Designing and Running an LVGL GUI with SquareLine Studio

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	Use SquareLine Studio to design an LVGL GUI, then integrate the generated UI files into a Nuvoton MCU project for compilation and execution.
BSP Version	M55M1_Series_BSP_CMSIS_V3.01.005
Hardware	NuMaker-X-M55M1D

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 program and document explain how to use SquareLine Studio to design an LVGL GUI, navigate the basic screen-design functions in SquareLine Studio, and integrate the generated UI files into a Nuvoton MCU project for execution.

This example uses the NuMicro M55M1 platform as the reference platform. It can also be applied to other platforms listed in Table 1-1.

1.1 Principle

1.1.1 SquareLine Studio Introduction

SquareLine Studio, hereinafter referred to as SLS, is a WYSIWYG GUI design tool built specifically for LVGL. Designers can drag and drop widgets such as buttons, labels, images, and sliders onto the canvas, then adjust layout, style, and events in the property panel. SLS automatically generates the corresponding LVGL C source code, reducing the effort required to manually write UI code.

The LVGL Library 9.2 version in SLS includes built-in options for several Nuvoton development boards, as shown in Figure 1-1. Users can directly select the corresponding platform to create a project. Together with the "lv_port_nuvoton" BSP, the exported UI files can be integrated into the project and compiled and executed on a Nuvoton development board, enabling a seamless workflow between screen design and firmware development.

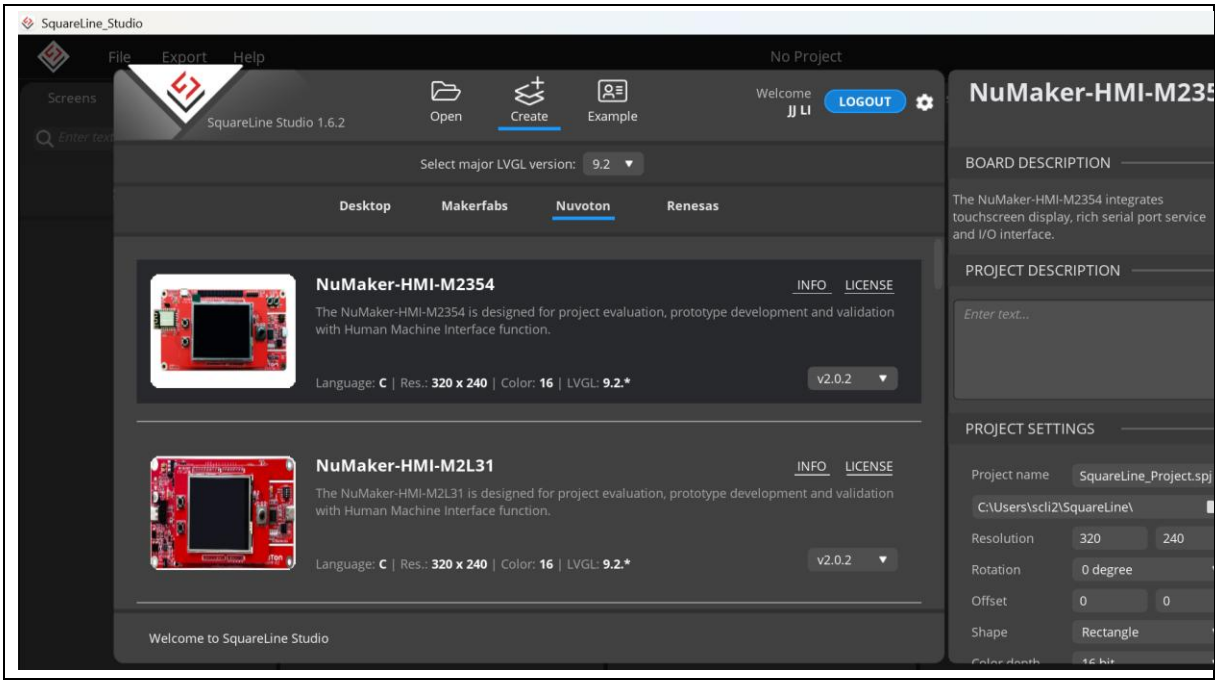


Figure 1-1 SquareLine Studio LVGL 9.2 Version

The platforms currently ported and supported in the "lv_port_nuvoton" BSP are listed in Table 1-1. LVGL sample programs have also been ported for the corresponding development boards.

Supported platforms may change depending on the LVGL version update; refer to the official GitHub documentation for details.

NuMicro Series	NuMaker Platforms
M2354 Series	NuMaker-HMI-M2354
M2L31 Series	NuMaker-HMI-M2L31
M3334 Series	NuMaker-HMI-M3334
M3351 Series	NuMaker-HMI-M3351
M487 Series	NuMaker-HMI-M487
M467 Series	NuMaker-HMI-M467
M55M1 Series	NuMaker-X-M55M1D
M5531 Series	NuMaker-HMI-M5531
N9H Series	NuMaker-HMI-N9H20 NuMaker-HMI-N9H26 NuMaker-HMI-N9H30 NuMaker-HMI-N9H31
NUC980 Series	NuMaker-IIoT-NUC980G1D NuMaker-IIoT-NUC980G2D
MA35D1 Series	NuMaker-HMI-MA35D1
MA35H0 Series	NuMaker-HMI-MA35H0

Table 1-1 Supported Platform List

1.1.2 Execution Steps

1. Download and register SquareLine Studio. Refer to the official SLS download page.
2. Create an SLS project:

1) After opening SLS, click Create and select 9.2 under Select major LVGL version, as shown in Figure 1-2.

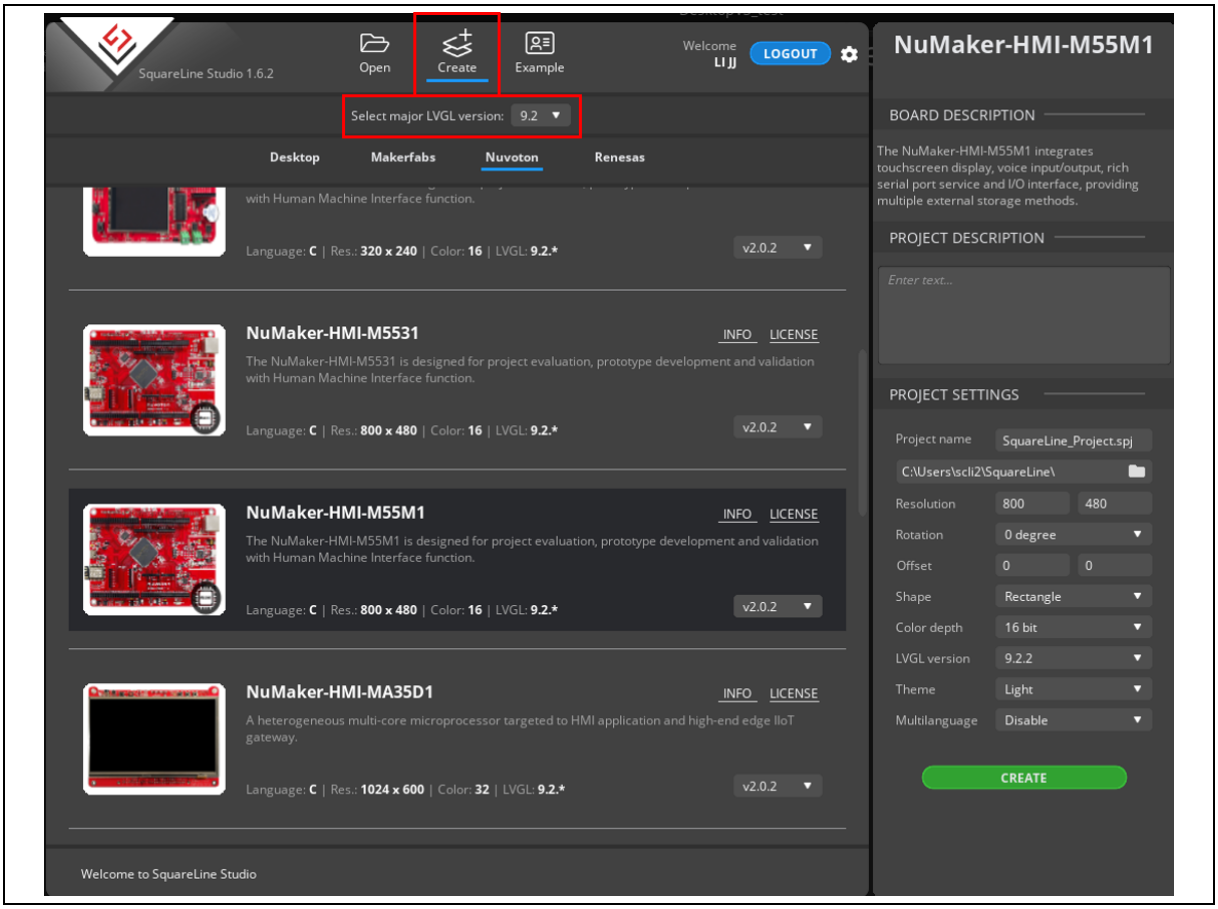


Figure 1-2 Create an SLS Project

2) If the corresponding platform and development environment are available in the menu, select the matching option directly, as shown in Figure 1-2. For example, for M55M1, select NuMaker-HMI-M55M1. The Project Settings panel on the right will already be configured according to the corresponding platform specifications.

   If the corresponding platform and development environment cannot be found in the menu, select another development board or a Desktop version first, and then configure the corresponding project properties in the Project Settings panel. For the project property settings and related notes, refer to Table 1-2.

Properties	Notes
Project Name/Path	The name and path must not contain spaces or non-English characters.
Resolution	The resolution must match the actual LCD resolution; otherwise, the screen aspect ratio will be incorrect.
Rotation	The rotation angle is usually set to 0. Adjust it only if the physical panel orientation is different.

Properties	Notes
Offset	Keep the offset setting at 0.
Shape	The panel shape can be configured as rectangular or circular.
Color depth	Set the color depth to 16-bit or 32-bit depending on the project requirements.
LVGL version	Use the synchronized version described in the Nuvoton GitHub documentation. Backward compatibility is supported.
Theme	The theme can be selected as dark or light; this only affects the default appearance.

Table 1-2 Project Settings

3. SLS GUI interface navigation:

   The SLS interface is shown in Figure 1-3. Refer to Table 1-3 for the description of each function. For advanced interface settings and features, refer to the official SLS documentation.

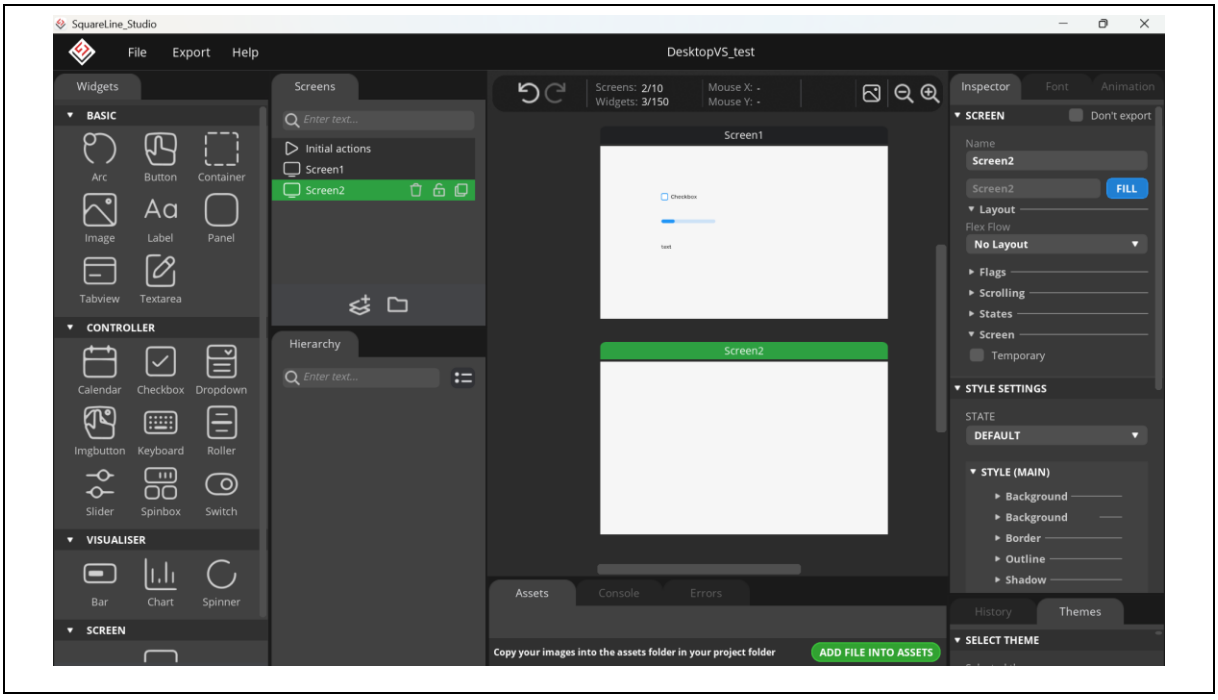


Figure 1-3 SLS Interface

Name	Purpose
Widgets	Widget library. Widgets can be clicked or dragged directly onto the canvas.
Screens	Manages the design of multiple screens.
Canvas	WYSIWYG area for actual placement and layout.

Name	Purpose
Hierarchy	Displays the hierarchical relationship of all widgets on the screen in a tree structure.
Inspector	Configures properties of the selected widget, including position, size, style, and events.
Animations	Adds events or animations to widgets.

Table 1-3 SLS Interface Function Description

4. SLS UI screen design: implement a simple dashboard function

   Follow the steps below to complete the simple dashboard screen design shown in Figure 1-4. This example includes two screens: Screen1 displays a dynamic value and an Arc linkage effect, while Screen2 uses a Slider to control the displayed value.

   Screen1: Dynamic value and Arc display

   On Screen1, the value changes between 0 and 100 and synchronously drives the Arc display position or progress. A Button is also placed on the screen to switch to Screen2.

   Screen2: Slider value control

   On Screen2, the Slider can be dragged to adjust a value range from 0 to 1000. The number displayed on the screen is updated in real time according to the current Slider value.

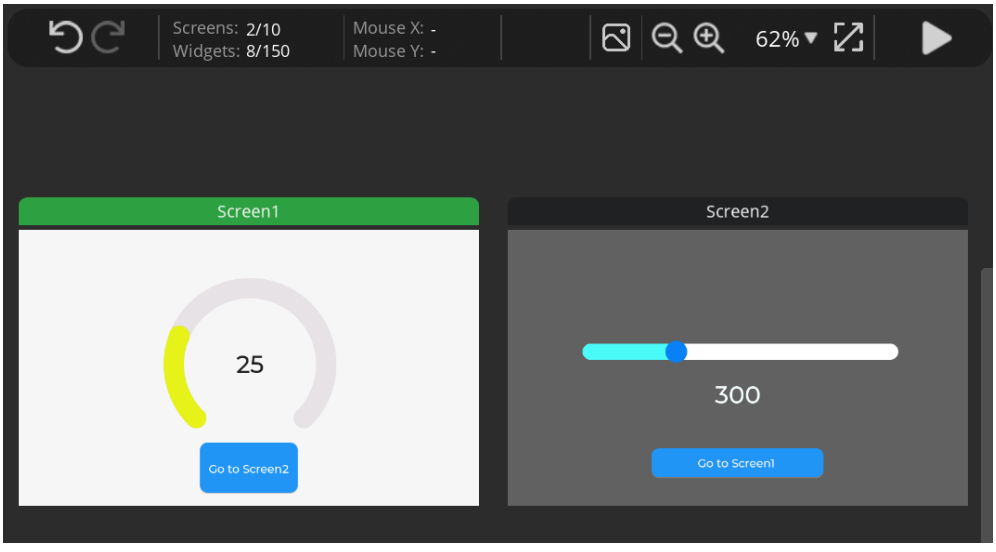


Figure 1-4 Dashboard Implementation

1) Add Screen2: As shown in Figure 1-5, add Screen2 in the Screen window, and set a different background color in STYLE SETTINGS in the Inspector.

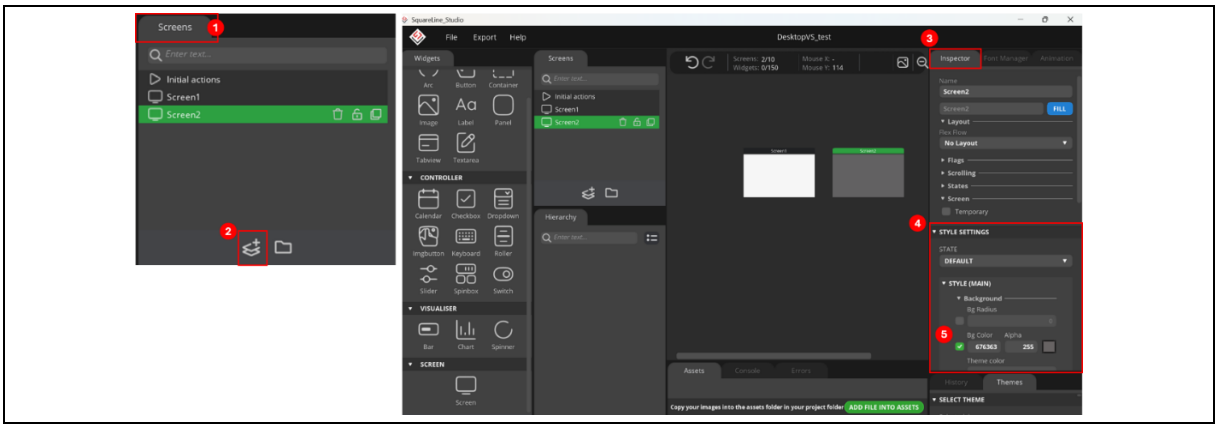


Figure 1-5 Add Screen2 and Set the Background Color

2) Add an Arc widget: Select Screen1 to add objects to Screen1. As shown in Figure 1-6, click Arc from Widgets. After it is selected, the Arc appears on the Screen1 canvas. Then configure the Arc properties in the Inspector according to Table 1-4, including size, value range, display angle, and color settings.

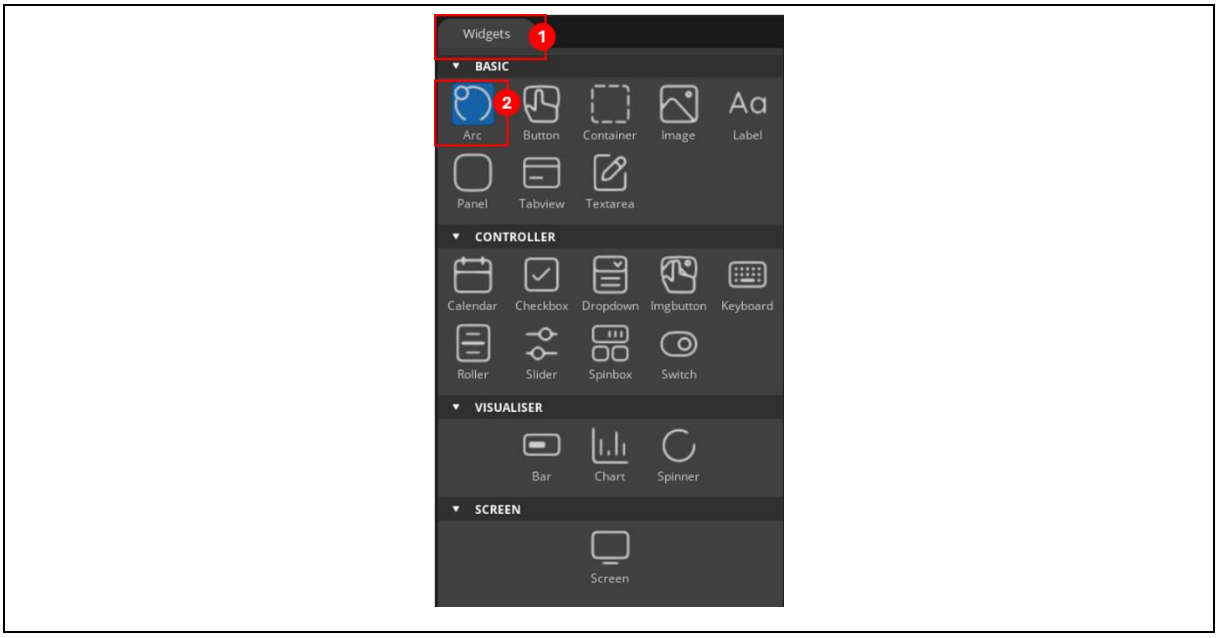
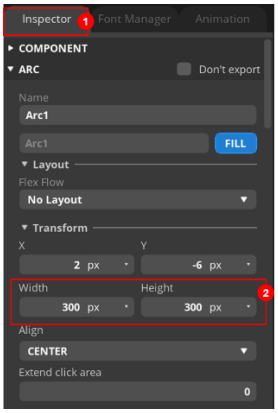
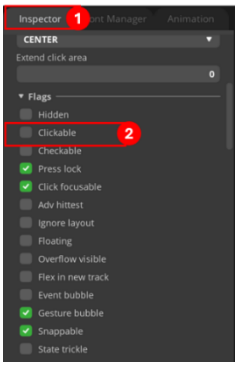


Figure 1-6 Add Widgets – Arc

Contents	Functions	Settings
Width and Height	The overall width and height of the Arc.	
Flags-Clickable	Sets whether the widget can be touched by the user.	
Range, Value, Angle	Range defines the Arc value range, Value defines the initial displayed value, and Angle defines the display angle of the Arc.	

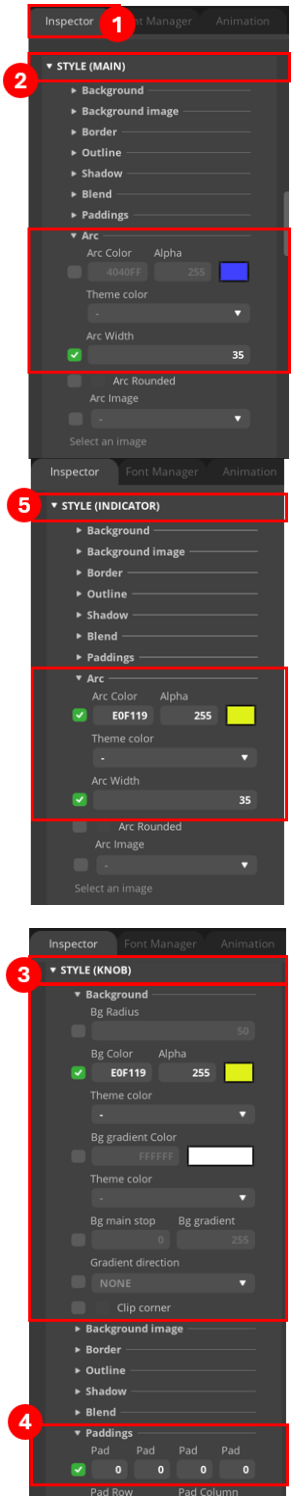
Contents	Functions	Settings
STYLE	Style is divided into three parts: MAIN controls the main Arc body settings, KNOB controls the knob design settings, and INDICATOR controls the indicator design when the Arc value changes.	

Table 1-4 Arc Property Settings

3) Add the value display on Screen1: Select Label from Widgets. After it is selected, the Label will be added to the Screen1 canvas. Drag it to the center of the Arc on the canvas, and configure the Label properties according to Table 1-5.

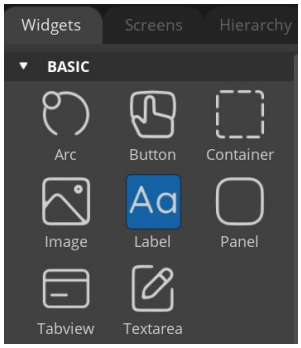


Figure 1-7 Add Widgets – Label

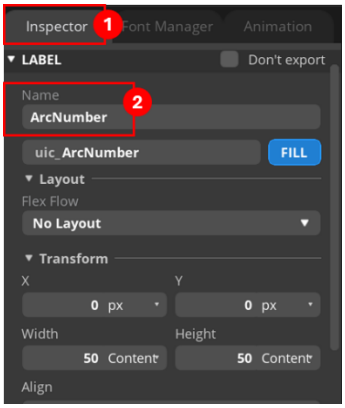
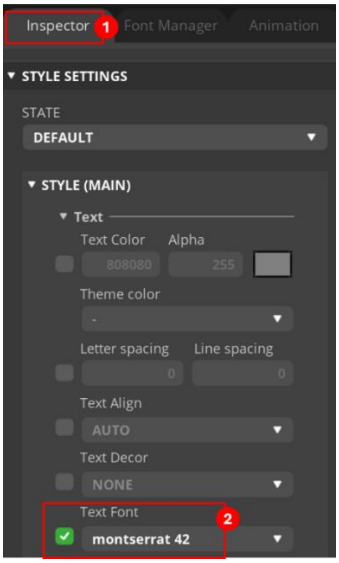
Contents	Functions	Settings
Name	The object name can be customized to make it easier to identify the object when linking it to other objects or using it in the project.	
STYLE-Text Font	The Label color and font can be configured in Style. The font settings include the font size.	

Table 1-5 Label Property Settings

4) Add a Button on Screen1 to switch to Screen2: As shown in Figure 1-8, select Button from Widgets and rename the Button to “NextBTN”. If text needs to be displayed on the Button, add a separate Label and place it on top of the Button.

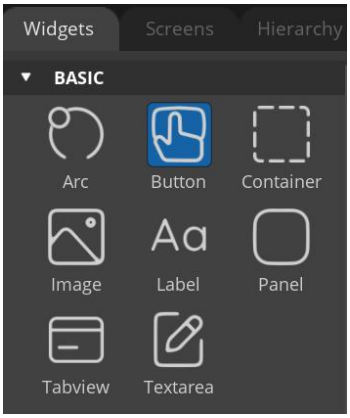


Figure 1-8 Add Widgets – Button

5) Set Screen1 to switch to Screen2 when “NextBTN” is pressed: As shown in Figure 1-9, select “NextBTN”, click ADD EVENT in the Inspector property settings, set Trigger to Clicked, and set Action to Change Screen. This indicates that the displayed action is a screen switch. After adding the action, select the target screen as Screen2 and set the transition effect to Move left.

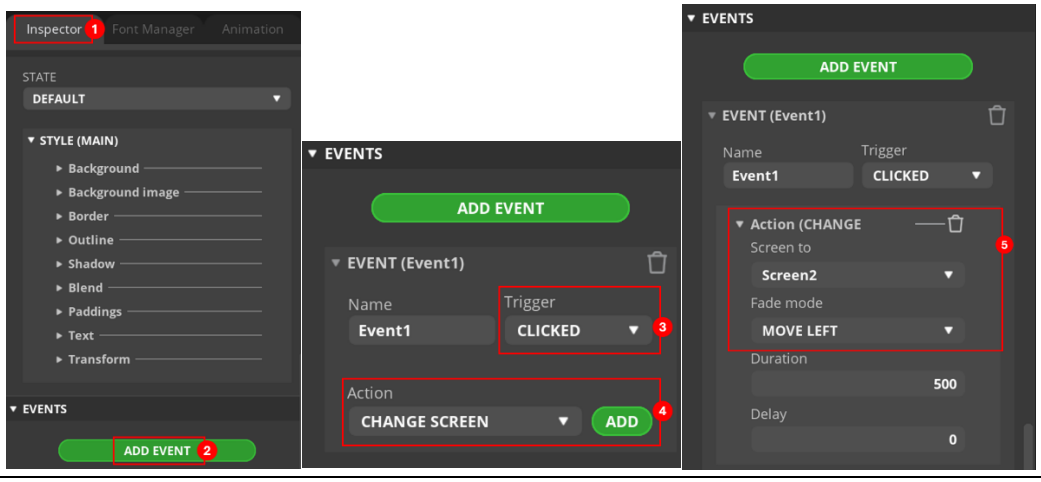


Figure 1-9 Button Action Settings

6) Add widgets to Screen2: Select Screen2. As shown in Figure 1-10, select Slider from Widgets, and configure the Slider properties according to Table 1-6.

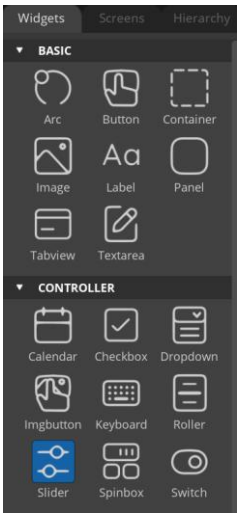


Figure 1-10 Add Widgets – Slider

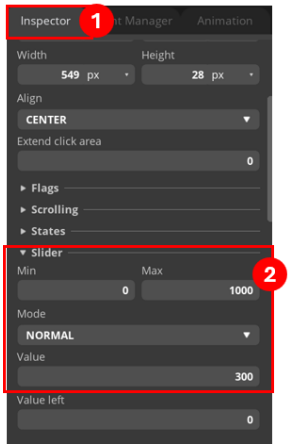
Contents	Functions	Settings
Min/Max	Defines the value range of the Slider.	
Value	Defines the initial display position of the Slider.	

Table 1-6 Slider Property Settings

7) Add the value display on Screen2: Select Label from Widgets. After it is selected, the Label will be added to the Screen2 canvas. Referring to the setup steps in Figure 1-7, rename this Label to SliderNumber.

8) Link the Slider change to the SliderNumber value: Select the Slider object on Screen2. As shown in Figure 1-11, click ADD EVENT in the Inspector, set Trigger to VALUE_CHANGED, set Action to SET VALUE FROM SLIDER, and set Target to the previously configured SliderNumber.

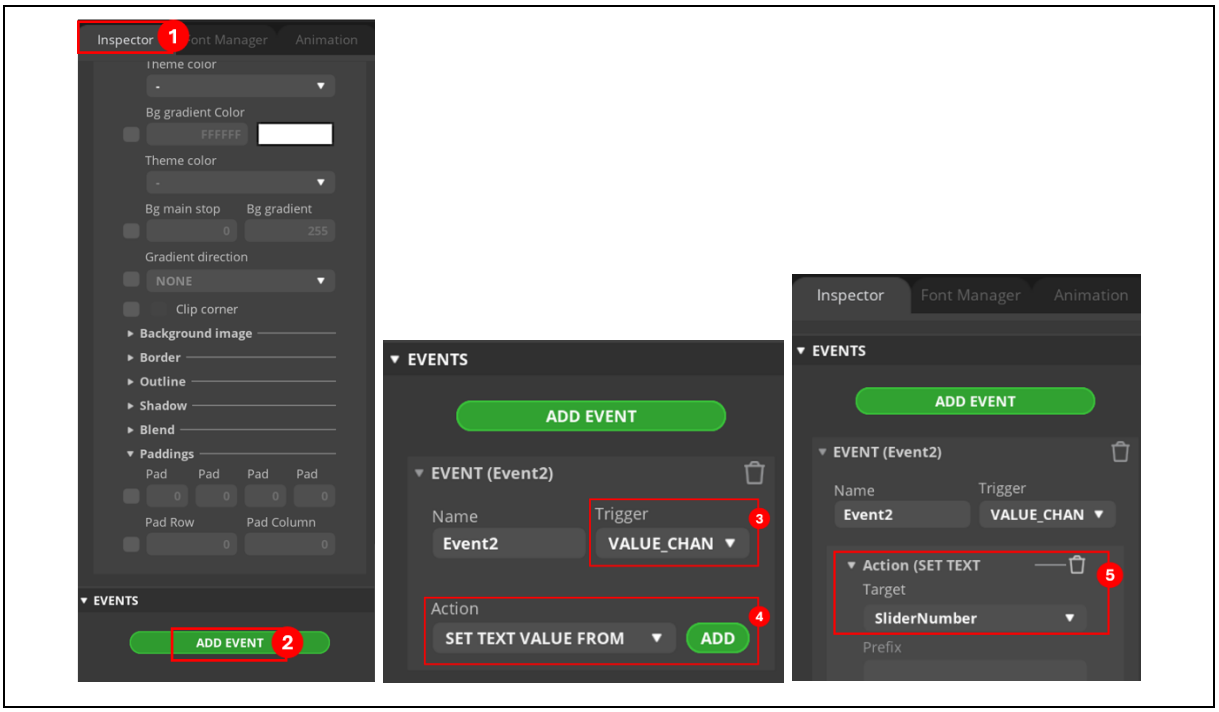


Figure 1-11 Slider and Value Linkage Settings

9) Add a Button to return to Screen1: Following the same method as step 5), add a Button and configure the linkage by referring to Figure 1-9.

10) Play Mode simulation: After completing the design, click Play as shown in Figure 1-12 to simulate the screen on the computer.

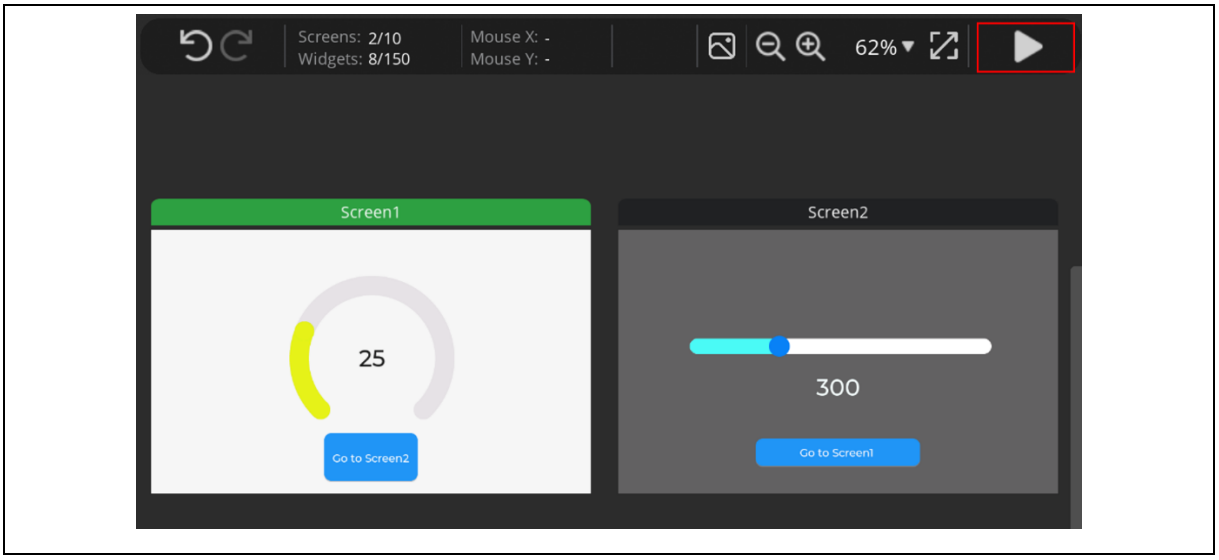


Figure 1-12 Play Mode

5. Export and open the project: In the upper-left corner of the SLS window, select Export. There are two export methods.

Export UI Files – SLS exports only the UI configuration-related files. This example also uses this export method. The export workflow and the method for opening the project are as follows.

1) As shown in Figure 1-13, when Export UI Files is selected, the exported files include those listed in Table 1-7. The save path can be configured and checked in Project Settings, as shown in Figure 1-14.

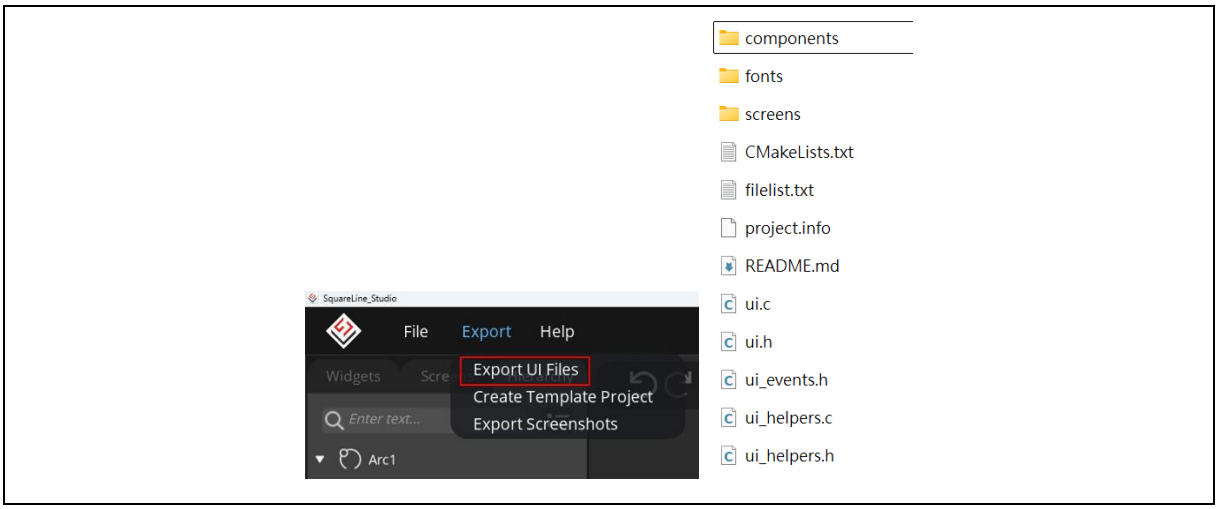


Figure 1-13 Export UI Files

Contents	Notes
Components	Stores Components, such as Status Bar and Navigation Bar, for use across multiple Screens.
Fonts	Font resources.
Screens	Each Screen corresponds to a set of ui_ScreenX.c / .h files, which create all Widgets on the screen.
ui.c / ui.h	The main UI entry point, including ui_init(), Screen initialization, default screen loading, and global Widget variable definitions.
ui_events.h	Function prototype declarations generated when an event is set to CALL FUNCTION. The user must implement the actual execution logic in ui_events.c.
ui_helpers.c ui_helpers.h	General helper functions generated by SLS.

Table 1-7 UI Files List

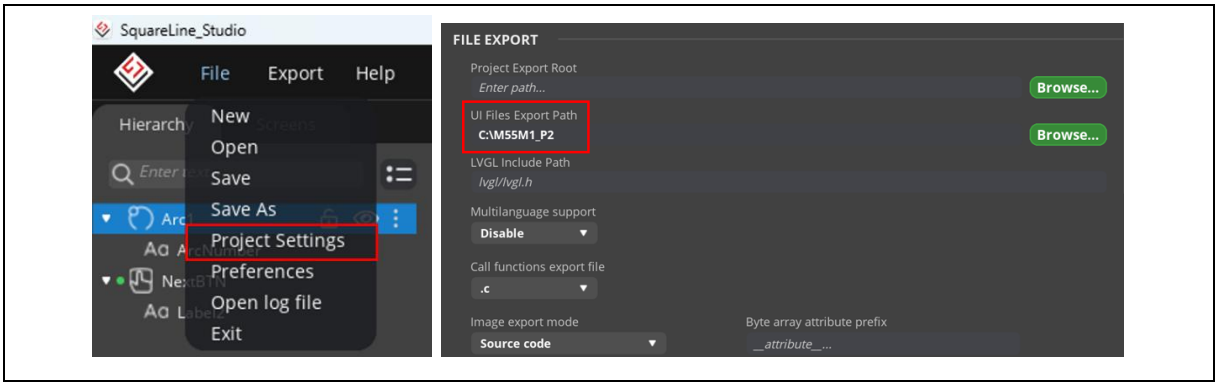


Figure 1-14 Project Settings

2) Copy the SLS UI output files to the project directory: Copy all UI-related files generated by SLS to the following path in the project: \board\numaker-x-m55m1d\sls_files.

3) Open the project as described in Section 5.

   If the project is opened with Keil MDK, add the UI-related files to the project according to the steps shown in Figure 1-15.

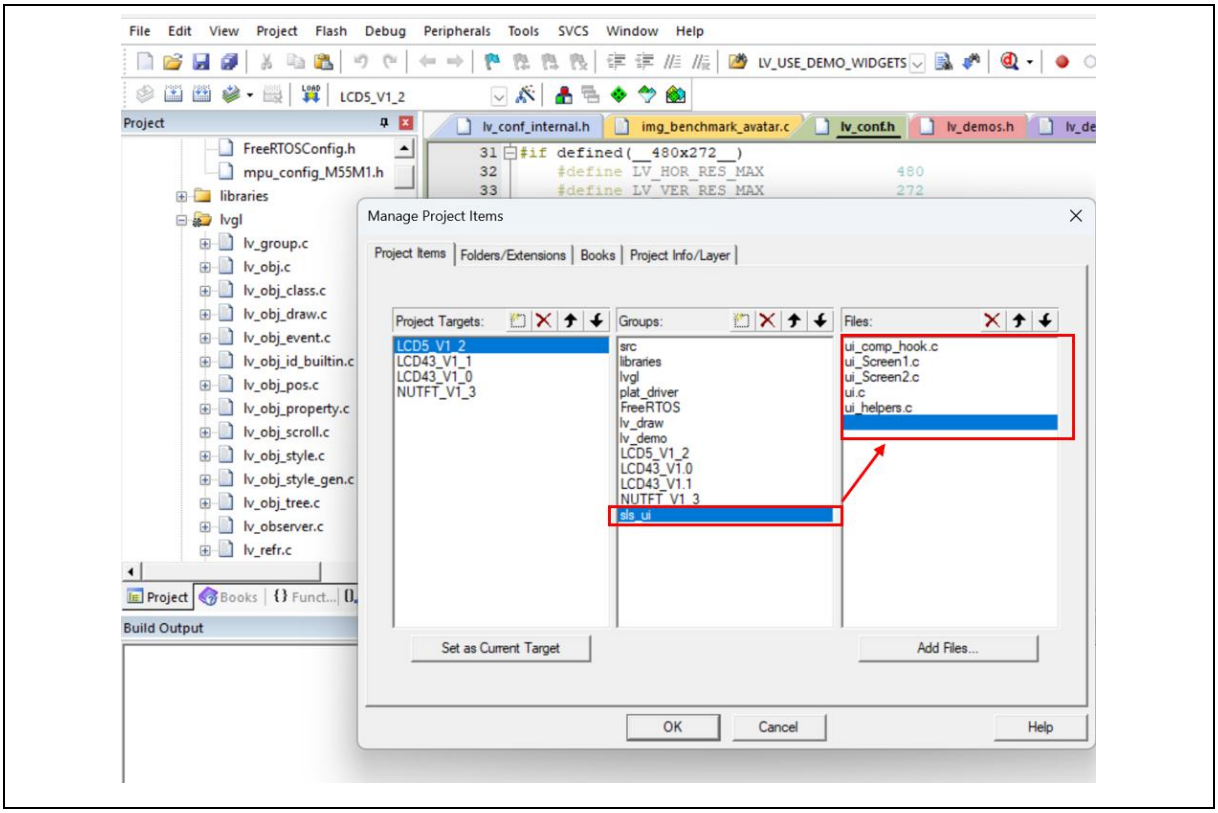


Figure 1-15 Add UI Files in Keil

If the project is opened with VS Code, as shown in Figure 1-16, any newly added UI files that need to be compiled must be manually added to the sls_files folder, and the .cproject.yml build file must also include those files.

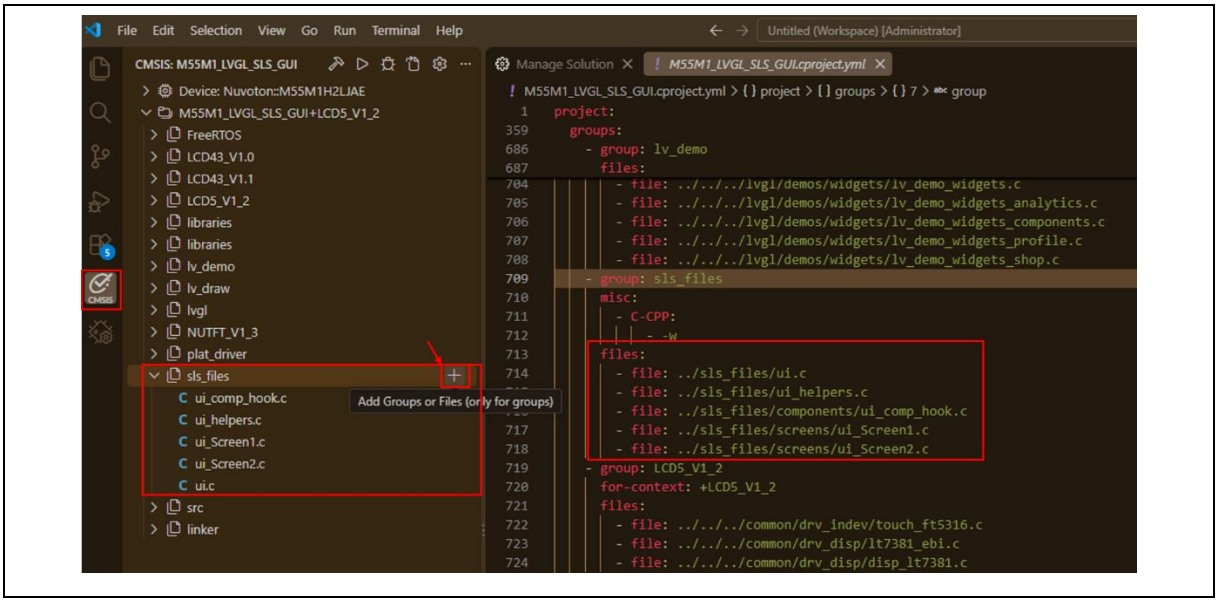


Figure 1-16 Add UI Files in VS Code

6. Add value reading and display the value on the Label object and Arc object on Screen1: When the MCU is connected to a sensor or has calculated values, the corresponding value can be written to the target display component in task_lv.c. Refer to Chapter 2 for the API and related program flow.

1.2 Execution Result

After this sample program is compiled and programmed, Screen1 displays an automatically played animation, while Screen2 provides screen control, as shown in Figure 1-17.

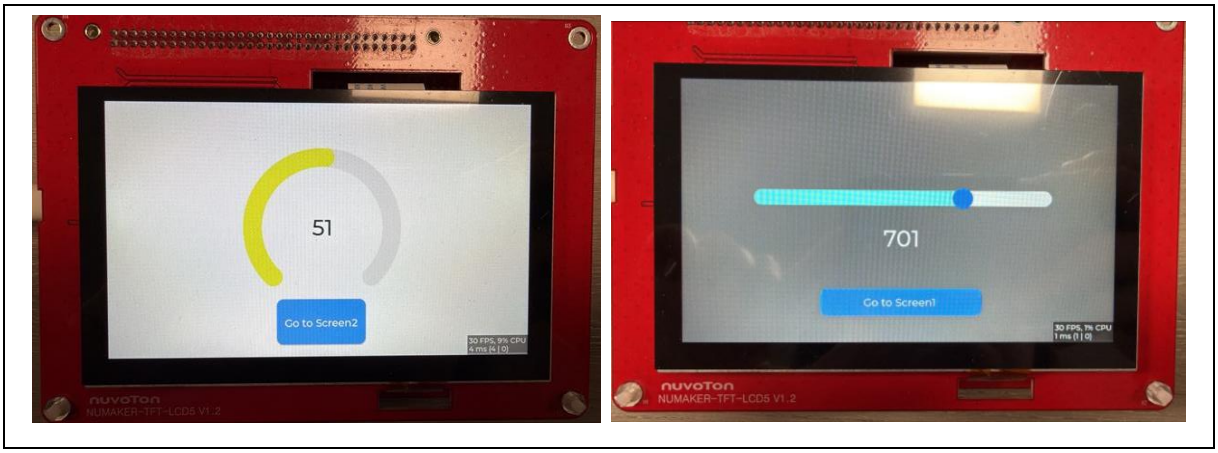


Figure 1-17 Display Output After Programming

2. Code Description

This section introduces the key code used in this example. The main program flow in main.c consists of three parts: system initialization, LVGL initialization, and the UI program flow.

```
int main(void)
{
    sys_init();

    int task_lv_init(void);
    task_lv_init();

    xTaskCreate(sensor_task, "sensor", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY +
1, NULL);

    /* Start scheduling. */
    vTaskStartScheduler();

    for (;;);

    return 0;
}
```

1. System initialization sys_init()

   This example initializes the hardware pins used by the sample program. The main settings include EBI for driving the LCD, I2C1 for driving the LCD touch interface, and SPI2, PDMA0/1, TMR0, and EADC0 for other peripheral functions. GDMA350 is also used as the acceleration function for LVGL rendering.

```
static void sys_init(void)
{
…
    /* Enable EBI clock */
    CLK_EnableModuleClock(EBI0_MODULE);

    /* Enable I2C1 clock */
    CLK_EnableModuleClock(I2C1_MODULE);

    /* Enable SPI2 clock */
    CLK_EnableModuleClock(SPI2_MODULE);

    /* Enable PDMA clock */
    CLK_EnableModuleClock(PDMA0_MODULE);
    CLK_EnableModuleClock(PDMA1_MODULE);

    /* Select TIMER clock source */
    CLK_SetModuleClock(TMR0_MODULE, CLK_TMRSEL_TMR0SEL_HIRC, 0);

    /* Enable TIMER module clock */
    CLK_EnableModuleClock(TMR0_MODULE);

    /* EADC Analog Pin */
    CLK_EnableModuleClock(EADC0_MODULE);
```

```
    /* Select EADC peripheral clock source. */
    CLK_SetModuleClock(EADC0_MODULE, CLK_EADCSEL_EADC0SEL_PCLK0, CLK_EADCDIV_EADC0DIV(8));

    /*---------------------------------------------------------------------------------*/
    /* Init I/O Multi-function                                                         */
    /*---------------------------------------------------------------------------------*/
    SET_SPI2_SS_PA11();
 …
    SET_I2C1_SDA_PB10();
    SET_I2C1_SCL_PB11();

    SET_EBI_AD0_PA5();
  …

    SYS->GPB_MFP1 &= ~(SYS_GPB_MFP1_PB7MFP_Msk | SYS_GPB_MFP1_PB6MFP_Msk);
    SYS->GPB_MFP1 |= (SYS_GPB_MFP1_PB7MFP_EADC0_CH7 | SYS_GPB_MFP1_PB6MFP_EADC0_CH6);
    SYS->GPB_MFP2 &= ~(SYS_GPB_MFP2_PB9MFP_Msk | SYS_GPB_MFP2_PB8MFP_Msk);
    SYS->GPB_MFP2 |= (SYS_GPB_MFP2_PB9MFP_EADC0_CH9 | SYS_GPB_MFP2_PB8MFP_EADC0_CH8);

    /* Disable digital path on these EADC pins */
    GPIO_DISABLE_DIGITAL_PATH(PB, BIT6 | BIT7 | BIT8 | BIT9);

    /* Enable SysTick clock */
    CLK_EnableSysTick(CLK_STSEL_ST0SEL_HIRC_DIV2, 0);

    /*---------------------------------------------------------------------------------*/
    /* Init I/O Multi-function                                                         */
    /*---------------------------------------------------------------------------------*/
    SetDebugUartMFP();

    InitDebugUart();

    /* Enable SPIM module clock */
    CLK_EnableModuleClock(SPIM0_MODULE);

    /* Enable OTFC module clock */
    CLK_EnableModuleClock(OTFC0_MODULE);

    DMA350DevInit();

}
```

2. LVGL initialization

   After the FreeRTOS scheduler starts, LVGL runs the lv_nuvoton_task() function in the lv_hdler task.

```
int task_lv_init(void)
{
    printf("stack addr = %p\n", lvgl_stack);
    xTaskCreate(lv_tick_task, "lv_tick", configMINIMAL_STACK_SIZE, NULL,
CONFIG_LV_TASK_PRIORITY, NULL);
    xTaskCreateStatic(lv_nuvoton_task, "lv_hdler", CONFIG_LV_TASK_STACKSIZE, NULL,
CONFIG_LV_TASK_PRIORITY, lvgl_stack, &lvgl_tcb);
    return 0;
}
```

The lv_nuvoton_task() function completes the LVGL core initialization, log registration, GDMA drawing unit registration, binding of time and delay functions to the FreeRTOS tick, display-device porting for the target LCD device, input-device porting, and user-interface creation. It then creates an LVGL timer for data updates and enters the main loop.

The ui_init() function related to user-interface creation is generated by SLS. It directly enters and executes the loading process from the sls_files/ui.c file.

```
void lv_nuvoton_task(void *pdata)
{
    lv_init();

#if LV_USE_LOG
    lv_log_register_print_cb(lv_nuvoton_log);
#endif

#if LV_USE_DRAW_GDMA
    void lv_draw_gdma_init(void);
    lv_draw_gdma_init();
#endif

    lv_tick_set_cb(xTaskGetTickCount);
    lv_delay_set_cb(vTaskDelay);

    extern void lv_port_disp_init(void);
    lv_port_disp_init();

    extern void lv_port_indev_init(void);
    lv_port_indev_init();

    extern void ui_init(void);
    ui_init();

    /* Sample faster than the producer so no sensor value is skipped */
    lv_timer_create(sensor_ui_timer_cb, 10, NULL);

    while (1)
    {
        lv_task_handler();
        vTaskDelay(pdMS_TO_TICKS(1));
    }
}
```

3. UI program flow

To display values obtained on the MCU side, such as EADC conversion results, I²C sensor readings, or timer counts, on the Arc gauge and Label on Screen1, two steps are required.

1) Generate values on the MCU side: In main.c, the sensor_task function simulates an I²C sensor. The simulated value is g_i32I2CSensorValue. This value scans back and forth between 0 and 100, advances one step every 50 ms, and stays at both endpoints for 1 second for easier observation.

```
static void sensor_task(void *pdata)
{
```

```
    int32_t value = 0;
    int32_t step = 1;

    (void)pdata;

    for (;;)
    {
        g_i32I2CSensorValue = value;

        if (value >= 100)
        {
            step = -1;
            vTaskDelay(pdMS_TO_TICKS(1000));
        }
        else if (value <= 0)
        {
            step = 1;
            vTaskDelay(pdMS_TO_TICKS(1000));
        }
        else
        {
            vTaskDelay(pdMS_TO_TICKS(50));
        }

        value += step;
    }
}
```

2) Bind the value on the LVGL side: In task_lv.c, which can be customized without being overwritten by SLS, call lv_arc_set_value() to update the Arc value, and call lv_label_set_text_fmt() to update the displayed number.

   The APIs for other Widgets can be found in the LVGL official documentation.

```
static void sensor_ui_timer_cb(lv_timer_t *timer)
{
    static int32_t last_value = -1;
    int32_t value = g_i32I2CSensorValue;

    (void)timer;

    if (value == last_value) return;

    last_value = value;

    if (ui_Arc1)      lv_arc_set_value(ui_Arc1, value);
    if (ui_ArcNumber) lv_label_set_text_fmt(ui_ArcNumber, "%d", (int)value);
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M55M1_Series_BSP_CMSIS_V3.01.005
- IDE version
  - Keil uVision 5.43
  - VS Code 1.138.0
- SquareLine Studio version
  - SquareLine Studio 1.6.2

3.2 Hardware Requirements

- Circuit components
  - NuMaker-X-M55M1D

4. Directory Information

The directory structure is shown below.

EC_M55M1_LVGL_SLS_GUI_V1.00	
board	Board-level files including the main application.
numaker-x-m55m1	
Keil	Keil Project
VSCode	VSCode Project
components	task_lv
sls_files	ui files generated from SLS
bsp	Board-level for board support package
m55m1	
common	Shared source code, including display drivers, drawing, touch input.
lvgl	Open-source LVGL graphics library
SLS	SquareLine Studio project file
EC_M55M1_LVGL_SLS_GUI.spj	
thirdparty	Third-party components, primarily the FreeRTOS library

Figure 4-1 Directory Structure

5. Example Code Execution

5.1 Keil MDK

1. Navigate to the **Keil** project folder in Windows Explorer and open the *M55M1_LVGL_SLS_GUI.uvprojx* project file.
2. Build
3. Download
4. Enter debug mode
    - Run

5.2 VS Code

1. Select **File** in VS Code IDE
    - Click **Open Folder**
    - Select the **VSCode** folder under the project directory
2. Select **CMSIS**
    - Click **Build solution** to compile the project
    - Click **Load & Run application** to program the target chip
    - Click **Load & Debug application** to enter debug mode

6. Revision History

Date	Revision	Description
2026.09.30	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.