LCD Display on M55M1 Using GDMA

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code demonstrates how to perform partial screen updates on the LCD using GDMA with emWin AppWizard on the M55M1 series microcontroller (MCU).
BSP Version	M55M1_Series_BSP_CMSIS_V3.01.003
Hardware	NuMaker-X-M55M1

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

This sample code demonstrates how to update an LCD display on the NuMaker-X-M55M1 development platform using emWin AppWizard together with GDMA (Graphic Direct Memory Access). Compared to traditional software-based memory copy methods, the parallel processing capability of GDMA effectively reduces CPU utilization. By minimizing the image rendering workload on the CPU, it frees up processor resources for other tasks. This ensures that the system maintains overall smoothness and real-time responsiveness during GUI (Graphical User Interface) execution, making it an ideal practical reference for high-performance embedded systems.

1.1 NuMaker-X-M55M1 Development Platform

The M55M1 series microcontrollers are built on the Arm® Cortex®-M55 core, operating at frequencies of up to 220 MHz. They incorporate Arm® Helium™ DSP technology to significantly enhance digital signal processing capabilities, and integrate the Arm® Ethos™-U55 microNPU, providing strong performance for both DSP tasks and machine learning inference. This architecture enables efficient acceleration of machine learning workloads on edge devices.

This example utilizes the NuMaker-TFT-LCD5 display module, featuring a resolution of 800 × 480 and an RGB565 (16-bit) color format.



Figure 1-1 NuMaker-X-M55M1 Evaluation Board



Figure 1-2 NuMaker-TFT-LCD5 V1.2 Display Module

1.2 SEGGER EmWin AppWizard

This example uses SEGGER's emWin AppWizard as the GUI framework. It provides a rich set of widgets and various interaction functions, allowing users to develop graphical interfaces efficiently and significantly reduce development time. After completing the basic GUI framework, the project can be exported from the tool and imported into the Keil IDE for further functional implementation and testing.

Nuvoton also provides an emWin package (https://www.nuvoton.com/resource-download.jsp?tp_GUID=SW15202503130722211 9) including all required libraries, drivers, and supporting development tools. Users can download the latest version from the Nuvoton official website and refer to the User Guide for instructions on how to import an emWin AppWizard project into the Keil project.

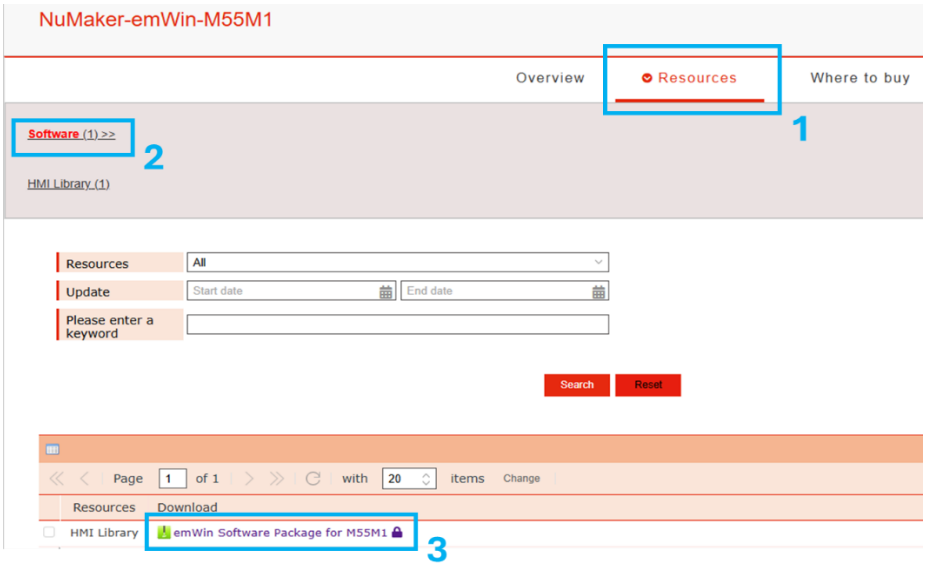


Figure 1-3 How to Download the Package in Website

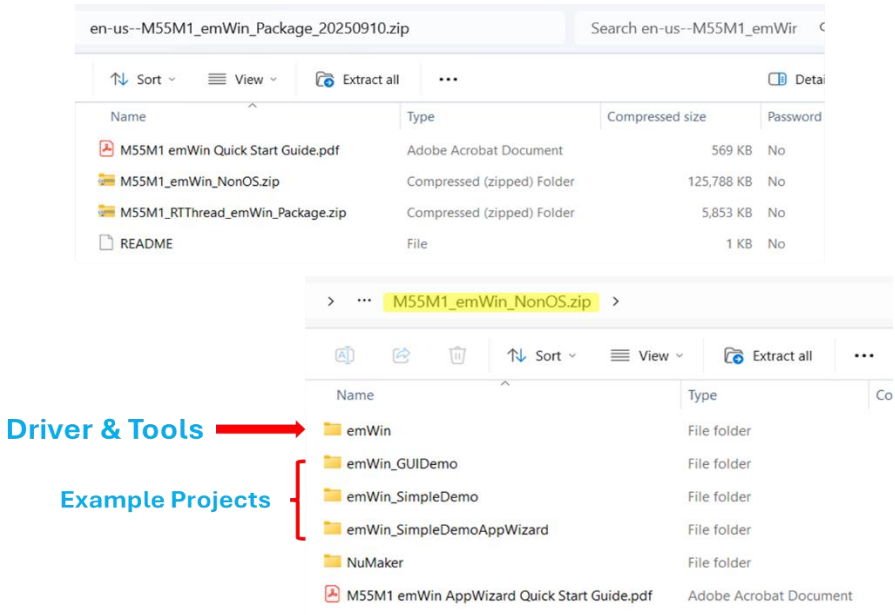


Figure 1-4 EmWin Package Contents

1.3 GDMA Working Principle

In traditional CPU-based data transfer, each pixel is copied individually (byte-by-byte), which consumes significant processing resources and may lead to UI update lag. The core concept of GDMA lies in offloading both data addressing and transfer execution from the CPU to a dedicated hardware controller.

The CPU only needs to configure a one-time task in the GDMA registers, specifying the source address, destination address, and data length, to initiate the transfer. Once the channel is enabled, GDMA autonomously issues read and write requests on the bus according to the configuration. This means that data movement no longer requires the CPU to handle each byte-level access instruction. GDMA automatically manages address incrementing, burst transfers, and synchronization with peripherals, allowing the CPU to focus on more critical data processing tasks.

As shown in Figure 1-5, during partial screen updates (Dirty Region Update), GDMA transfers only the regions of the display that have actually changed (dirty regions) to the LCD VRAM, rather than copying the entire frame buffer.

- pSRC: Pointer to source.
- pDst: Pointer to destination.
- Dirty Region: The area of the screen that requires updating.
- Stride: The width of each row in pixels (e.g., 800 pixels).
- Offset: The calculated starting address for data transfer, for example: pBG + (800 × y0 + x0).
- DMA 2D Transfer: GDMA transfers data based on the size of the dirty region (xSize × ySize) and the stride, automatically skipping unchanged pixel areas (skip stride gap).
- 16-bit RGB Data: This example uses the RGB565 color format for data transfer.

With this approach, GDMA updates only the regions that have actually changed, significantly reducing data transfer volume and CPU load. This helps prevent UI lag while maintaining efficient display performance.

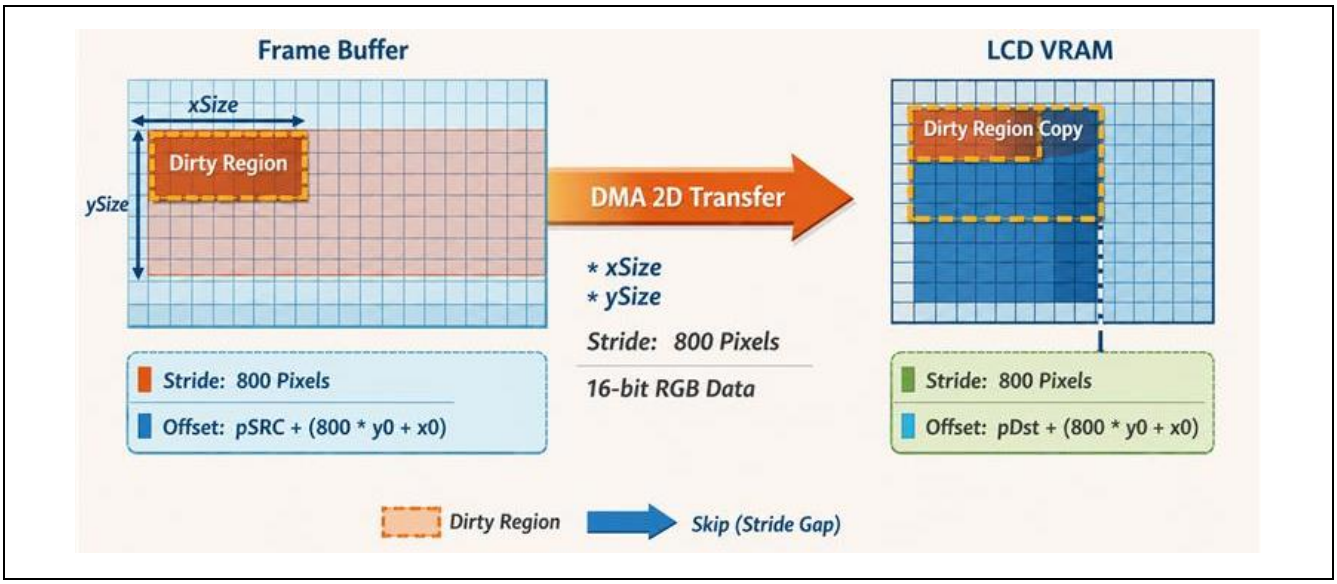


Figure 1-5 GDMA Workflow

1.4 System Flow and Design Description

Developed on the M55M1 platform, this example demonstrates a graphical user interface created via emWin AppWizard. By leveraging the GDMA controller, the system implements partial updates for the LCD display. This approach significantly offloads the CPU and improves frame update efficiency.

Furthermore, this project provides a performance benchmark comparing GDMA against traditional software-based memory copying. For detailed logic flows and performance metrics, please refer to Figure 1-6 and Figure 1-7.

Item	CPU memory copy	GDMA copy
Data flow	Memory → CPU → Memory	Memory → GDMA → Memory
CPU Participation	High (processing each stroke individually)	Low (Only setup)
Efficiency	Slower	High speed (burst)
CPU loading	High	Almost none
parallel	X	Can parallel with CPU

Figure 1-6 Comparison of Memory and GDMA Processing

Upon power-up, the system performs initialization, including LCD and GUI-related configurations (e.g., *LCD_X_DisplayDriver()*), followed by the execution of the GUI application via *MainTask()*. In this example, clicking a button on the screen triggers the data copy function. Once the task is complete, *IMAGE_SetBitmap()* is called to specify the target Image Widget and refresh the display for that specific area. The execution flow is illustrated in Figure 1-7.

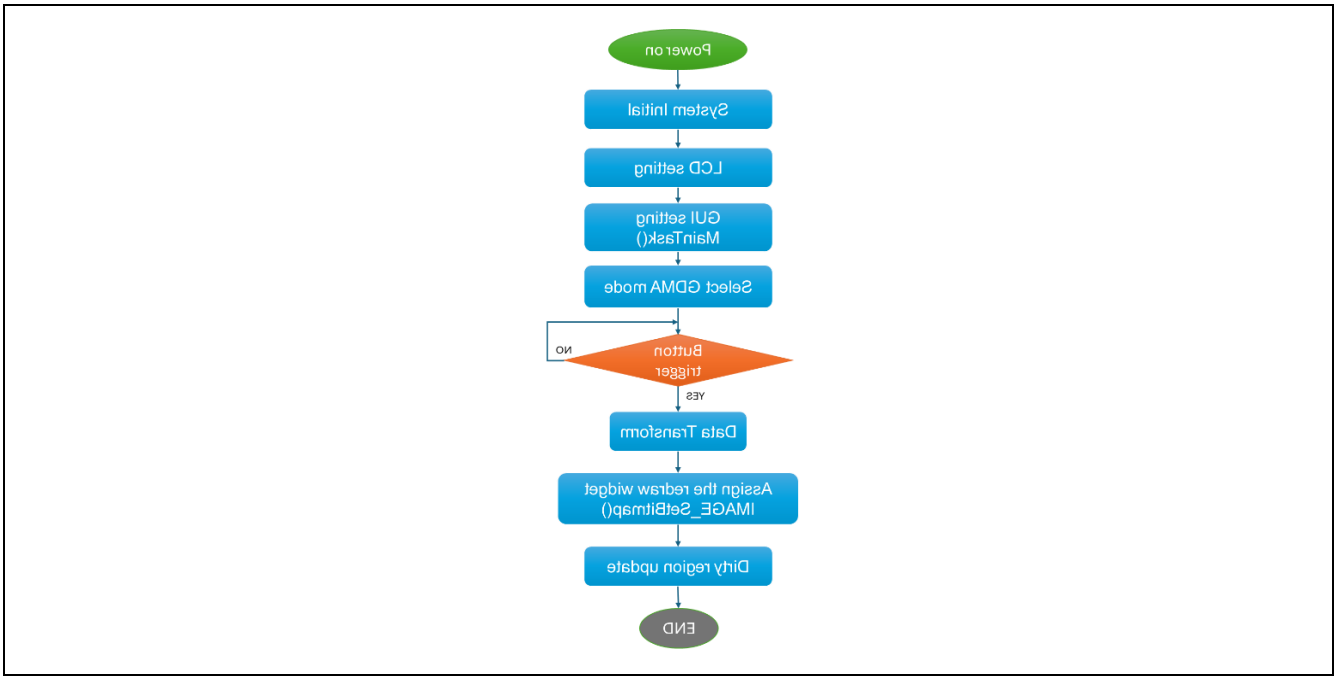


Figure 1-7 System Flowchart

1.4.1 Details of the process description are listed as follows:

- Call the display driver (*LCD_X_DisplayDriver()*)
- GUI Event Trigger

  - Initiates data copying by clicking a screen button (Button Widget).

- GDMA Setup

  - Source address: Original image data location.
  - Destination address: LCD frame buffer.
  - Transfer size: Width × Height × Bytes (per pixel).
  - Line offset: Stride (for 2D copying).
  - Transfer mode: Memory-to-Memory: Switch via GUI menu. Supports image rotation and mirroring across different modes

- GDMA Execution
- Frame Buffer Update

  - GDMA writes the image data for the designated region into the LCD frame buffer.

- Window Redraw Request (emWin Rendering/Invalidation)

  - Upon an event trigger (e.g., data update), emWin updates the specific widget and flags the affected region as an invalid area.

- Dirty Region / Invalid Area Calculation

  - emWin precisely calculates the dirty region to ensure only necessary pixels are redrawn. This optimizes display performance and reduces CPU load by avoiding a full-screen refresh.

- Panel Refresh (LCD Display Update)

  - The LCD controller continuously reads data from the frame buffer and outputs it to the panel, completing the display update.

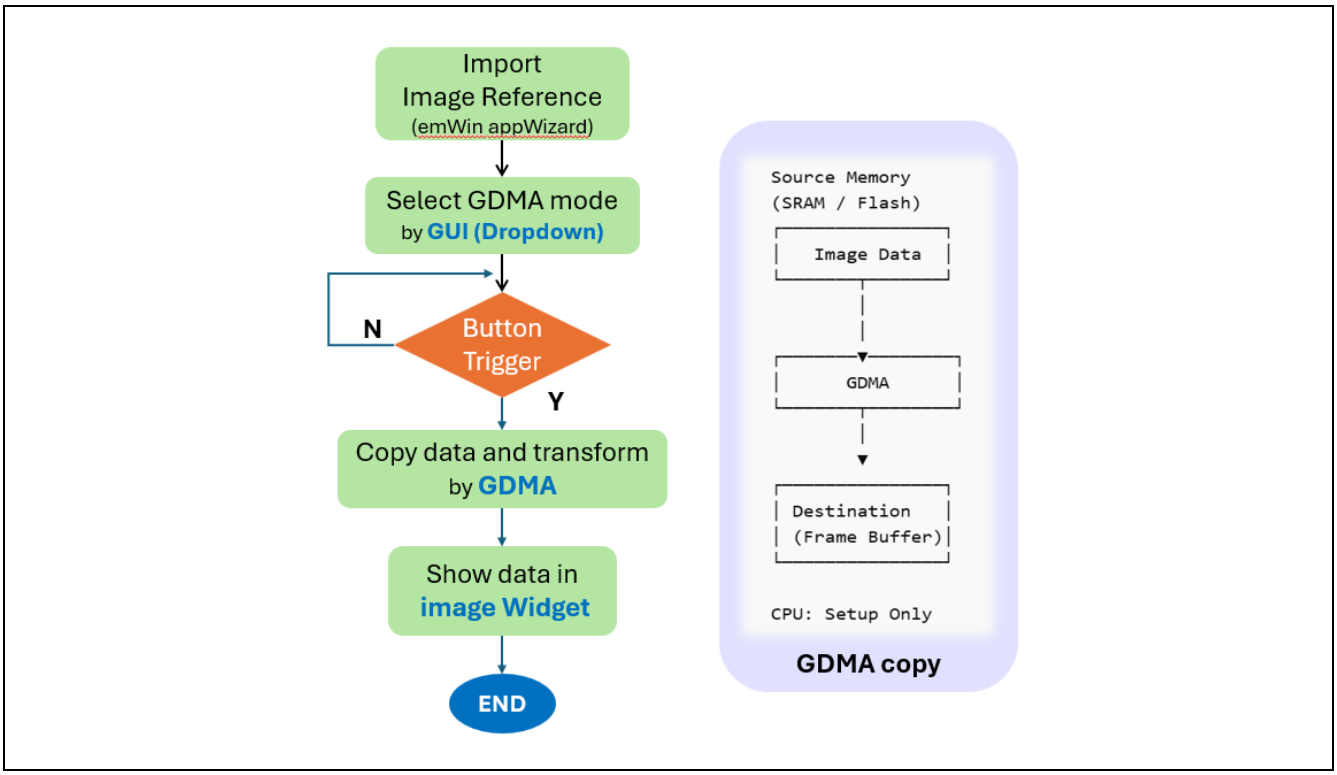


Figure 1-8 Program Flow

1.4.2 Configure Widgets of EmWin AppWizard

This example uses emWin AppWizard to develop the GUI. A dropdown menu (Dropdown Widget) is provided to select the desired GDMA data transfer mode, and a button (Button Widget) is used to trigger the data processing.

Two image objects (Image Widgets) are used by default to display the source image and the processed image. After selecting the processing mode, the user can press the button to start capturing image data and perform a partial screen update. The processed image is then rendered and displayed on the corresponding Image Widget.

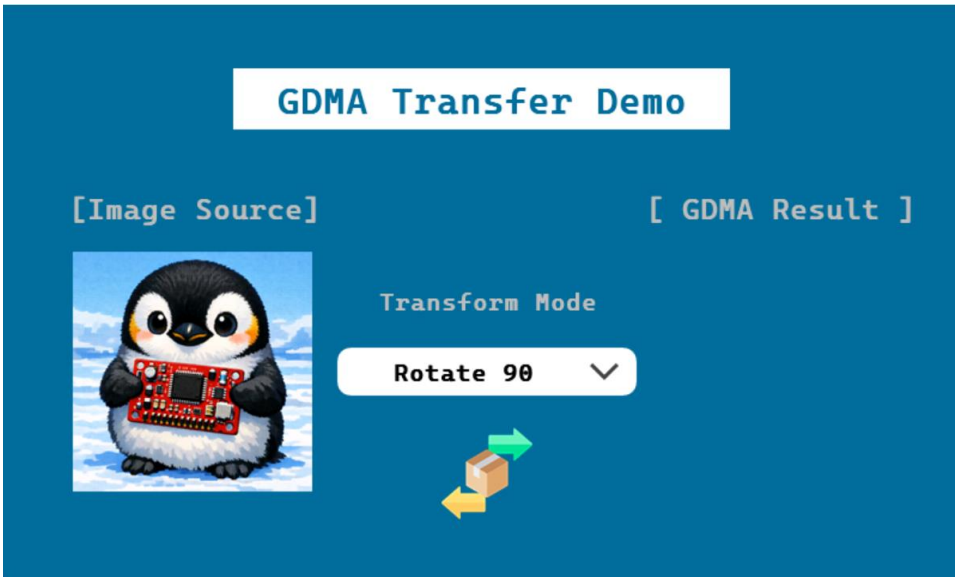


Figure 1-9 Default GUI of the Example

The description of of EmWin AppWizard widgets is shown in Figure 1-10. For detailed operation steps of emWin AppWizard, please refer to the document in the Nuvoton emWin package.

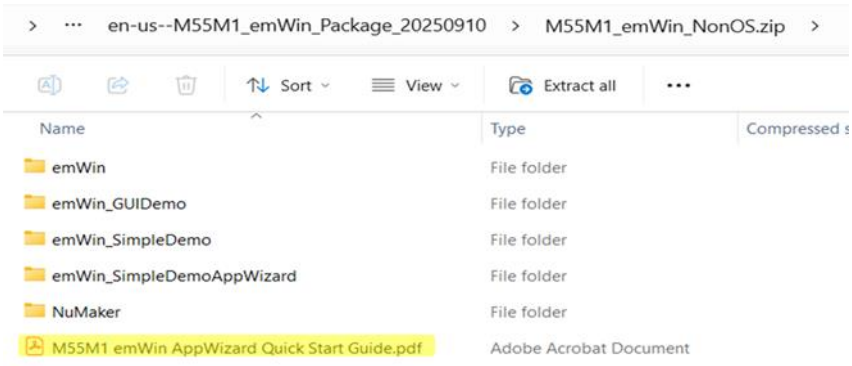


Figure 1-10 Document Location of the Package

Description	Appearance	Widget Type	Widget ID
Background 800x480		Box	ID_BOX_00
Instructions	GDMA Transfer Demo	Text	ID_TEXT_00

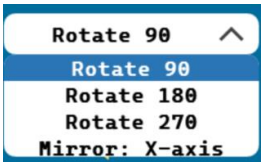
	[Image Source]		ID_TEXT_01
	[GDMA Result]		ID_TEXT_02
	Transform Mode		ID_TEXT_03
Target Image 200x200		Image	ID_IMG_SRC
Copied Image (GDMA results) 200x200	 Empty object (no fill)		ID_IMG_GDMA
Dropdown Menu (250x40)		Dropdown	ID_DROPDOWN
Copy Button 80x80		Button	ID_BTN_MOVE

Table 1-1 Description of EmWin AppWizard Widgets

After adding widgets, users can adjust the display format of the text widget in the properties panel, as shown in Figure 1-1.

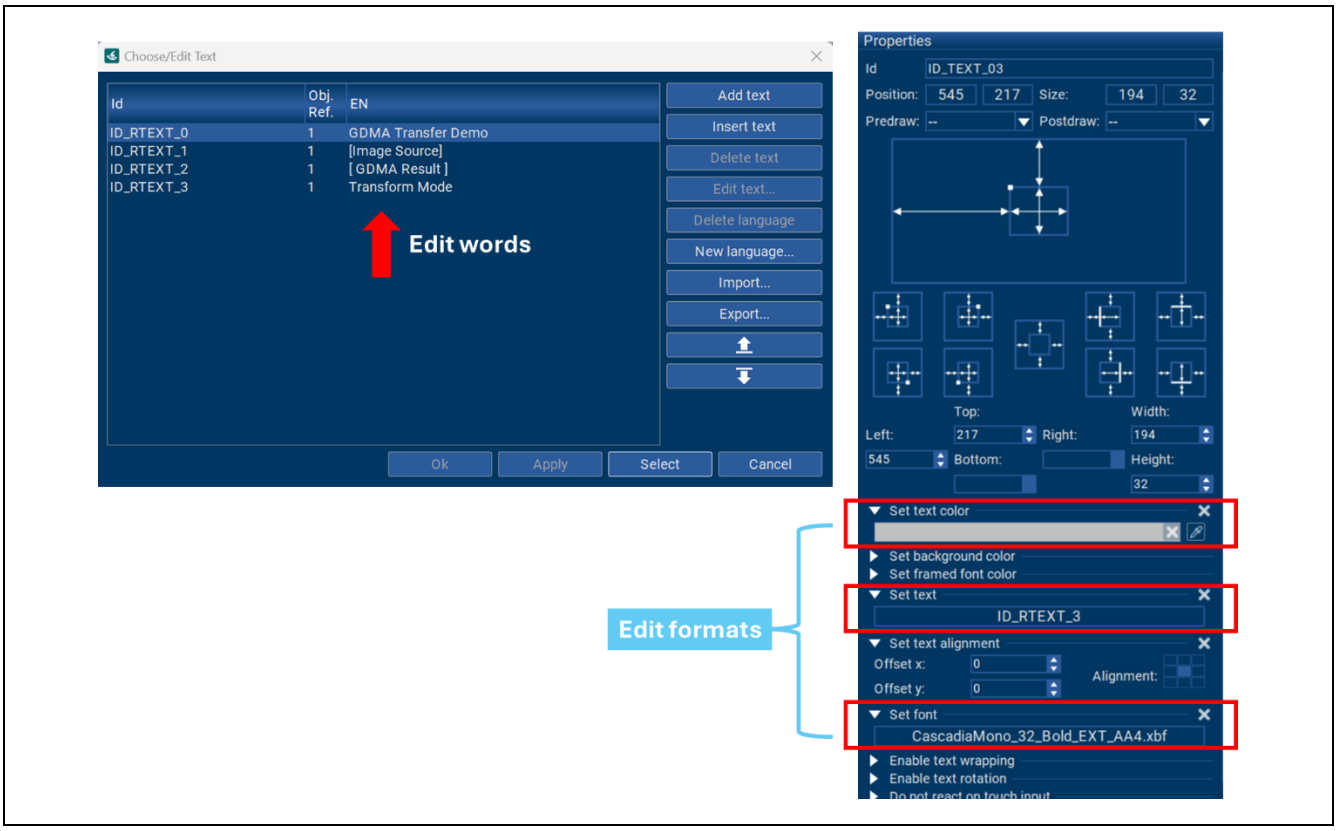


Figure 1-11 Properties of Text Widget

Users can modify the appearance through the "**Set Bitmap**" field of the button and image widgets. The widget *ID_IMG_SRC* will be the source image, so its format should be set to "**High Color (565), RB swap**" to prevent color distortion in the processed image.

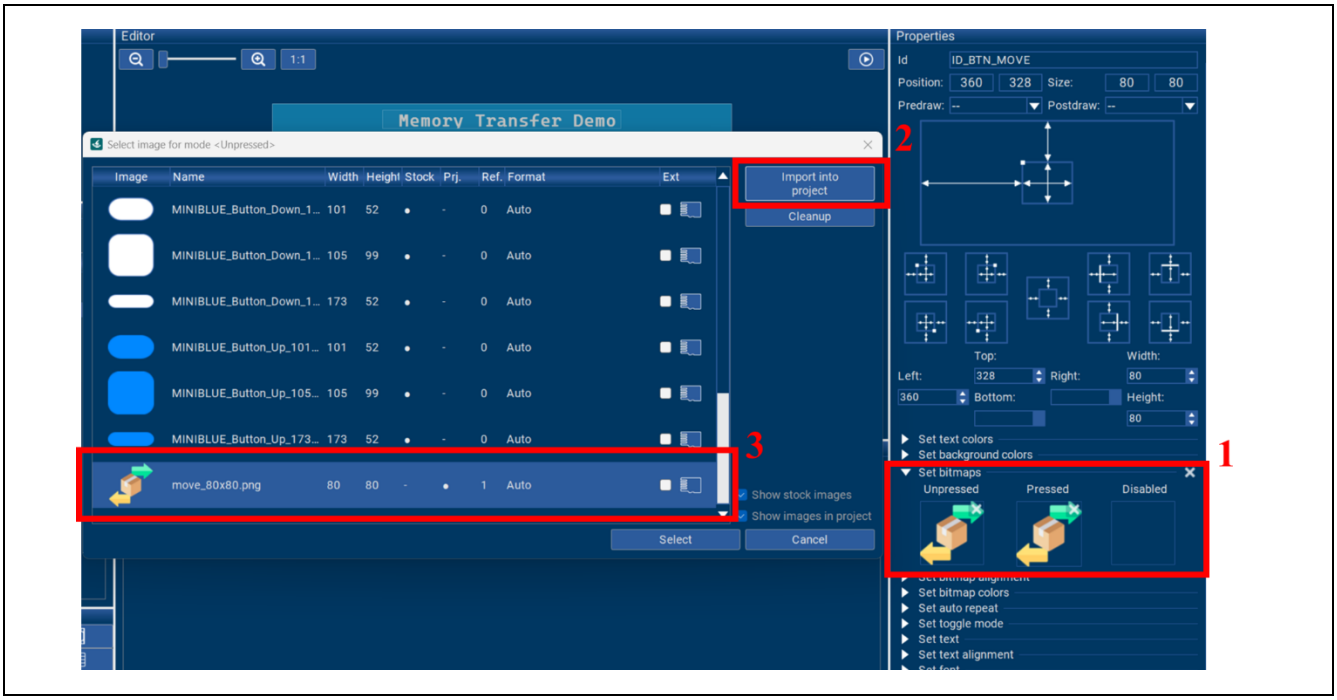


Figure 1-12 Properties of Button Widget

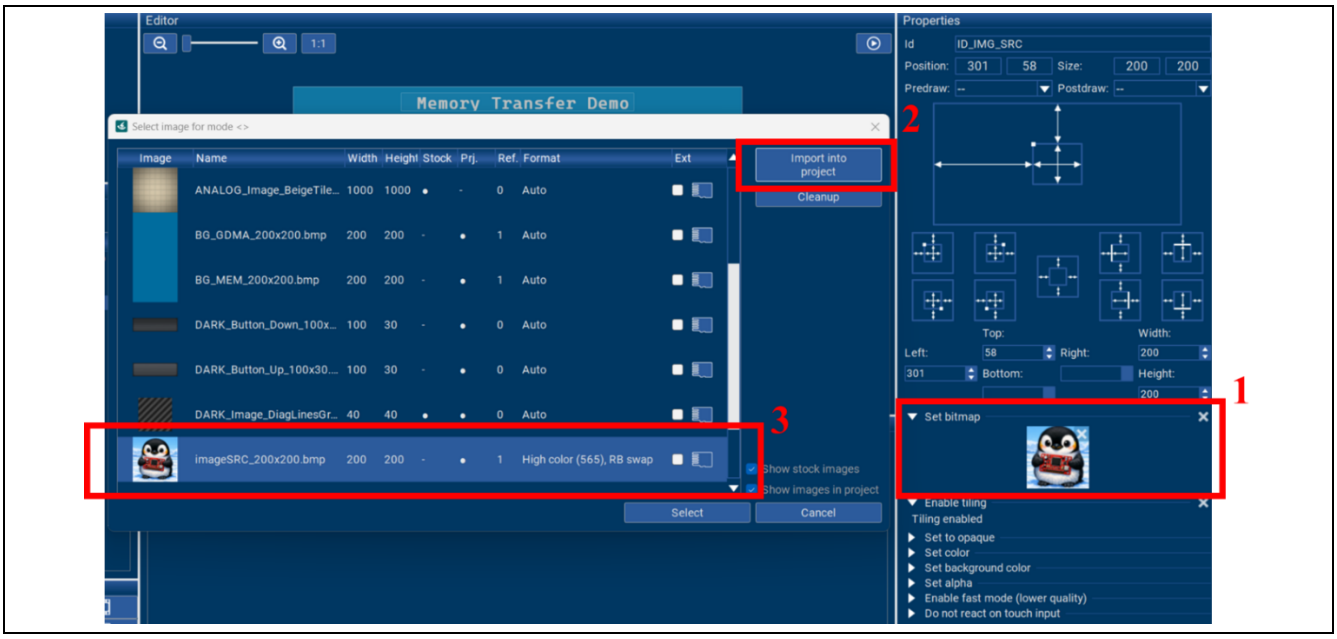


Figure 1-13 Properties of Image Widget

After adding a Dropdown object, you can adjust its appearance in the properties panel. You can also use "**Set Content**" to add the items you want to display in the menu, as shown in Figure 1-14 and Figure 1-15.

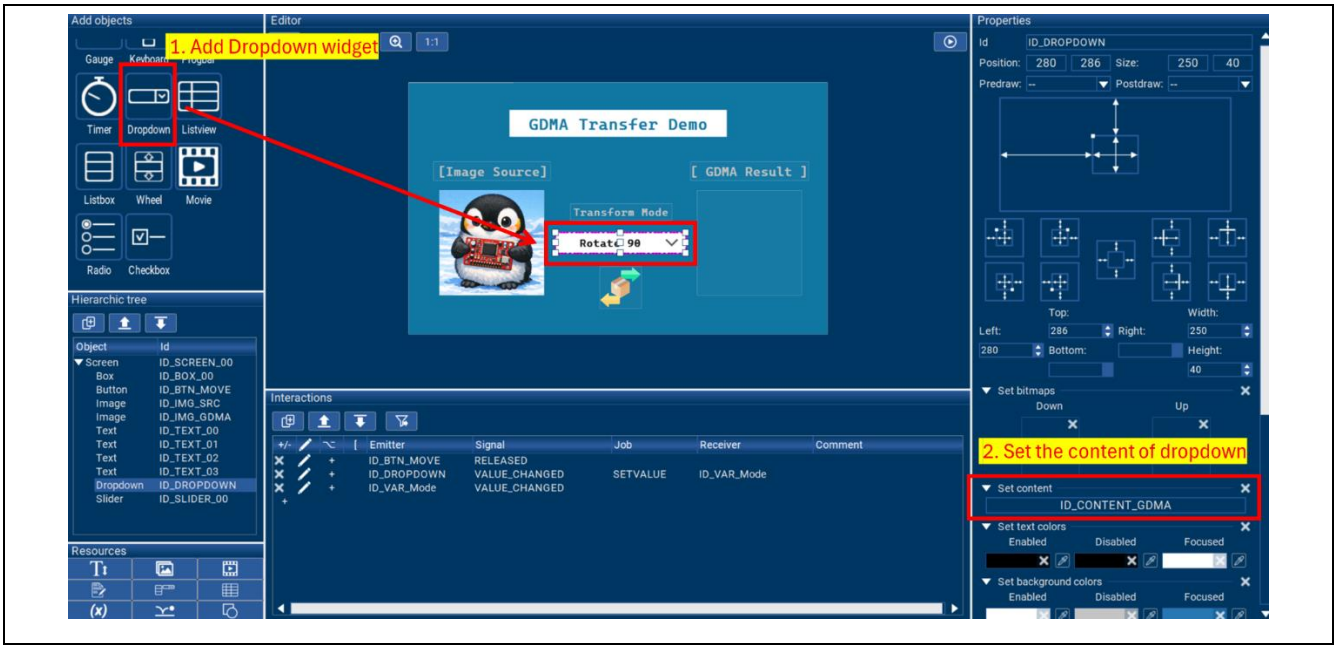


Figure 1-14 Dropdown Widget Settings

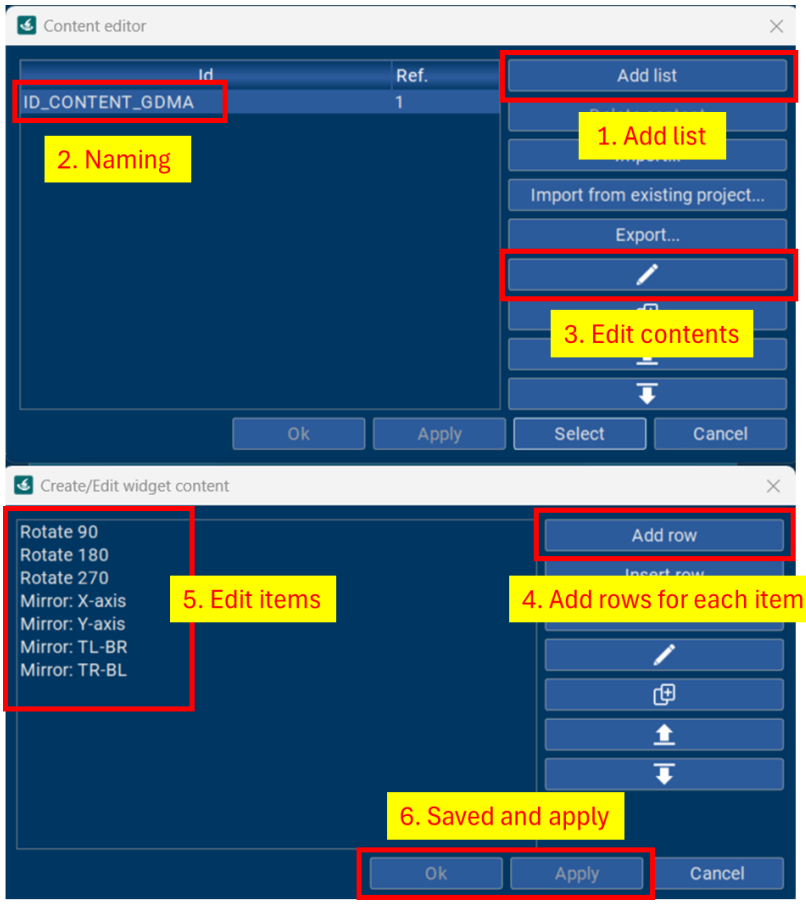


Figure 1-15 Content Settings of Dropdown Widget

After configuring all widgets, add an interaction for the Button widget in the Interaction panel. Set the parameters as “Emitter (ID_BTN_MOVE)”, “Signal (RELEASED)”, “Job (NULL)”, and “Comment (NULL)”. This generates the command function ***ID_SCREEN_00__ID_BTN_MOVE__WM_NOTIFICATION_RELEASED.*** Users can also add custom code in Edit Code (optional).

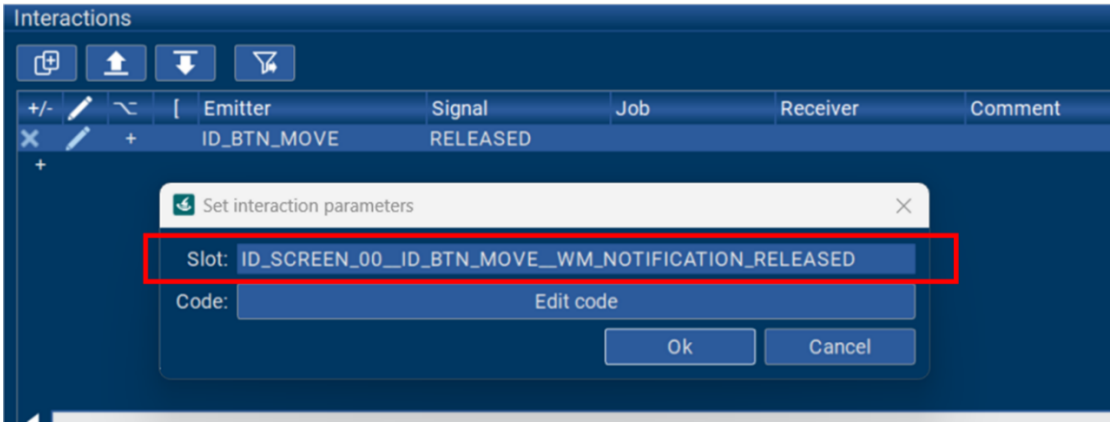


Figure 1-16 Settings of Interaction and Generated Slot

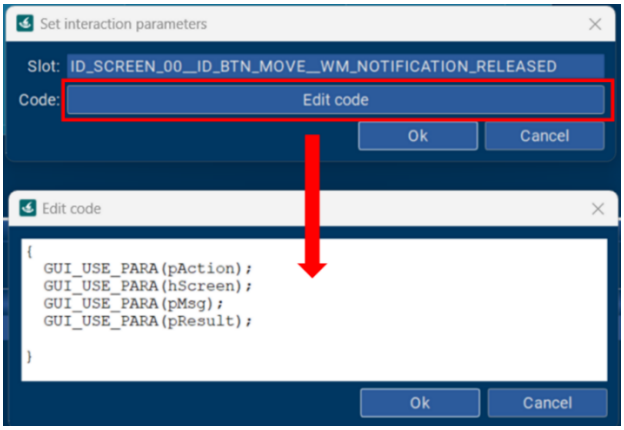


Figure 1-17 Slot Contents

In this example, the call parameters for GDMA processing (*dma350_lib_transform_t*) are changed based on the selected menu item. Therefore, a new variable (ID_VAR_Mode) is added to store the data from the dropdown selection.

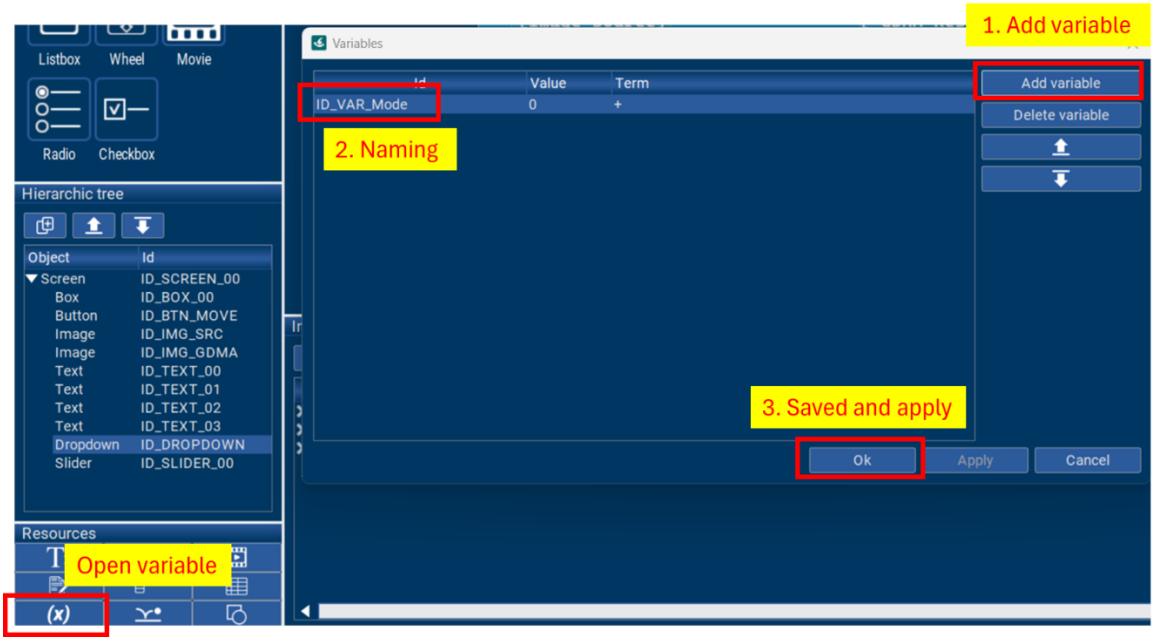


Figure 1-18 Add the Variable

After adding the variable, define the relationship between the Dropdown object and the variable ID_VAR_Mode (as shown in Table 1-2) to record updates of the dropdown state. Then, add code in the "**Edit Code**" section, modify the code so that the program can use the *APPW_GetVarData()* function to read the index value of the Dropdown object (starting from 0) in order to identify the item selected by the user.

Emitter	Signal	Job	Receiver	Emitter
ID_DROPDOWN	VALUE_CHANGED	SETVALUE	ID_VAR_Mode	ID_DROPDOWN
ID_VAR_Mode	VALUE_CHANGED	X	X	ID_VAR_Mode

Table 1-2 Interaction Settings of the Variable

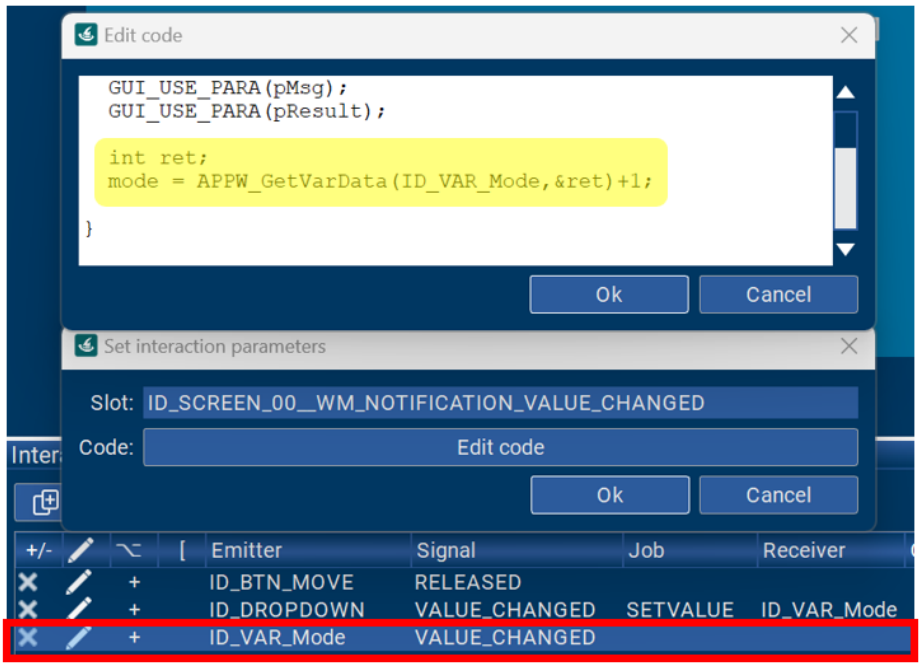


Figure 1-19 Interaction Settings

1.4.3 Export the EmWin AppWizard Project

After configuring all widgets, export the emWin AppWizard project via the toolbar ***File → Export & Save***. Then, copy the exported Resource and Source folders to replace the existing ones in the *Application* folder of your Keil project. (It is recommended to delete the original files before placing the new ones.)

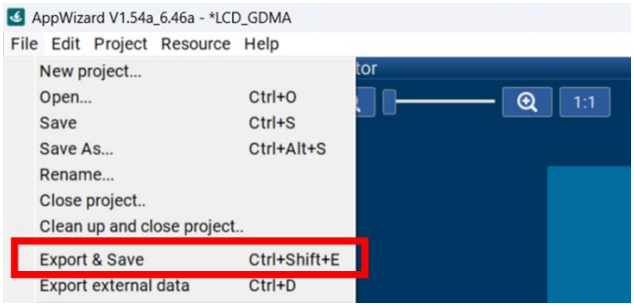


Figure 1-20 Export the GUI Project

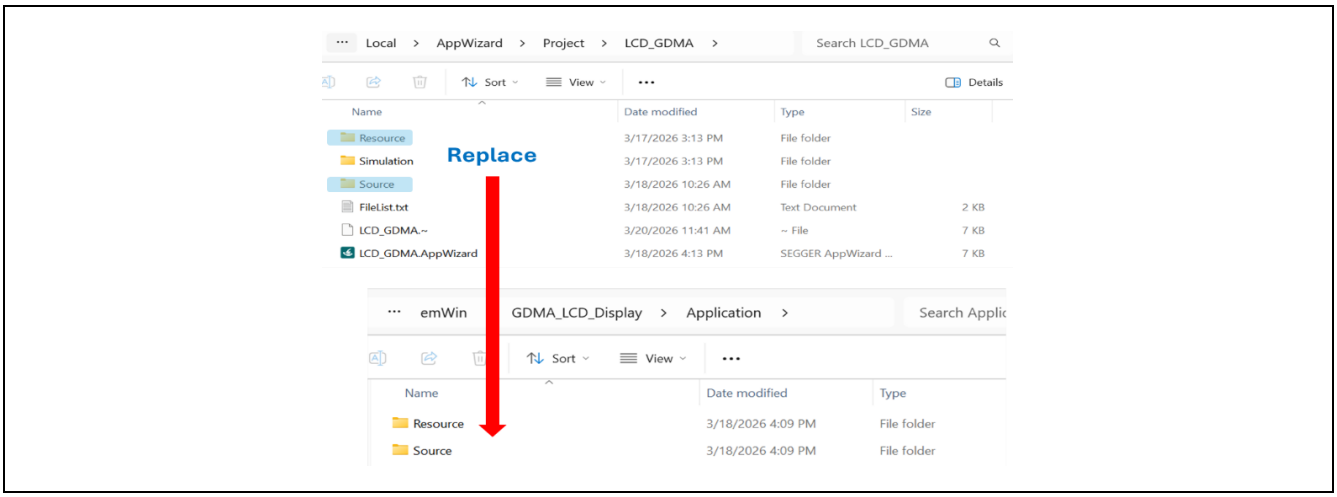


Figure 1-21 Replace Folders to the Keil Project

1.4.4 Download the Application to the Evaluation Board

Open the Keil project and right-click "*Add Existing Files to Group 'Application'*" to add all files generated by AppWizard. Make sure that all file paths are included in the **Includes Path** field, then build the project.

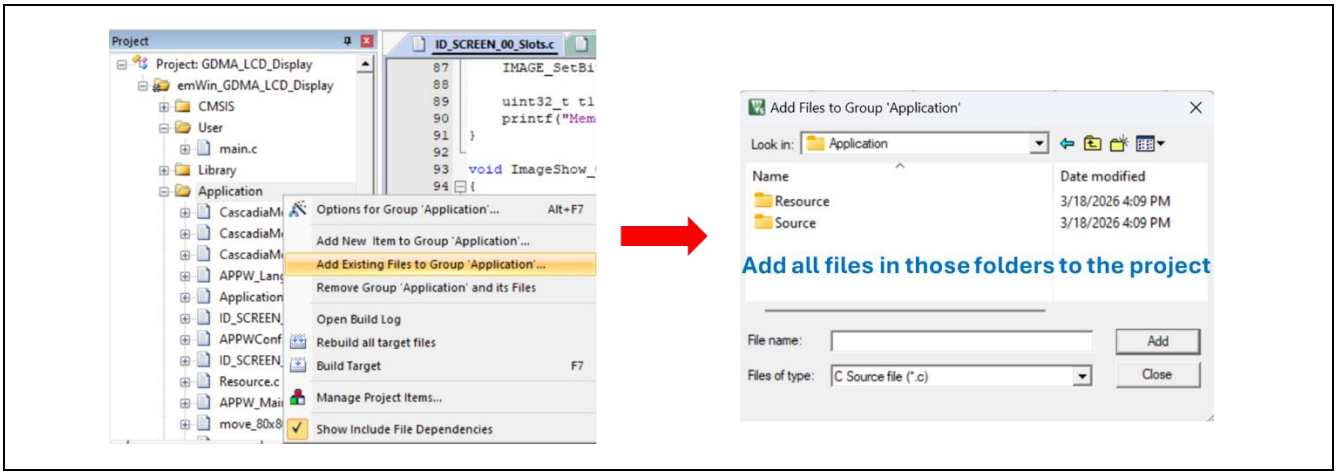


Figure 1-22 Add the Generated Files from AppWizard to the Project

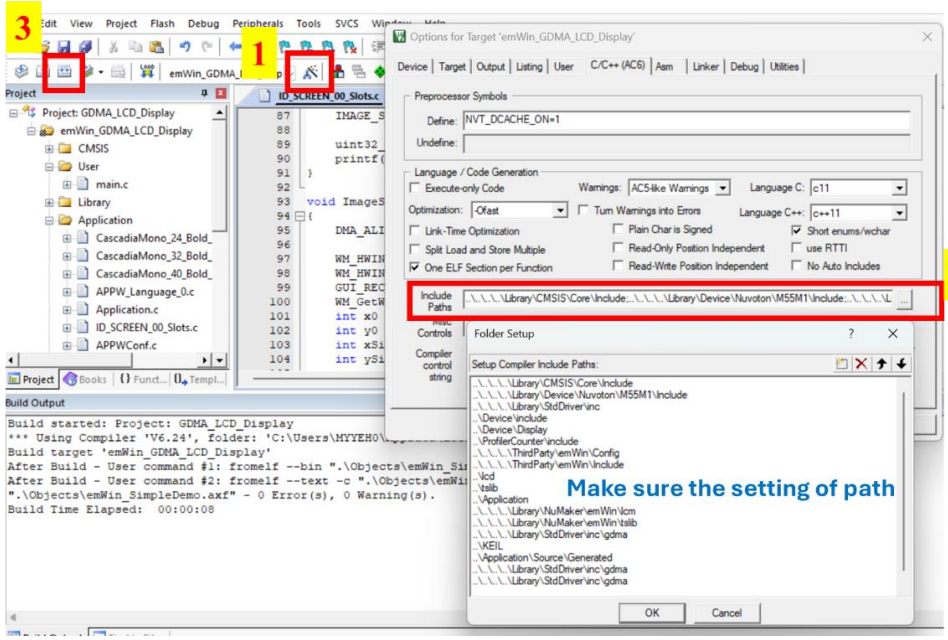


Figure 1-23 Configuration of Files Path

Finally, connect the PC to the M55M1 evaluation board via a Type-C cable. Then load and run the program.

1.4.5 Convert a Keil Project to a VS Code Project

Users requiring a VS Code development environment can convert a Keil project that already contains imported GUI files. First, launch the VS Code IDE, navigate to the "**Explorer**" tab in the sidebar, and open the folder containing the Keil project.

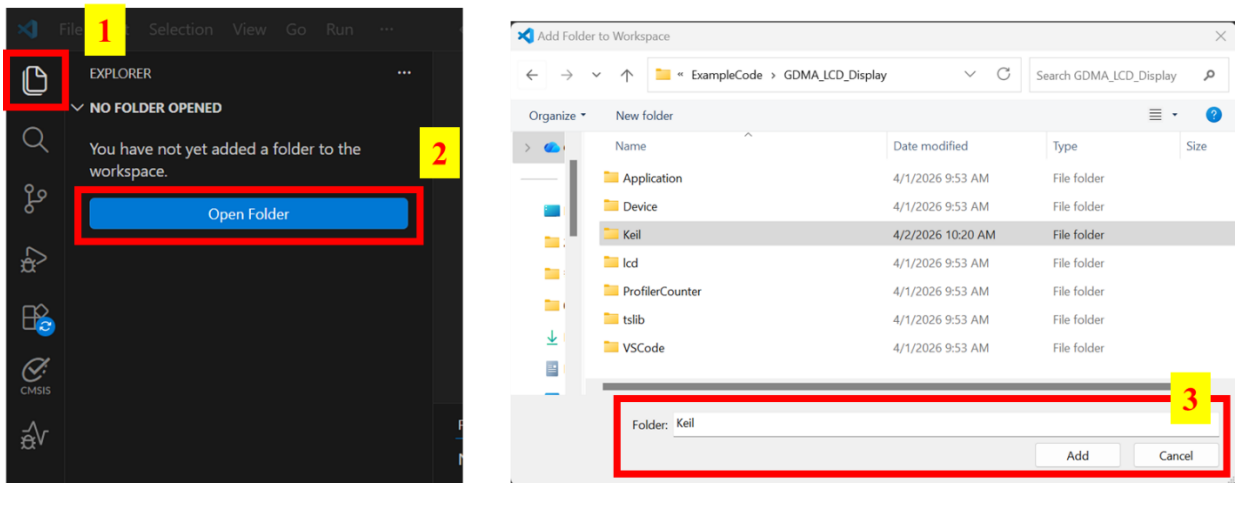


Figure 1-24 Open the Keil Folder in VS Code IDE

After importing the folder, locate the ***.uvprojx*** file of the Keil project. Right-click the file and select "*Convert µVision project to CMSIS solution*" to initiate the conversion.

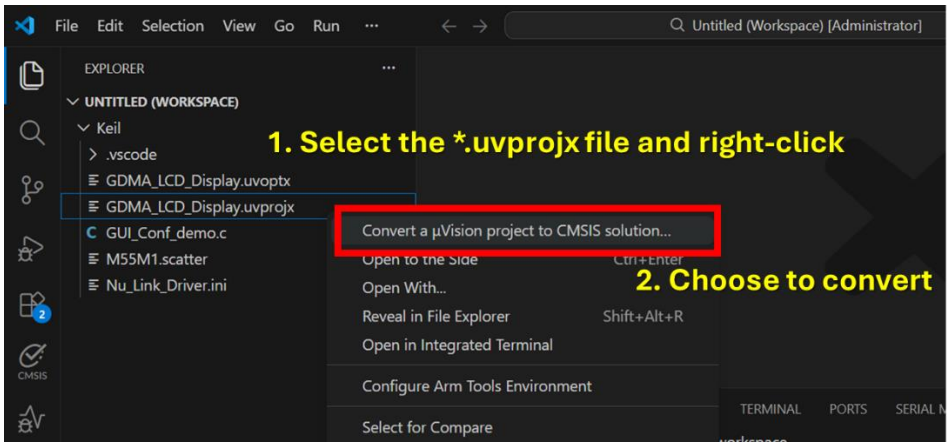


Figure 1-25 Convert Using VS Code's Built-in Features

Upon conversion, VS Code project files—including *launch.json*, *tasks.json*, *vcpkg-configuration.json*, **.cproject.yml*, and **.csolution.yml*, will be generated automatically.

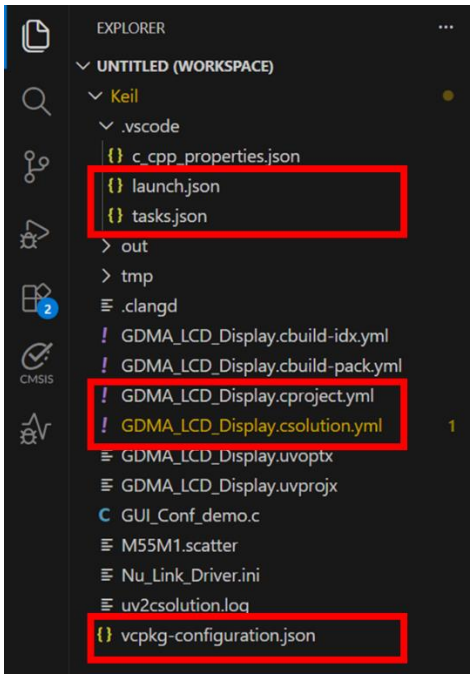


Figure 1-26 Converted VS Code Project Files

1.5 Demo Result

The execution results of this example are shown in Figure 1-27 and Figure 1-28. Click the button in the center of the screen to use GDMA to copy the target image (Image Source), and render the processed result onto the image object according to the user-selected method.



Figure 1-27 Execution Result (GUI Display)

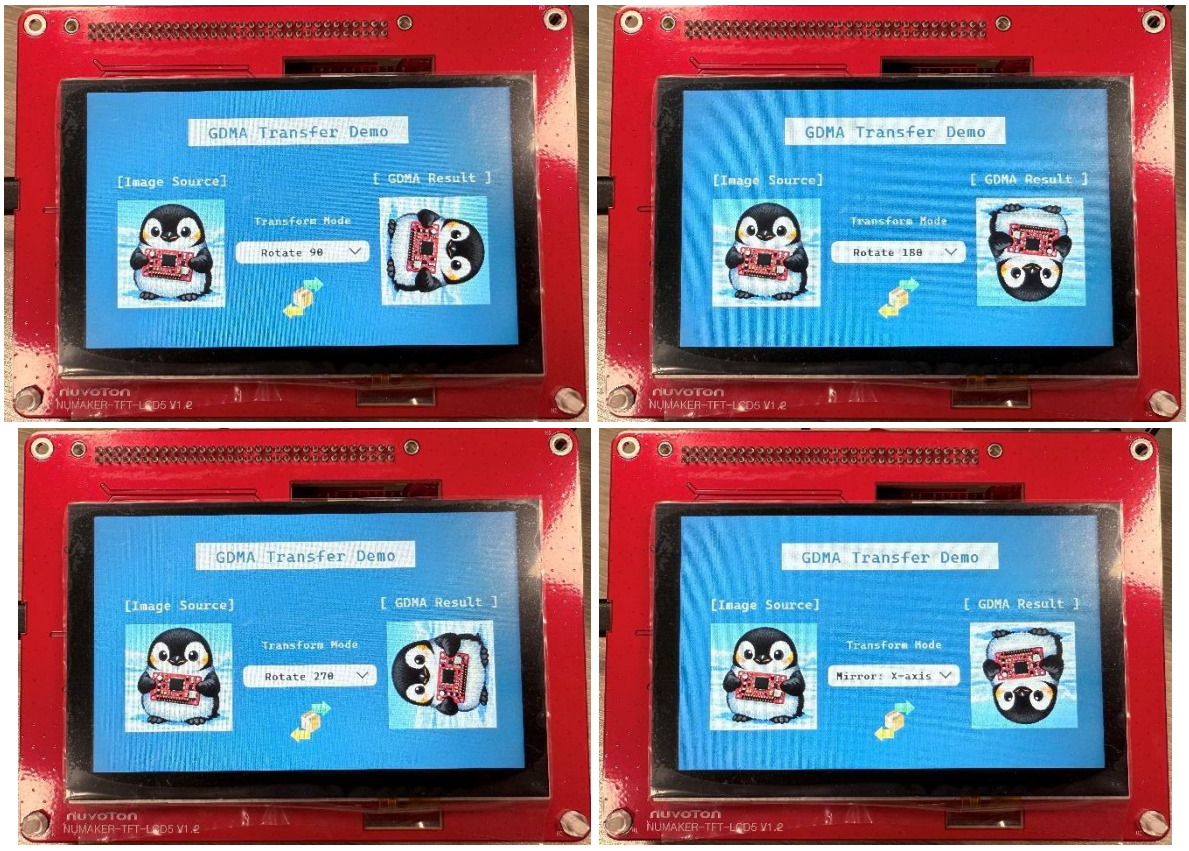




Figure 1-28 Operation Result

2. Code Description

This example uses emWin to design the GUI and imports the generated files into the Keil project. The *main.c* file performs system initialization, including configuring the touch panel, frame buffer, and multiple buffering. It then transfers control to the main application function *MainTask()* to start executing the project. The following code sections are located in *main.c* and *APPW_MainTask.c*, respectively.

```
int main(void)
{
    /* Init System, peripheral clock and multi-function I/O */
    SYS_Init();

    /* Init Debug UART to 115200-8N1 for print message */
    InitDebugUart();

    g_pu8FrameBufPtr  = (U8 *)((U32)g_u8FrameBuf);
    g_enable_Touch = 0;
    //
    // Initialize timer
    //
    tmr_init();
    //
    // NVT_LCM_VERSION defined in lcm.h
    //
#if NVT_LCM_VERSION == 1

    FT5316_low_level_init();
    FT5316_startup_sequence();

#elif NVT_LCM_VERSION == 2
    GUI_Init();
    WM_MULTIBUF_Enable(1);
    Init_TouchPanel();
    #if 0   /* set 1 to do touch calibration */
    ts_calibrate(__DEMO_TS_WIDTH__, __DEMO_TS_HEIGHT__);
    #else   /* set 0 to utilize default touch parameters */
    ts_init();
    #endif
#else
    #error "NVT_LCM_VERSION in lcm.h defined as unknown version!"
#endif
    g_enable_Touch = 1;

    printf("\n\nCPU @ %dHz\n", SystemCoreClock);
    printf("+-----------------------------------------------------+ \n");
    printf("|                  Example Code Demo                  | \n");
    printf("|                 GDMA Transform Ways                 | \n");
    printf("+-----------------------------------------------------+ \n");

    MainTask();
}
```

APPW_MainTask.c

```
#include "Generated/Resource.h"

void MainTask(void) {
  //
  // Setup configuration dependent pointers
  //
  APPW_X_Setup();
  //
  // Initialize AppWizard
  //
  APPW_Init(APPW_PROJECT_PATH);
  //
  // Create all persistent screens except initial screen
  //
  APPW_CreatePersistentScreens();
  //
  // Create initial screen...
  //
  APPW_CreateRoot(APPW_INITIAL_SCREEN, WM_HBKWIN);
  //
  // ...and keep it alive
  //
  for (APPW_Run = 1; APPW_Run; ) {
    APPW_Exec();
    GUI_X_Delay(5);
  }
}
```

Widgets created by emWin AppWizard are automatically assigned buffers within the project to store their respective data. Additionally, a corresponding C file is generated for each screen (Screen Widget) to facilitate coding. Widget-related definitions can be found in *ID_SCREEN_00.c*, where users can look up the specific data definition names used in the program.

For example, in this project, *ID_IMG_SRC* is defined as ***acimageSRC_200x200***. **Note:** The name will vary depending on the imported file from emWin AppWizard.

ID_SCREEN_00.c

```
static APPW_CREATE_ITEM _aCreate[] = {
  { WM_OBJECT_WINDOW_Create,
    ID_SCREEN_00, 0,
    { { { DISPOSE_MODE_REL_PARENT, 0, 0, 0 },
        { DISPOSE_MODE_REL_PARENT, 0, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
      },
      800, 480, 0, 0, 0, 0
    },
    { 0, 0 }
  },
  { WM_OBJECT_BOX_Create,
    ID_BOX_00, ID_SCREEN_00,
    { { { DISPOSE_MODE_REL_PARENT, 0, 0, 0 },
        { DISPOSE_MODE_REL_PARENT, 0, 0, 0 },
```

```
        { DISPOSE_MODE_REL_PARENT, 0, 0, 0 },
        { DISPOSE_MODE_REL_PARENT, 0, 0, 0 },
      },
      0, 0, 0, 0, 0, 0
    },
    { 0, 0 }
  },
  { WM_OBJECT_BUTTON_Create,
    ID_BTN_MOVE, ID_SCREEN_00,
    { { { DISPOSE_MODE_REL_PARENT, 365, 0, 0 },
        { DISPOSE_MODE_REL_PARENT, 351, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
      },
      80, 80, 0, 0, 0, 0
    },
    { 0, 0 }
  },
  { WM_OBJECT_IMAGE_Create,
    ID_IMG_SRC, ID_SCREEN_00,
    { { { DISPOSE_MODE_REL_PARENT, 59, 0, 0 },
        { DISPOSE_MODE_REL_PARENT, 206, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
      },
      200, 200, 0, 0, 0, 0
    },
    { 0, 0 }
  },
  { WM_OBJECT_IMAGE_Create,
    ID_IMG_GDMA, ID_SCREEN_00,
    { { { DISPOSE_MODE_REL_PARENT, 551, 0, 0 },
        { DISPOSE_MODE_REL_PARENT, 206, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
      },
      200, 200, 0, 0, 0, 0
    },
    { 0, 0 }
  },
  { WM_OBJECT_TEXT_Create,
    ID_TEXT_00, ID_SCREEN_00,
    { { { DISPOSE_MODE_REL_PARENT, 193, 0, 0 },
        { DISPOSE_MODE_REL_PARENT, 53, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
      },
      415, 51, 0, 0, 0, 0
    },
    { 0, 0 }
  },
  { WM_OBJECT_TEXT_Create,
    ID_TEXT_01, ID_SCREEN_00,
    { { { DISPOSE_MODE_REL_PARENT, 47, 0, 0 },
        { DISPOSE_MODE_REL_PARENT, 151, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
```

```
        { DISPOSE_MODE_NULL, 0, 0, 0 },
      },
      224, 36, 0, 0, 0, 0
    },
    { 0, 0 }
  },
  { WM_OBJECT_TEXT_Create,
    ID_TEXT_02, ID_SCREEN_00,
    { { { DISPOSE_MODE_REL_PARENT, 538, 0, 0 },
        { DISPOSE_MODE_REL_PARENT, 151, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
      },
      226, 36, 0, 0, 0, 0
    },
    { 0, 0 }
  },
  { WM_OBJECT_TEXT_Create,
    ID_TEXT_03, ID_SCREEN_00,
    { { { DISPOSE_MODE_REL_PARENT, 309, 0, 0 },
        { DISPOSE_MODE_REL_PARENT, 231, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
      },
      194, 32, 0, 0, 0, 0
    },
    { 0, 0 }
  },
  { WM_OBJECT_DROPDOWN_Create,
    ID_DROPDOWN, ID_SCREEN_00,
    { { { DISPOSE_MODE_REL_PARENT, 280, 0, 0 },
        { DISPOSE_MODE_REL_PARENT, 286, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
        { DISPOSE_MODE_NULL, 0, 0, 0 },
      },
      250, 40, 0, 0, 0, 0
    },
    { 0, 0 }
  },
};

/*********************************************************************
*
*       _aSetup
*/
static GUI_CONST_STORAGE APPW_SETUP_ITEM _aSetup[] = {
  { ID_BOX_00,    APPW_SET_PROP_COLOR,        { ARG_V(0xff006c9e) } },
  { ID_BTN_MOVE,  APPW_SET_PROP_SBITMAPS,     { ARG_VP(0, acmove_80x80),
                                                ARG_VP(0, acmove_80x80), } },
  { ID_IMG_SRC,   APPW_SET_PROP_TILE,         { ARG_V(0) } },
  { ID_IMG_SRC,   APPW_SET_PROP_SBITMAP,      { ARG_VP(0, acimageSRC_200x200),
                                                ARG_V(160054), } },
  { ID_IMG_GDMA,  APPW_SET_PROP_TILE,         { ARG_V(0) } },
  { ID_IMG_GDMA,  APPW_SET_PROP_SBITMAP,      { ARG_VP(0, NULL), } },
  { ID_TEXT_00,   APPW_SET_PROP_COLOR,        { ARG_V(0xff006c9e) } },
  { ID_TEXT_00,   APPW_SET_PROP_ALIGNTEXT,    { ARG_V(GUI_ALIGN_HCENTER |
GUI_ALIGN_VCENTER),
```

```
                                                ARG_V(0),
                                                ARG_V(0) } },
  { ID_TEXT_00,   APPW_SET_PROP_TEXTID,       { ARG_V(ID_RTEXT_0) } },
  { ID_TEXT_00,   APPW_SET_PROP_FONT,         { ARG_VPF(0, acCascadiaMono_40_Bold_EXT_AA4,
&APPW__aFont[0]) } },
  { ID_TEXT_00,   APPW_SET_PROP_BKCOLOR,      { ARG_V(GUI_WHITE) } },
  { ID_TEXT_01,   APPW_SET_PROP_COLOR,        { ARG_V(0xffc0c0c0) } },
  { ID_TEXT_01,   APPW_SET_PROP_ALIGNTEXT,    { ARG_V(GUI_ALIGN_HCENTER |
GUI_ALIGN_VCENTER),
                                                ARG_V(0),
                                                ARG_V(0) } },
  { ID_TEXT_01,   APPW_SET_PROP_TEXTID,       { ARG_V(ID_RTEXT_1) } },
  { ID_TEXT_01,   APPW_SET_PROP_FONT,         { ARG_VPF(0, acCascadiaMono_32_Bold_EXT_AA4,
&APPW__aFont[1]) } },
  { ID_TEXT_02,   APPW_SET_PROP_COLOR,        { ARG_V(0xffc0c0c0) } },
  { ID_TEXT_02,   APPW_SET_PROP_ALIGNTEXT,    { ARG_V(GUI_ALIGN_HCENTER |
GUI_ALIGN_VCENTER),
                                                ARG_V(0),
                                                ARG_V(0) } },
  { ID_TEXT_02,   APPW_SET_PROP_TEXTID,       { ARG_V(ID_RTEXT_2) } },
  { ID_TEXT_02,   APPW_SET_PROP_FONT,         { ARG_VPF(0, acCascadiaMono_32_Bold_EXT_AA4,
&APPW__aFont[1]) } },
  { ID_TEXT_03,   APPW_SET_PROP_COLOR,        { ARG_V(0xffc0c0c0) } },
  { ID_TEXT_03,   APPW_SET_PROP_ALIGNTEXT,    { ARG_V(GUI_ALIGN_HCENTER |
GUI_ALIGN_VCENTER),
                                                ARG_V(0),
                                                ARG_V(0) } },
  { ID_TEXT_03,   APPW_SET_PROP_TEXTID,       { ARG_V(ID_RTEXT_3) } },
  { ID_TEXT_03,   APPW_SET_PROP_FONT,         { ARG_VPF(0, acCascadiaMono_26_Bold_EXT_AA4,
&APPW__aFont[2]) } },
  { ID_DROPDOWN,  APPW_SET_PROP_SBITMAPS,     { ARG_VP(0,
acDARK_Angle_Arrow_Down_Gray_24x14),
                                                ARG_VP(0,
acDARK_Angle_Arrow_Up_Gray_24x14), } },
  { ID_DROPDOWN,  APPW_SET_PROP_BKCOLORS,     { ARG_V(GUI_WHITE),
                                                ARG_V(0xffc4c4c4),
                                                ARG_V(0xff2777b3) } },
  { ID_DROPDOWN,  APPW_SET_PROP_COLORS,       { ARG_V(GUI_BLACK),
                                                ARG_V(GUI_BLACK),
                                                ARG_V(GUI_WHITE) } },
  { ID_DROPDOWN,  APPW_SET_PROP_COLOR,        { ARG_V(0xff3c3c3c) } },
  { ID_DROPDOWN,  APPW_SET_PROP_FONT,         { ARG_VPF(0, acCascadiaMono_26_Bold_EXT_AA4,
&APPW__aFont[2]) } },
  { ID_DROPDOWN,  APPW_SET_PROP_CONTENT,      { ARG_P(&ID_CONTENT_GDMA_Data) } },
  { ID_DROPDOWN,  APPW_SET_PROP_ALIGN,        { ARG_V(GUI_ALIGN_HCENTER |
GUI_ALIGN_VCENTER) } },
  { ID_DROPDOWN,  APPW_SET_PROP_RADIUS,       { ARG_V(11) } },
  { ID_DROPDOWN,  APPW_SET_PROP_BORDER,       { ARG_V(15) } },
  { ID_DROPDOWN,  APPW_SET_PROP_SIZE,         { ARG_V(100) } },

};
```

Images used in emWin AppWizard are automatically converted into C files upon export (e.g., imageSRC_200x200.c in this example). To prevent data misalignment—which can cause image shifting during GDMA processing—you must add the data alignment definition here: ***#define EMWIN_SUFFIX_DTA DMA.***

```
#include "AppWizard.h"

#define EMWIN_SUFFIX_DTA   __attribute__((aligned(4)))   //word-alignment for DMA

#ifndef EMWIN_PREFIX_DTA
  #define EMWIN_PREFIX_DTA
#endif

#ifndef EMWIN_SUFFIX_DTA
  #define EMWIN_SUFFIX_DTA
#endif
EMWIN_PREFIX_DTA GUI_CONST_STORAGE unsigned char acimageSRC_200x200[80016UL + 1]
EMWIN_SUFFIX_DTA = {
…
…
};
```

The BSP provided by Nuvoton includes the GDMA library. In this example, the function ***dma350_draw_from_canvas()*** is used to perform data transfer. Users can include the required headers at the beginning of the program with ***#include "dma350_lib.h"*** and ***#include "gdma.h"***, and refer to *dma350_lib.h* for additional documentation on related **library** functions.

dma350_lib.h

```
/* DMA350 DMA Library transformation enumeration type */
enum dma350_lib_transform_t
{
    DMA350_LIB_TRANSFORM_NONE = 0,
    DMA350_LIB_TRANSFORM_ROTATE_90,   /*!< Rotate 90 degrees clockwise */
    DMA350_LIB_TRANSFORM_ROTATE_180,  /*!< Rotate 180 degrees clockwise */
    DMA350_LIB_TRANSFORM_ROTATE_270,  /*!< Rotate 270 degrees clockwise */
    DMA350_LIB_TRANSFORM_MIRROR_VER,  /*!< Mirror along X axis. Top to bottom */
    DMA350_LIB_TRANSFORM_MIRROR_HOR,  /*!< Mirror along Y axis. Left to right */
    DMA350_LIB_TRANSFORM_MIRROR_TLBR, /*!< Mirror along diagonal.TopLeft to BottomRight */
    DMA350_LIB_TRANSFORM_MIRROR_TRBL, /*!< Mirror along diagonal.TopRight to BottomLeft */
};

…

/**
 * \brief 2D Copy from canvas (area within a source bitmap) to within a
 *        destination bitmap, while applying various possible transformations.
 *        If the destination size is larger than the source, the source image
 *        will be wrapped (repeated).
 *
 * \param[in] dev             DMA350 channel device struct \ref dma350_ch_dev_t
 * \param[in] src             Source address, top left corner
 * \param[in] des             Destination address, top left corner
 * \param[in] src_width       Source width
 * \param[in] src_height      Source height
 * \param[in] src_line_width  Source line width
 * \param[in] des_width       Destination width
 * \param[in] des_height      Destination height
 * \param[in] des_line_width  Destination line width
 * \param[in] pixelsize       Size of a pixel as in \ref dma350_ch_transize_t
 * \param[in] transform       Transform type as in \ref dma350_lib_transform_t
 * \param[in] exec_type       Execution type as in \ref dma350_lib_exec_type_t
 *
 * \return Result of the operation \ref dma350_lib_error_t
 *
```

```
 * \note Destination width and height denote the area which will be filled at
 *       the destination address. The copy always starts from the top left
 *       corner of the source. If the requested destination size does not match
 *       the source, the resulting image will be repeated / cropped.
 */
enum dma350_lib_error_t dma350_draw_from_canvas(struct dma350_ch_dev_t *dev,
                                                const void *src, void *des,
                                                uint32_t src_width, uint16_t src_height,
                                                uint16_t src_line_width,
                                                uint32_t des_width, uint16_t des_height,
                                                uint16_t des_line_width,
                                                enum dma350_ch_transize_t pixelsize,
                                                enum dma350_lib_transform_t transform,
                                                enum dma350_lib_exec_type_t exec_type);
```

All operations in this example are implemented in *ID_SCREEN_00_slots.c*, where the interaction callback function generated by emWin AppWizard (*ID_SCREEN_00__ID_BTN_MOVE__WM_NOTIFICATION_RELEASED*) is also defined. Users can add custom code in this file.

To use data as the appearance of a widget, a buffer with the GUI_BITMAP format must be defined. The structure definition can be found in *GUI_Type.h*.

In this example, a buffer of type unsigned short is declared and assigned to an Image Widget. The image content is refreshed by updating the buffer data.

Note: The *IMAGE_SetBitmap()* function must be called to redraw the display after each update.

ID_SCREEN_00_slots.c

```
#include "dma350_lib.h"
#include "gdma.h"

#define IMG_W 200                                                    //image width
#define IMG_H 200                                                    //image height
#define DMA_ALIGNMENT   __attribute__((aligned(DCACHE_LINE_SIZE))) // alignment

int mode = 1;                 //default mode of GDMA

DMA_ALIGNMENT static unsigned short GDMA_Buffer01[ DCACHE_ALIGN_LINE_SIZE(200 * 200) ];
static GUI_BITMAP GDMA_IMG_200x200 = {
             200,
             200,
             200*2,
             16,
             (const unsigned char*)GDMA_Buffer01,
          NULL,
             GUI_DRAW_BMPM565
};

// customer's code
void ImageShow_GDMA(WM_HWIN hScreen, const unsigned char *src, int mode)
{
    DMA_ALIGNMENT const uint8_t *image = src + 16;   // skip header

    WM_HWIN ImgWidget1 = WM_GetDialogItem(hScreen, ID_IMG_GDMA);
```

```
    GUI_RECT ImgRect;
    WM_GetWindowRectEx(ImgWidget1, &ImgRect);
      int x0    = ImgRect.x0;
    int y0    = ImgRect.y0;
    int xSize = ImgRect.x1 - ImgRect.x0 + 1;
    int ySize = ImgRect.y1 - ImgRect.y0 + 1;

    // GDMA transformation
    dma350_draw_from_canvas(
        GDMA_CH_DEV_S[0],
        (const void *)image,
        GDMA_Buffer01,            // destination line start,
        xSize, ySize, xSize,      // width, height, source stride(line width of image)
        xSize, ySize, xSize,      // dest width, height, destination stride(line width of
buffer)
        DMA350_CH_TRANSIZE_16BITS,
        mode,          // transformation enumeration type, refer to dma350_lib_transform_t
        DMA350_LIB_EXEC_BLOCKING);// Assign a GUI_BITMAP to the Image widget for display
on redraw

#if (NVT_DCACHE_ON == 1)
    // Cache maintenance: Invalidate the DMA buffer to ensure CPU reads the latest data
before redrawing.
     SCB_InvalidateDCache_by_Addr (GDMA_Buffer01, xSize*ySize*2 );
#endif

    IMAGE_SetBitmap(ImgWidget1, &GDMA_IMG_200x200);
}

…

// Interaction function
void ID_SCREEN_00__WM_NOTIFICATION_VALUE_CHANGED(APPW_ACTION_ITEM * pAction, WM_HWIN
hScreen, WM_MESSAGE * pMsg, int * pResult) {
  GUI_USE_PARA(pAction);
  GUI_USE_PARA(hScreen);
  GUI_USE_PARA(pMsg);
  GUI_USE_PARA(pResult);

  int ret;
  mode = APPW_GetVarData(ID_VAR_Mode,&ret)+1;
}
```

GUI_Type.h

```
typedef struct {
  I16 xSize;
  I16 ySize;
} GUI_SIZE;

typedef struct {
  void      (* pfDraw)  (int x0,
                         int y0,
                         int xsize,
                         int ysize,
```

```
                          const U8 * pPixel,
                          const LCD_LOGPALETTE * pLogPal,
                          int xMag,
                          int yMag);
                          GUI_COLOR (* pfIndex2Color)(LCD_PIXELINDEX Index);
                          void      (* pfDrawHW)(int x0,
                          int y0,
                          int xsize,
                          int ysize,
                          const U8 * pPixel,
                          const LCD_LOGPALETTE * pLogPal,
                          int xMag,
                          int yMag);
                          const LCD_API_COLOR_CONV * pColorConvAPI;
} GUI_BITMAP_METHODS;

typedef struct {
  U16P XSize;
  U16P YSize;
  U16P BytesPerLine;
  U16P BitsPerPixel;
  const U8 * pData;
  const GUI_LOGPALETTE * pPal;
  const GUI_BITMAP_METHODS * pMethods;
} GUI_BITMAP;
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M55M1_Series_BSP_CMSIS_V3.01.003
- IDE version
  - Keil uVision 5.43
  - Visual Studio Code V1.112.0
    - CMSIS Solution extension
    - Arm Tools Environment Manager
    - Arm Compiler (ARMClang 6.x)
- AppWizard version
  - AppWizard_V154a_646a

3.2 Hardware Requirements

- Circuit components
  - NuMaker-X-M55M1
- Connection Diagram
  - Connect the NuMaker-X-M55M1 to the PC via Type-C USB.

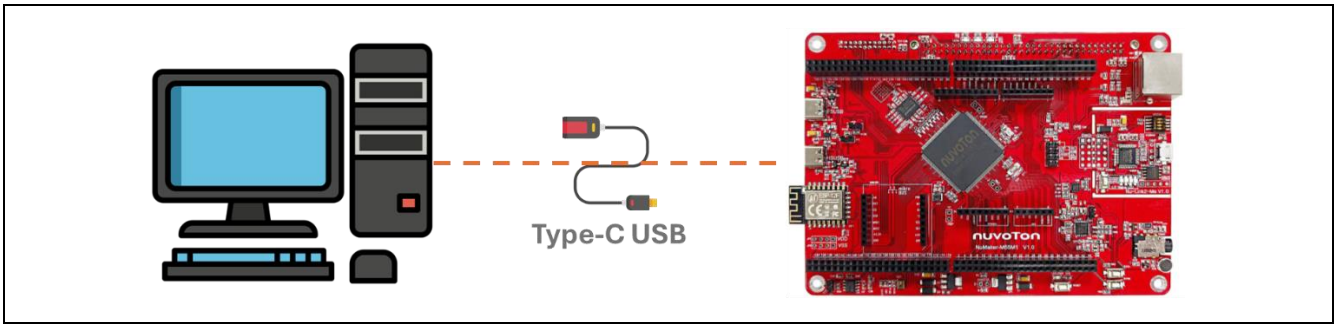


Figure 3-1 Connection Diagram

4. Directory Information

The directory structure is shown below.

EC_M55M1_GDMA_LCD_Display_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
NuMaker	
emWin	Libraries of touch screen
StdDriver	All peripheral driver header and source files
ThirdParty	
emWin	EmWin libraries and drivers
LCD_GDMA	GUI project of AppWizard. Please open it by *AppWizard_V154a_646a*.
ExampleCode	
GDMA_LCD_Display	
Keil	Source code of Keil project
VSCode	Source code of VS code project

Figure 4-1 Directory Structure

5. Example Code Execution

5.1 Keil MDK

1. Browse the sample code folder as described in the Directory Information section and double-click *GDMA_LCD_Display.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

5.2 VS Code

1. Open VS Code IDE and import the VSCode folder of this example package.

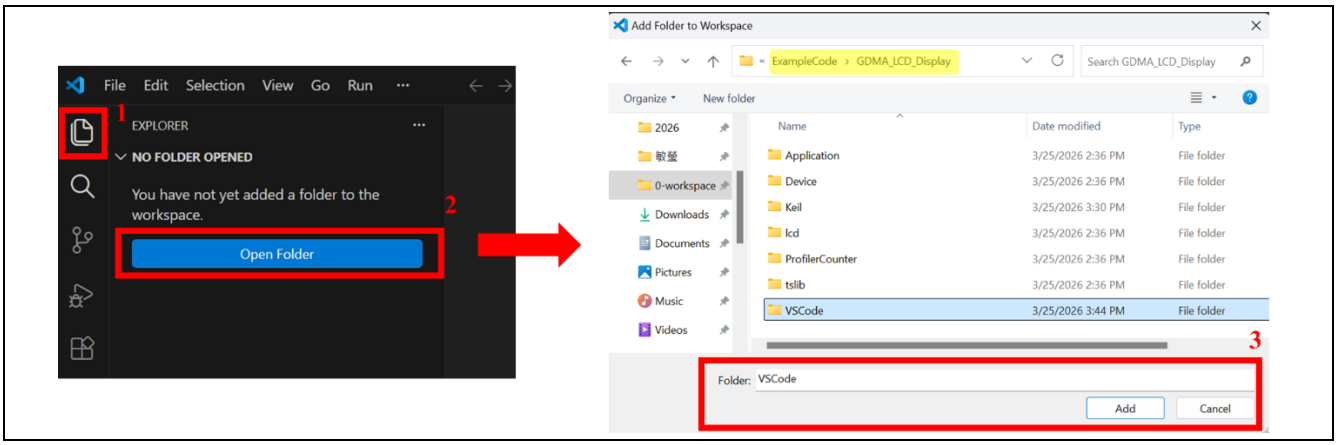


Figure 5-1 Open the “VSCode” Folder

2. Navigate to the "***CMSIS***" tab, then click "***Build Solution***" followed by "***Load and Run Application***." Once the process is complete, the execution results will be displayed on the LCD panel.

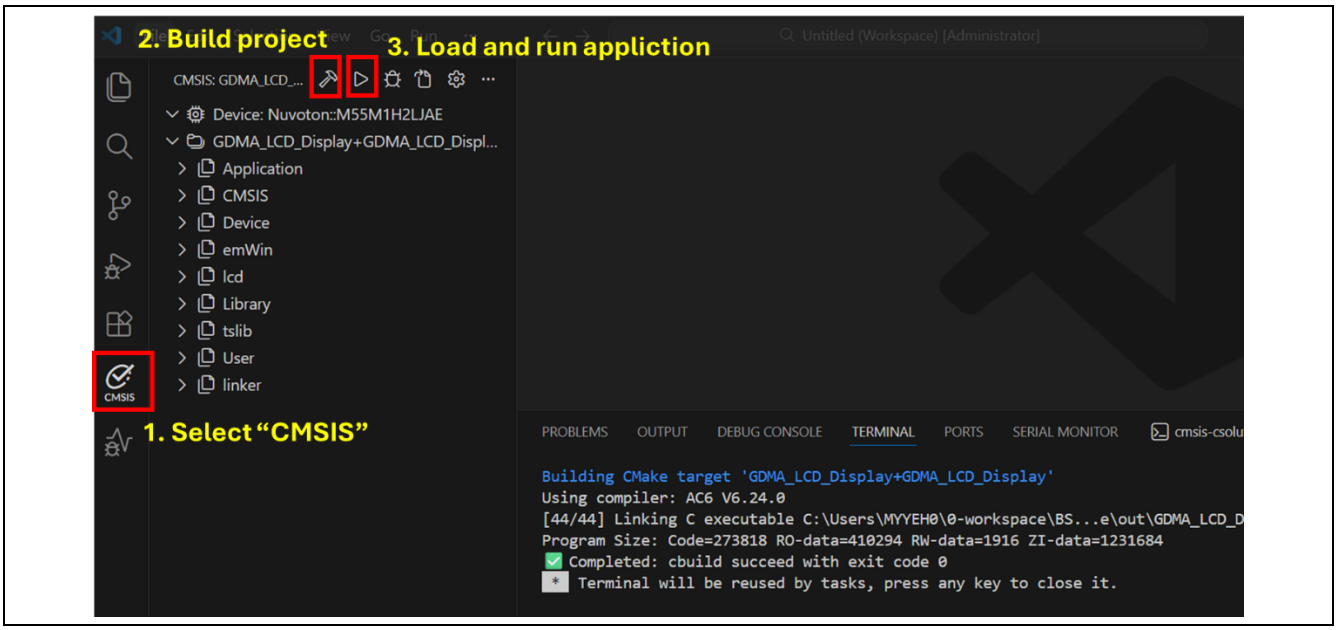


Figure 5-2 Build and Download the Project to the Evaluation Board

6. Revision History

Date	Revision	Description
2026.06.08	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.