Example Code Introduction for 32-bit NuMicro® Family

Drive RGB LCD Panel Using EBI

**grity**

|  |  |
| --- | --- |
| **Document Information** | |
| Application | This example code uses the M55M1 EBI to drive a synchronous signal RGB 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](http://www.nuvoton.com)

# Overview

The commonly used LD interface for microcontrollers is the MPU-type LCD. Since these displays 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, 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.

|  |
| --- |
|  |

Figure 1‑1 EBI Driving Sync Signal LCD Screen

|  |  |  |
| --- | --- | --- |
|  | **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 screen. 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** | Screens 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. | These screens 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

## 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 screen. 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).

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

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 *HSYNC\_ACT* to Inactive**  **Set *VSYNC\_ACT* to Inactive**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | **Set *HSYNC\_ACT* to Active**  **Set *VSYNC\_ACT* to Inactive**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | **Set *HSYNC\_ACT* to Inactive**  **Set *VSYNC\_ACT* to Inactive**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | **Set *HSYNC\_ACT* to Inactive**  **Set *VSYNC\_ACT* to Inactive**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** |
| **VPW** | **Set *HSYNC\_ACT* to Inactive**  **Set *VSYNC\_ACT* to Active**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | **Set *HSYNC\_ACT* to Active**  **Set *VSYNC\_ACT* to Active**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | **Set *HSYNC\_ACT* to Inactive**  **Set *VSYNC\_ACT* to Active**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | **Set *HSYNC\_ACT* to Inactive**  **Set *VSYNC\_ACT* to Active**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** |
| **VBP** | **Set *HSYNC\_ACT* to Inactive**  **Set *VSYNC\_ACT* to Inactive**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | **Set *HSYNC\_ACT* to Active**  **Set *VSYNC\_ACT* to Inactive**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | **Set *HSYNC\_ACT* to Inactive**  **Set *VSYNC\_ACT* to Inactive**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | **Set *HSYNC\_ACT* to Inactive**  **Set *VSYNC\_ACT* to Inactive**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** |
| **VACT** | **Set *HSYNC\_ACT* to Inactive**  **Set *VSYNC\_ACT* to Inactive**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | **Set *HSYNC\_ACT* to Active**  **Set *VSYNC\_ACT* to Inactive**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | **Set *HSYNC\_ACT* to Inactive**  **Set *VSYNC\_ACT* to Inactive**  **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | **Set *HSYNC\_ACT* to Inactive**  **Set *VSYNC\_ACT* to Inactive**  **Set *DEACT* to Active**  **Set *D[15-0]* to pixel data in every Line**  Raising blank event in latest descriptor. |

Table 1‑2 Correspondence of LCD Timing Stages and ***HSYNC\_ACT, VSYNC\_ACT*** and ***DEACT*** Pins

Since different types of displays have varying levels of support for sync signals, some screens support the ***DEACT-ONLY*** sync mode, meaning they only require the ***DEACT*** signal input and do not need ***HSYNC\_ACT*** or ***VSYNC\_ACT*** signals. When operating a ***DEACT-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 *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | | | |
| **VPW** |
| **VBP** |
| **VACT** | **Set *DEACT* to Inactive**  **Set *D[15-0]* to Dummy** | | | **Set *DEACT* to Active**  **Set *D[15-0]* to pixel data in every Line**  Raising blank event in latest descriptor. |

Table 1‑3 Correspondence of LCD Timing Stages and ***DEACT-Only*** Pin

As shown in Figure 1‑3, the three LCD timing sync signals—***HSYNC\_ACT***, ***VSYNC\_ACT***, and ***DEACT***—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 Pin Connection Diagram

## 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

# 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;

}

## disp\_sync\_gdma Component

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 ***HSYNC\_ACT***, ***VSYNC\_ACT***, and ***DEACT*** signals to the LCD screen, 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;

}

## disp\_sync\_pdma Component

In the implementation of ***disp\_sync\_pdma.c***, the function ***disp\_pdma\_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 ***HSYNC\_ACT***, ***VSYNC\_ACT***, and ***DEACT*** signals to the LCD screen, 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;

}

## disp\_example Component

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;

}

## Disp Driver Parameters Introduction

The ***disp.h*** header file is used to define the sync signal pins and timing parameters for the LCD screen. 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 ***VSYNC\_ACT***, ***HSYNC\_ACT***, and ***DEACT*** signals to the screen. If defined, only the ***DEACT*** 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 screen'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) \*/

# Software and Hardware Requirements

## Software Requirements

BSP version

M55M1\_Series\_BSP\_CMSIS\_V3.01.001。

IDE version

Keil uVision 5.40。

## Hardware Requirements

Circuit components

NuMaker-M55M1 V1.0

EBI16-RGB24 Glue board

Sync-type LCD Panel, Resolution: WQVGA

Schematic diagram of the route

Assemble the NuMaker-M55M1 V1.0, EBI16-RGB24 adapter board, and the synchronous signal LCD panel.

Use an USB Type-C to Type-A cable to connect the NuMaker-M55M1 V1.0 development board to the computer.

|  |
| --- |
|  |

Figure 3‑1 Pin Connection Diagram

# Directory Structure

The directory structure is shown in the diagram below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| * EC\_M55M1\_Drive\_RGB\_LCD\_Panel\_using\_EBI\_V1.00  |  |  | | --- | --- | | * Library | Sample code header and source files | | * + CMSIS | Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp. | | * + Device | CMSIS compliant device header file | | * + StdDriver | All peripheral driver header and source files | | * + SampleCode |  | | * + ExampleCode |  | |
| |  |  | | --- | --- | | * + - Project | Sources, extension and libraries | |

Figure 4‑1 Directory Structure

# Example Code Execution

* + - * 1. Navigate to the ***ExampleCode*** section and enter the **KEIL** folder. Double-click on ***Drive\_RGB\_LCD\_Panel\_using\_EBI.uvprojx***.
        2. Enter the compilation mode interface:

Compile the code.

Download the code to flash memory.

Run the code.

# Revision History

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision** | **Description** |
| 2025.03.21 | 1.00 | Initial version |

**A notice with text and black text

Description automatically generated with medium confidence**