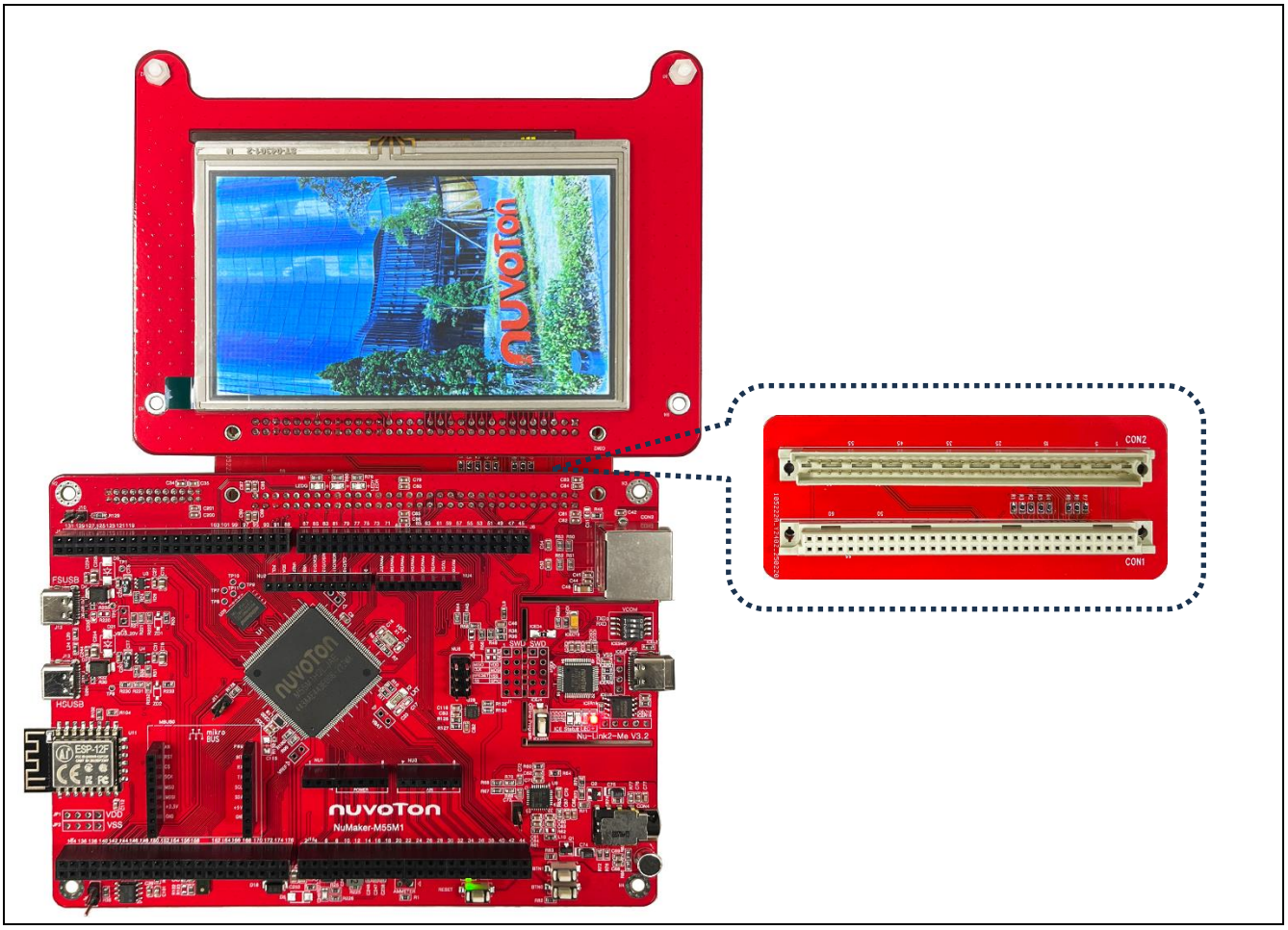

Figure 1-6 Image Display on Screen

2. Code Description

This example code handles the initialization and termination processes of each component, with the main program located in main.c. The initialization and termination functions, along with configuration files, are implemented in disp_sync_gdma.c, disp_sync_pdma.c, disp_example.c, and disp.h. Each component implementation uses the COMPONENT_EXPORT macro to register the component's name (.name), initialization callback function (.initialize), and termination callback function (.finalize). The registration data for these components is stored in the CompInitTab section, allowing the main program to call and execute them.

```
/**
  * @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

In the implementation of disp_sync_gdma.c, the function disp_gdma_dsc_init sequentially creates V Lines × H Stages DMA descriptors. For each different H/VLINE Stage, it assigns various memory-to-memory operation attributes and write addresses, outputting H_{SYNC_ACT}, V_{SYNC_ACT}, and DE_{ACT} signals to the LCD panel, along with the pixel data from the VRAM buffer. These descriptors are stored in a non-cacheable memory region, with the variable name s_sDecLCD and attribute declaration as NVT_NONCACHEABLE. This means that when the CPU writes data to these memory addresses, the data bypasses the CPU's data cache and is written directly to memory. During the interrupt service routine, when handling the Blank event, the program updates the starting address parameters of all VACT.HACT DMA descriptors to switch to different VRAM buffers. This approach avoids the need to repeatedly call the SCB_CleanDCache_by_Addr function to refresh the starting address parameters, thus improving the efficiency of switching between different VRAM buffers.

```
#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

In the implementation of disp_sync_pdma.c, the disp_pdma_dsc_init function sequentially creates V Lines × H Stages DMA descriptors. For each different H/VLINE Stage, it assigns various memory-to-memory operation attributes and write addresses, outputting H_{SYNC_ACT}, V_{SYNC_ACT}, and DE_{ACT} signals to the LCD panel, along with pixel data from the VRAM buffer. These descriptors are stored in a non-cacheable memory region, with the variable name s_sDecLCD and the attribute declaration as NVT_NONCACHEABLE. This means that when the CPU writes data to these memory addresses, the data bypasses the CPU's data cache and is written directly to memory. During the interrupt service routine, when handling the Blank event, the program updates the starting address parameters of all VACT.HACT DMA descriptors to switch to different VRAM buffers. This method avoids repeatedly calling the SCB_CleanDCache_by_Addr function to refresh the starting address parameters, thus improving the efficiency of switching between different VRAM buffers.

```
#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

In the implementation of disp_example.c, the .incbin assembly directive is used to embed two RGB565 format images directly into the program, storing them in FLASH to reduce compile time. In the code, PATH_IMAGE1_BIN and PATH_IMAGE2_BIN are defined as the file paths for the two image files, stored in the disp.h header file. Image files can be imported and downloaded in RGB565 format by using the online tool Image to RGB565 Converter (https://longfangsong.github.io/en/image-to-rgb565/), which supports importing PNG files with full-screen resolution. In the disp_example_init function, the callback function for the Blank event (disp_example_blankcb) is first registered. Then, the two RGB565 image data files are copied to the image buffer. Since the Cortex-M55 supports data cache, after the copy operation, the SCB_CleanDCache_by_Addr function is called to synchronize the image data to SRAM. In disp_example_blankcb, the disp_set_vrambufaddr function is used with different VRAM addresses to alternately update the screen, preventing screen tearing. However, due to the fast

update rate in the example, which leads to ghosting, the screen switching is conditioned to occur only after receiving 16 blank events.

```
#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 events, */
    /* 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

The disp.h header file is used to define the sync signal pins and timing parameters for the LCD panel. These parameters need to be configured according to the actual pin configuration of the board, such as:

- CONFIG_LCD_PANEL_USE_DE_ONLY: When undefined, the system will enable the output of V_{SYNC_ACT}, H_{SYNC_ACT}, and DE_{ACT} signals to the screen. If defined, only the DE_{ACT} signal will be output.
- CONFIG_DISP_*_ACTIVE_LOW: Set to 0 to indicate the corresponding signal is Active-High; otherwise, it will be Active-Low.
- CONFIG_DISP_*_BITIDX: The setting value is the pin number of the corresponding EBI_ADRx pin plus 1.
- CONFIG_TIMING_*: Used to configure the timing parameters for the sync signal screen, which should be adjusted according to the panel's datasheet.

```
#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. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M55M1_Series_BSP_CMSIS_V3.01.001
- IDE version
  - Keil uVision 5.40

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M55M1 V1.0
  - EBI16-RGB24 glue board
  - Sync-type LCD Panel; Resolution: WQVGA
- Pin connection
  - Assemble the NuMaker-M55M1 evaluation board, EBI16-RGB24 glue board, and the synchronous signal LCD panel.
  - Use an USB Type-C to Type-A cable to connect the NuMaker-M55M1 evaluation board to the computer.

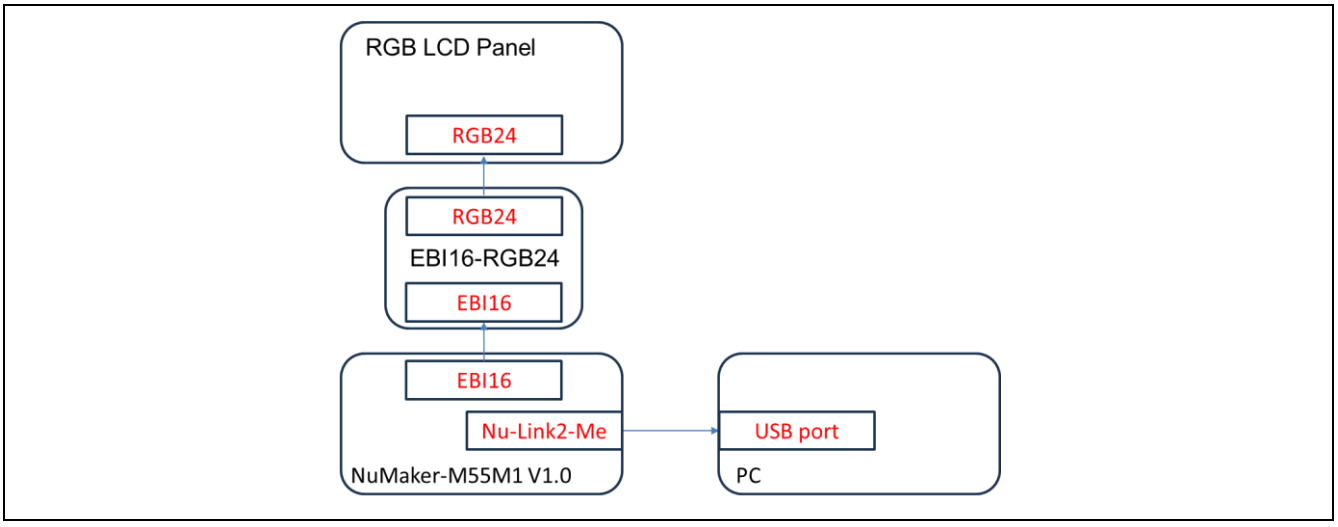

Figure 3-1 Pin Connection Diagram

4. Directory Structure

The directory structure is shown below.

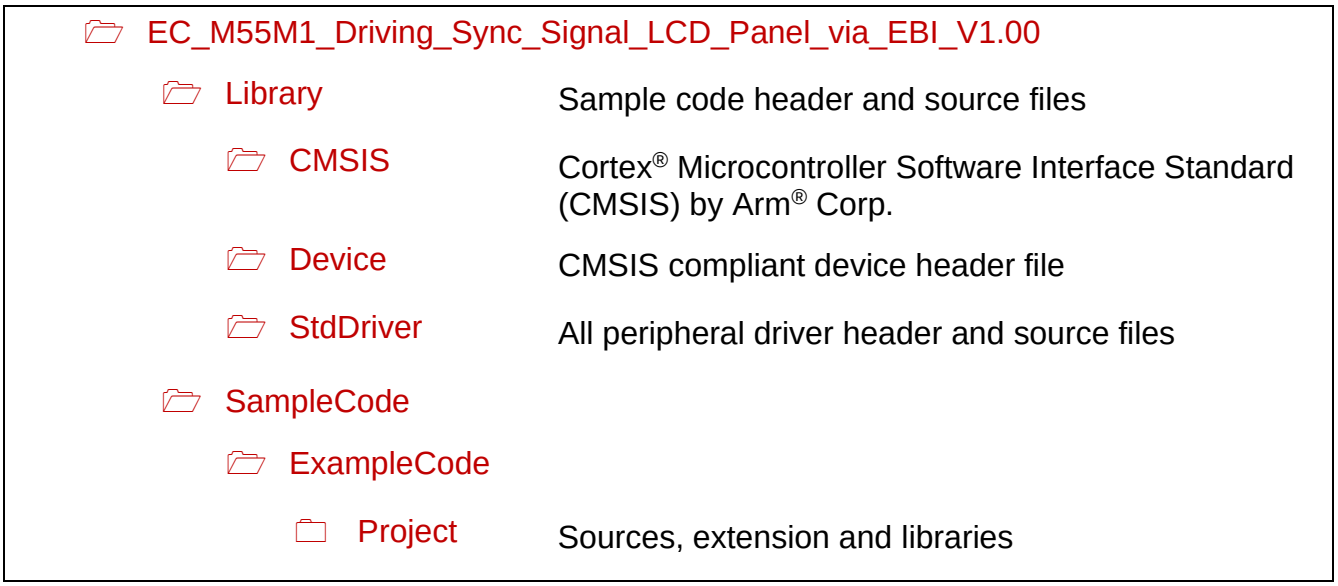

Figure 4-1 Directory Structure

5. Example Code Execution

Browse the sample code folder as described in the Directory Information section and double-click *EC_M55M1_Driving_Sync_Signal_LCD_Panel_via_EBI.uvprojx*.

1. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
2. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2025.03.21	1.00	Initial version

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.