GUI emWin Start Guide

V1.00.001

Publication Release Date: Jun. 2018

Support Chips:

M23 Series

Support Platforms:

Non-OS

Table of Contents

1. Introduction

1.1. Introduction

emWin is a graphic library with graphical user interface (GUI). It is designed to provide an efficient, processor- and display controller-independent graphical user interface (GUI) for any application that operates with a graphical display.

```
To utilize M2351 samples, please make sure the version of Keil MDK IDE is at
least V5.24.0.0.
```

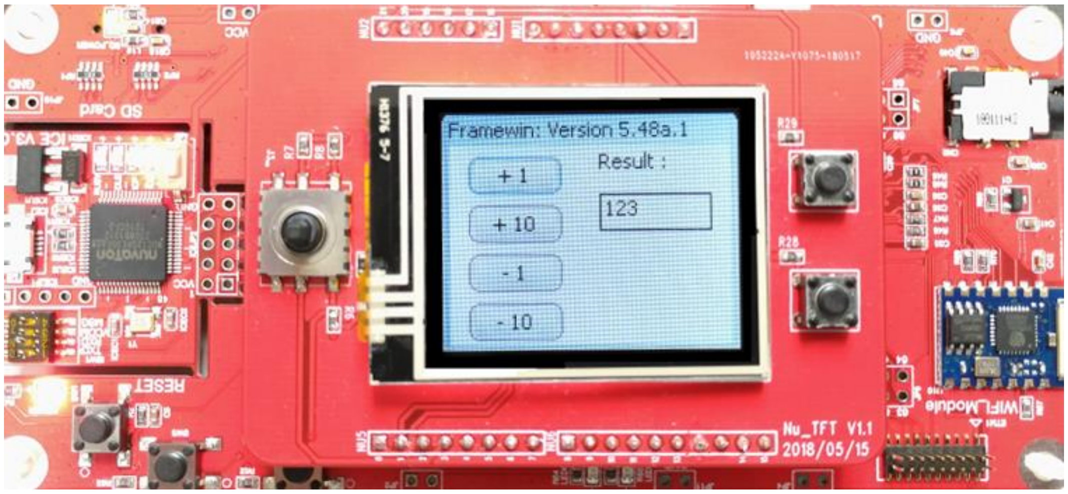


Figure 1.1-1 emWin runs on M2351.

2. Start emWin

2.1. Step 1: Open project

"emWin_SimpleDemo" is a sample code to demonstrate the emWin GUI system. It contains a frame window, four buttons, a text and a text editor.
We can touch the GUI button and check the result that shown on the text editor.

```
To utilize touch feature, please make sure we have the daughter board.
```

Here is the project path and structure:

"\SampleCode\emWin_SimpleDemo" is the emWin sample code path.

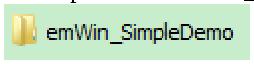

emWin_SimpleDemo

Figure 2.1-1 "emWin_SimpleDemo" sample path.

Sample project structure: \SampleCode\emWin_SimpleDemo\KEIL\emWin_SimpleDemo.uvprojx
The scope of BSP is in the blue part.
The scope of emWin is in the red part.

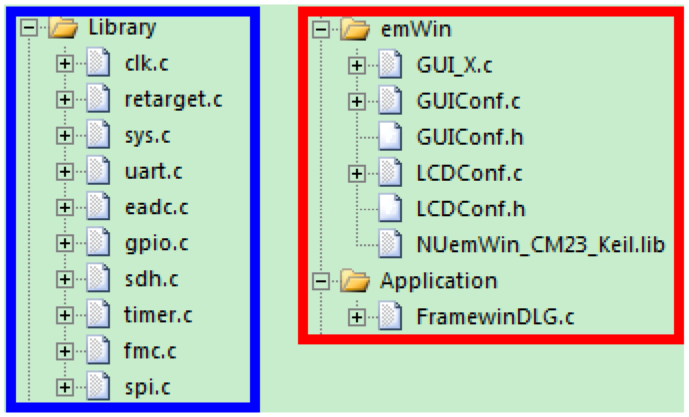

```
Library
    clk.c
    retarget.c
    sys.c
    uart.c
    eadc.c
    gpio.c
    sdh.c
    timer.c
    fmc.c
    spi.c
emWin
    GUI_X.c
    GUIConf.c
    GUIConf.h
    LCDConf.c
    LCDConf.h
    NUemWin_CM23_Keil.lib
Application
    FramewinDLG.c
```

Figure 2.1-2 "emWin_SimpleDemo" project structure.

2.2. Step 2: BSP Initilization

Initialize M2351 non-OS BSP to utilize the device system, e.g., Uart debug port, display output panel, and resistor-type touch panel.

```
To utilize touch feature, please make sure we have the daughter board.
```

BSP initization descripted in \SampleCode\emWin_SimpleDemo\main.c.

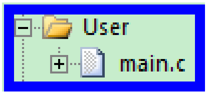


Figure 2.2-1 BSP initialization on main.c.

```
int main(void)
{
    //
    // Init System, IP clock and multi-function I/O
    //
    SYS_Init();
    //
    // Init UART to 115200-8n1 for print message
    //
    UART_Open(UART0, 115200);

    // Enable Timer0 clock and select Timer0 clock source
    //
    CLK_EnableModuleClock(TMR0_MODULE);
    CLK_SetModuleClock(TMR0_MODULE, CLK_CLKSEL1_TMR0SEL_HXT, 0);
    //
    // Initial Timer0 to periodic mode with 1000Hz
    //
    TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, 1000);
    //
    // Enable Timer0 interrupt
    //
    TIMER_EnableInt(TIMER0);
    NVIC_SetPriority(TMR0_IRQn, 1);
    NVIC_EnableIRQ(TMR0_IRQn);
    //
    // Start Timer0
    //
    TIMER_Start(TIMER0);

    //SysTick_Config(SystemCoreClock / 1000);
    printf("\n\nCPU @ %d Hz\n", SystemCoreClock);

    MainTask();
    while(1);
}
```

2.3. Step 3: emWin Initilization

To utilize emWin, we need to initialize emWin. MainTask() will start emWin GUI system.

\SampleCode\emWin_SimpleDemo\main.c:

```
void MainTask(void)
{
    GUI_Init();
    CreateFramewin();
    while (1)
    {
        GUI_Delay(500);
    }
}
```

2.4. Step 4: Build

To start working with the application, we need to utilize Keil MDK to build the project.

Press [F7] to compile the application or click "Rebuild".

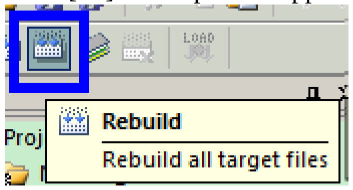


Figure 2.4-1 Build project.

2.5. Step 5: Download and run

Press CTRL + [F5] to download the application and start a debug session. After downloaded, it will halt at main() and we should see the similar screenshow below.

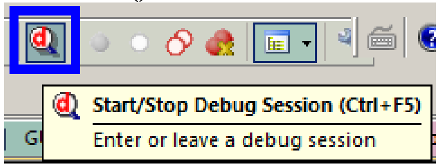
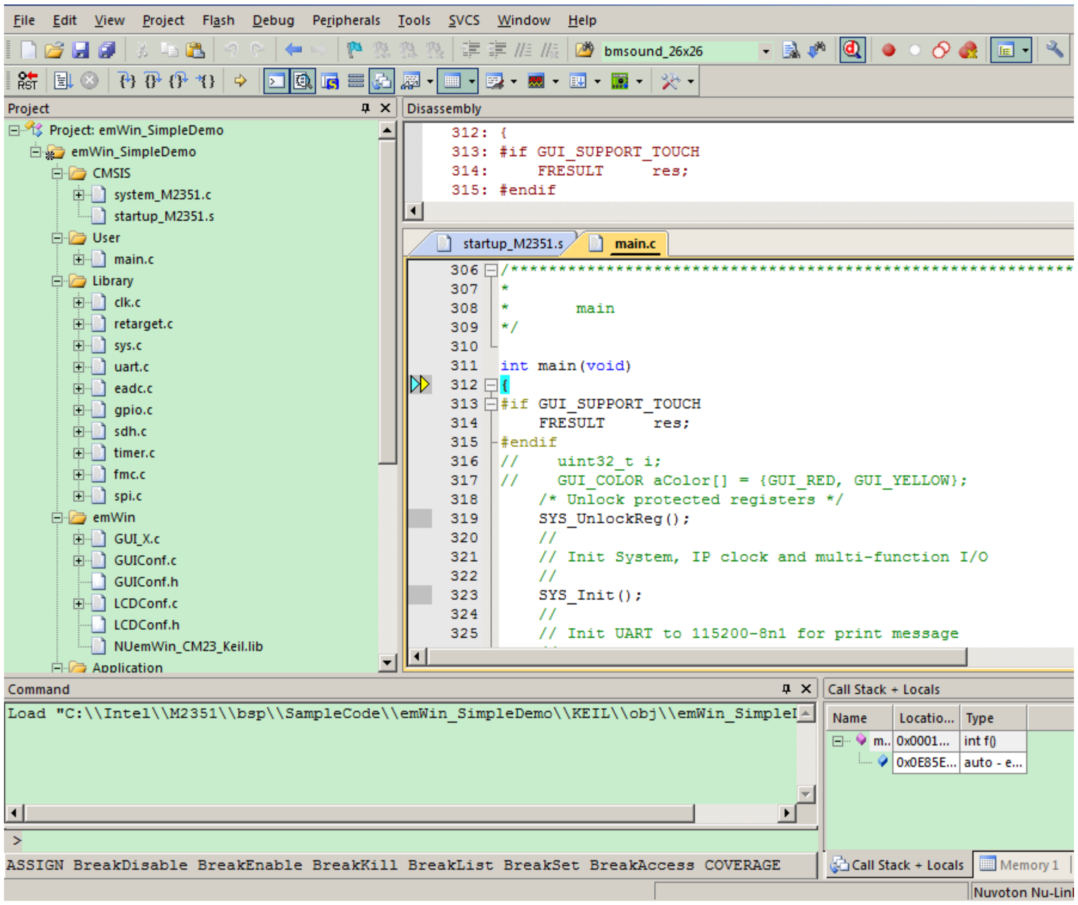

Start/Stop Debug Session (Ctrl+F5)
Enter or leave a debug session

Figure 2.5-1 Download and run application.



Project: emWin_SimpleDemo
- emWin_SimpleDemo
  - CMSIS
    - system_M2351.c
    - startup_M2351.s
  - User
    - main.c
  - Library
    - clk.c
    - retarget.c
    - sys.c
    - uart.c
    - eadc.c
    - gpio.c
    - sdh.c
    - timer.c
    - fmc.c
    - spi.c
  - emWin
    - GUI_X.c
    - GUIConf.c
    - GUIConf.h
    - LCDConf.c
    - LCDConf.h
    - NUemWin_CM23_Keil.lib
  - Application

```
312: {
313: #if GUI_SUPPORT_TOUCH
314:     FRESULT     res;
315: #endif
```

```
/*****************************************************************
*
*       main
*/

int main(void)
{
#if GUI_SUPPORT_TOUCH
    FRESULT     res;
#endif
//    uint32_t i;
//    GUI_COLOR aColor[] = {GUI_RED, GUI_YELLOW};
    /* Unlock protected registers */
    SYS_UnlockReg();
    //
    // Init System, IP clock and multi-function I/O
    //
    SYS_Init();
    //
    // Init UART to 115200-8n1 for print message
```

```
Load "C:\\Intel\\M2351\\bsp\\SampleCode\\emWin_SimpleDemo\\KEIL\\obj\\emWin_SimpleI
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE
```

Name	Locatio...	Type
m..	0x0001...	int f()
	0x0E85E...	auto - e...

Figure 2.5-2 Debug session.

2.6. Touch screen

We stored the calibrated parameters to APROM at 256 KB address. (0x40000)

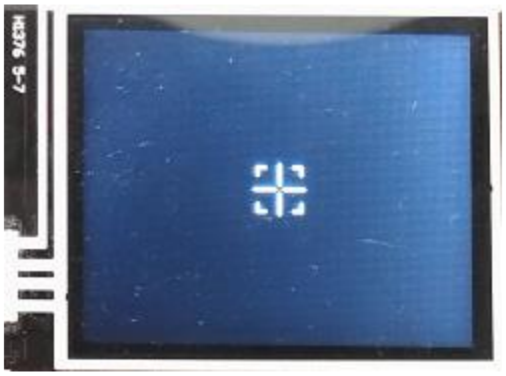


Figure 2.6-1 Third-party touch calibration.

```
/*
To utilize touch feature, please make sure we have the dauther board.
*/

#define __DEMO_TSFILE_ADDR__   0x00040000 /* SPI flash 256KB address */

#if GUI_SUPPORT_TOUCH
    g_enable_Touch = 0;

Init_TouchPanel();
    /* Unlock protected registers */
    SYS_UnlockReg();

    /* Enable FMC ISP function */
    FMC_Open();

    /* SPI flash 256KB + 0x1C marker address */
    if (FMC_Read(__DEMO_TSFILE_ADDR__ + 0x1C) != 0x55AAA55A)
    {
        FMC_ENABLE_AP_UPDATE();
        /* utilize open source "tslib" for the calibration */
        ts_calibrate(__DEMO_TS_WIDTH__, __DEMO_TS_HEIGHT__);
        // Erase page
        FMC_Erase(__DEMO_TSFILE_ADDR__);
        ts_writefile();
        FMC_DISABLE_AP_UPDATE();
    }
    else
    {
        ts_readfile();
    }

    /* Disable FMC ISP function */
    FMC_Close();

    /* Lock protected registers */
    SYS_LockReg();

    g_enable_Touch = 1;
#endif
```

For resistor-type touch panel, we can utilize EADC to convert the position of x and y.

\Library\StdDriver\src\eadc.c

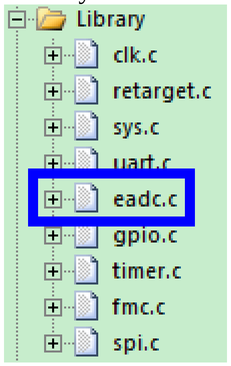


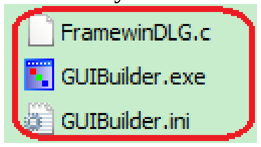
Figure 2.6-2 Utilize eadc to convert the voltage values of x and y.

3. Start emWin GUIBuilder

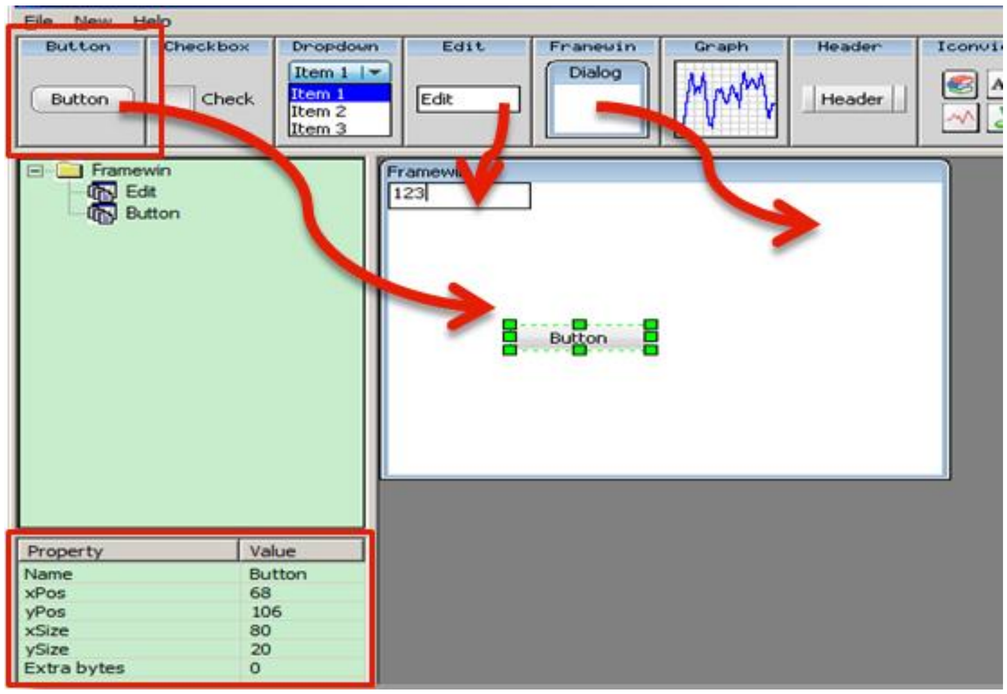
3.1. Step 1: Create widget

To create widget, we can utilize windows tool “GUIBuilder” to generate c source file.

\ThirdParty\emWin\Tool\GUIBuilder.exe:



FramewinDLG.c
GUIBuilder.exe
GUIBuilder.ini



Property	Value
Name	Button
xPos	68
yPos	106
xSize	80
ySize	20
Extra bytes	0

Figure 3.1-1 emWin GUIBuilder.

After execute “File” → “Save…”, we can get the source file called “FramewinDLG.c”.

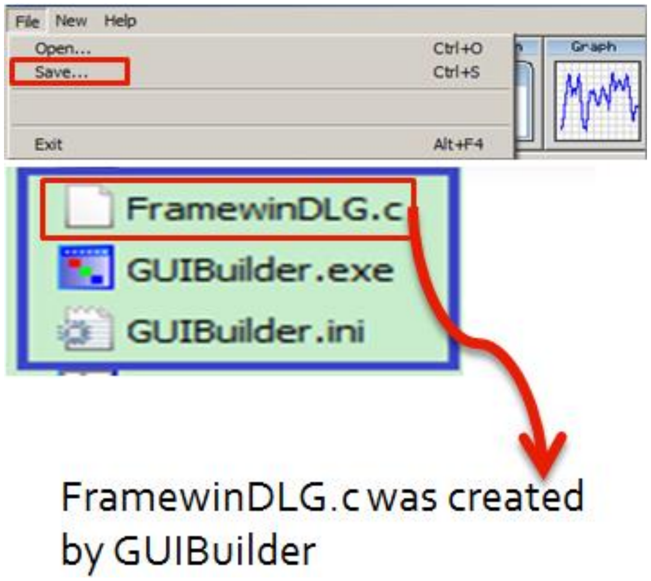


Figure 3.1-2 emWin GUIBuilder can generate a GUI layout and c source file.

3.2. Step 2: Handle widget event

In “FramewinDLG.c”, we can add code to utilize widget event, e.g., initialization, button click, release and change the content data of text editor.

\SampleCode\emWin_SimpleDemo\Application\FramewinDLG.c:

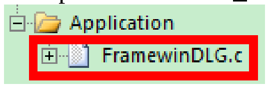


Figure 3.2-1 emWin GUI application c source file.

```
switch (pMsg->MsgId) {
case WM_INIT_DIALOG:
//
// Initialization of 'Edit'
//
value = 123;
sprintf(sBuf,"%d    ", value);
hItem = WM_GetDialogItem(pMsg->hWin, ID_EDIT_0);
EDIT_SetText(hItem, sBuf);

// USER START (Optionally insert additional code for further widget
initialization)
// USER END
break;
case WM_NOTIFY_PARENT:
Id    = WM_GetId(pMsg->hWinSrc);
NCode = pMsg->Data.v;
switch(Id) {
case ID_BUTTON_0: // Notifications sent by '+ 1'
switch(NCode) {
case WM_NOTIFICATION_CLICKED:
// USER START (Optionally insert code for reacting on notification message)
// USER END
printf("clicked\n");
break;
case WM_NOTIFICATION_RELEASED:
// USER START (Optionally insert code for reacting on notification message)
value += 1;
sprintf(sBuf,"%d    ", value);
```

```
hItem = WM_GetDialogItem(pMsg->hWin, ID_EDIT_0);
EDIT_SetText(hItem, sBuf);
printf("released\n");
// USER END
break;
```

4. How to change display panel

4.1. Step 1: emWin display

Please note that if the display controller is “non readable”, some features of emWin will not work. The list is shown below:

- Cursors and Sprites
- XOR-operations, required for text cursors in EDIT and MULTIEDIT widgets
- Alpha blending
- Antialiasing

\ThirdParty\emWin\Config\LCDConf.c declared the resolution of the display panel.

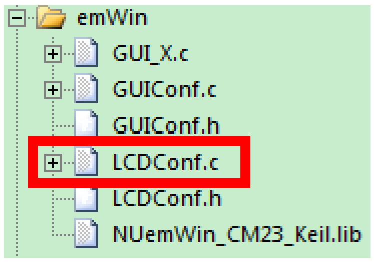


Figure 4.1-1 emWin display definition.

In \ThirdParty\emWin\Config\LCDConf.c and .h, we need to assign MPU-type render approach and resolution:

```
/**********************************************************************
*
*      Layer configuration
*
**********************************************************************
*/
//
// Physical display size
//
#define XSIZE_PHYS 162
#define YSIZE_PHYS 132
```

```
//
// Orientation
//
Config.Orientation = (GUI_MIRROR_X | GUI_SWAP_XY);
GUIDRV_FlexColor_Config(pDevice, &Config);
//
// Set controller and operation mode
//
PortAPI.pfWrite8_A0  = _Write0;
PortAPI.pfWrite8_A1  = _Write1;
PortAPI.pfWriteM8_A1 = _WriteM1;

/* FIXME if panel supports read back feature */
PortAPI.pfRead8_A1   = _Read1;
PortAPI.pfReadM8_A1  = _ReadM1;
GUIDRV_FlexColor_SetFunc(
pDevice, &PortAPI, GUIDRV_FLEXCOLOR_F66709, GUIDRV_FLEXCOLOR_M16C0B8);
```

The implementation to write command/data to LCD controller:

Write command:

```
/*-----------------------------------------------*/
// Write control registers of LCD module
//
/*-----------------------------------------------*/
static void _Write0(uint8_t cmd)
{
    LCM_DC_CLR;
    SPI_CS_CLR;

    SPI_WRITE_TX(SPI_LCD_PORT, Cmd);
    while(SPI_IS_BUSY(SPI_LCD_PORT));

    SPI_CS_SET;
}
```

Wrtie data:

```
/*********************************************************************
*
*      _Write1
*/
static void _Write1(U8 Data) {
    LCM_DC_SET;
    SPI_CS_CLR;

    SPI_WRITE_TX(SPI_LCD_PORT, Data);

    while(SPI_IS_BUSY(SPI_LCD_PORT));
    SPI_CS_SET;
}
```

The implementation to read data from LCD controller:

```
/*-----------------------------------------------*/
// Read data from SRAM of LCD module
//
/*-----------------------------------------------*/
uint8_t _Read1(void)
{
#if 1
    /* FIXME if panel supports read back feature */
    return 0;
#else
    LCM_DC_SET;
    SPI_CS_CLR;
    SPI_WRITE_TX(SPI_LCD_PORT, 0x00);
    SPI_READ_RX(SPI_LCD_PORT);
    SPI_CS_SET;
    return (SPI_READ_RX(SPI_LCD_PORT));
#endif
}
```

4.2. Step 2: BSP display

To utilize SPI interface, we can communicate with display controller:

- send command
- send data
- receive data (optional)

\Library\StdDriver\src\spi.c

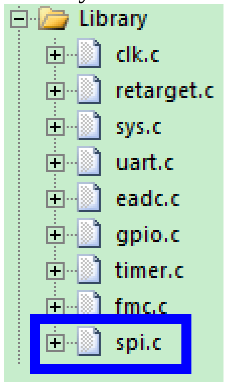


Figure 4.2-1 BSP SPI driver.

5. Revision History

Version	Date	Description
V1.00.001	Jun. 27, 2018	● Created