GUI emWin
Start Guide

V1.00.001

Publication Release Date: Jun. 2018

Support Chips:

M0 Series

Support Platforms:

Non-OS

The information in this document is subject to change without notice.

The Nuvoton Technology Corp. shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

This documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from the Nuvoton Technology Corp.

Nuvoton Technology Corp. All rights reserved.

Table of Contents

1. Introduction 4
   - 1.1. Introduction 4
2. Start emWin 5
   - 2.1. Step 1: Open project 5
   - 2.2. Step 2: BSP Initilization 6
   - 2.3. Step 3: emWin Initilization 8
   - 2.4. Step 4: Build 9
   - 2.5. Step 5: Download and run 10
   - 2.6. Touch screen 11
3. Start emWin GUIBuilder 13
   - 3.1. Step 1: Create widget 13
   - 3.2. Step 2: Handle widget event 14
4. How to change display panel 16
   - 4.1. Step 1: emWin display 16
   - 4.2. Step 2: BSP display 19
5. Resource usage 20
   - 5.1. System resource usage 20
   - 5.2. Widget resource usage 21
     - GUI componets memory requirements: 21
     - Stack memory requirements: 21
6. Revision History 22

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.

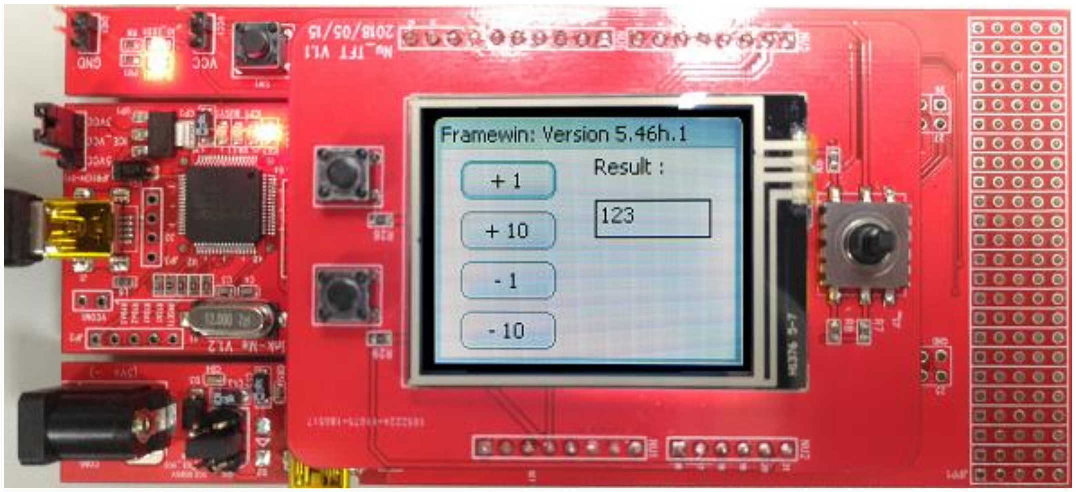

Figure 1.1-1 emWin runs on NUC126.

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 panel, please make sure we have the touch panel on the
NUC126 daughter board.
```

Here is the project path and structure:

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

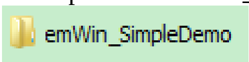

Figure 2.1-1 "emWin_SimpleDemo" sample path.

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

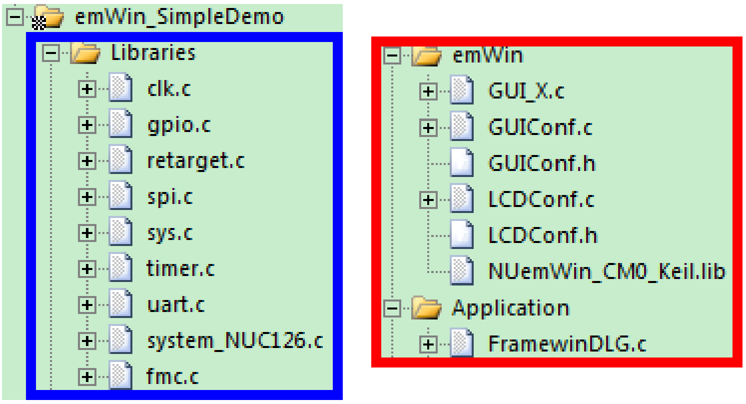

Figure 2.1-2 "emWin_SimpleDemo" project structure.

2.2. Step 2: BSP Initilization

Initialize NUC126 non-OS BSP to utilize the device system, e.g., Uart debug port, display output panel, and resistor-type touch panel.

```
To utilize touch panel, please make sure we have the touch panel on the
NUC126 daughter board.
```

BSP initization descripted in \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.

\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”.

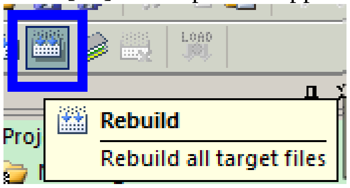

Figure 2.4-1 Build project.

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.

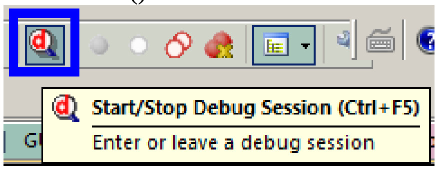

Figure 2.5-1 Download and run application.

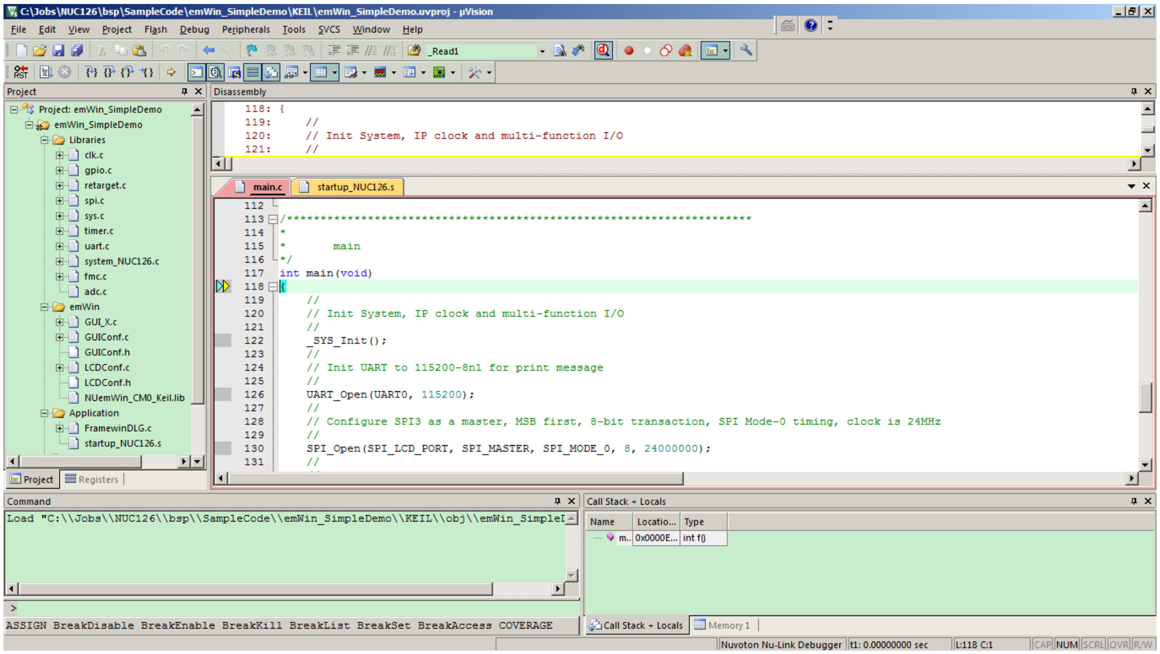

Figure 2.5-2 Debug session.

2.6. Touch screen

We stored the calibrated parameters to APROM at 128bytes address. (0x20000)

```
To utilize touch panel, please make sure we have the touch panel on the
NUC126 dauther board.

#if GUI_SUPPORT_TOUCH
    g_enable_Touch = 0;

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

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

    /* SPI flash 128KB + 0x1C marker address */
    if (FMC_Read(0x20000 + 0x1C) != 0x55AAA55A)
    {
        FMC_EnableAPUpdate();
        /* utilize open source "tslib" for the calibration */
        ts_calibrate(162, 132);
        // Erase page
        FMC_Erase(0x20000);
        ts_writefile();
        FMC_DisableAPUpdate();
    }
    else
    {
        ts_readfile();
    }

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

    /* Lock protected registers */
    SYS_LockReg();
```

```
//    ts_test(162, 132);


    g_enable_Touch = 1;
#endif
```

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

\Library\StdDriver\src\adc.c

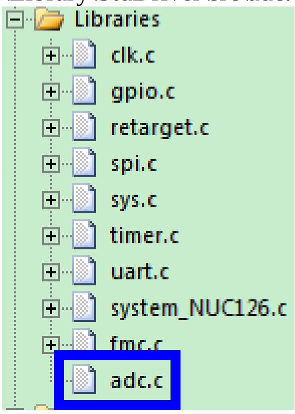

Figure 2.6-1 utilize eadc to convert the position of x and y.

3. Start emWin GUIBuilder

3.1. Step 1: Create widget

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

\emWin\Tool\GUIBuilder.exe:

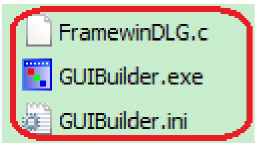

Figure 3.1-1 emWin GUIBuilder.

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

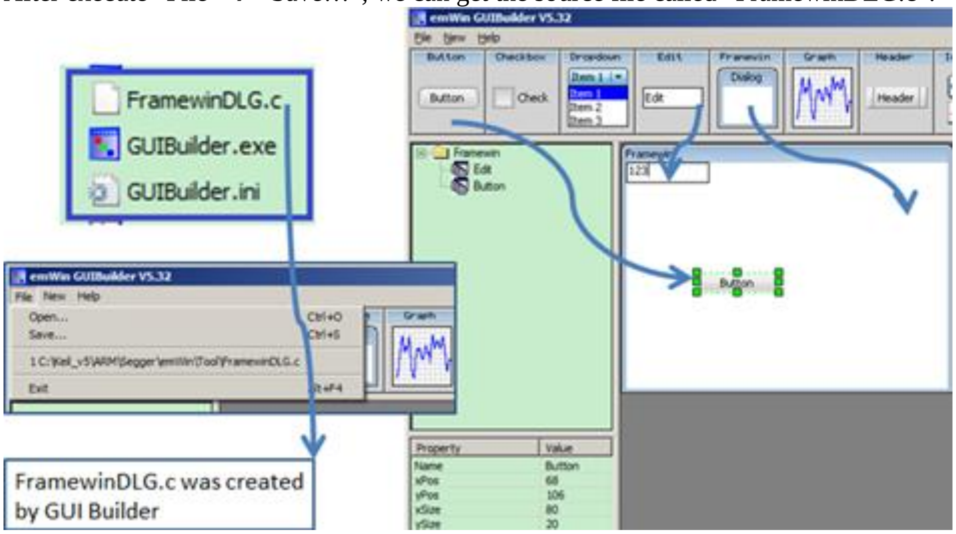

Figure 3.1-2 emWin GUIBuilder can generate a GUI layout and source file.

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.

\emWin_SimpleDemo\Application\FramewinDLG.c:

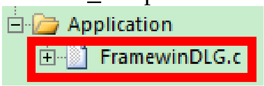

Figure 3.2-1 emWin GUI application source file.

```
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 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

emWin LCDConf.c declare the resolution of the display panel.

\ThirdParty\emWin\Config\LCDConf.c

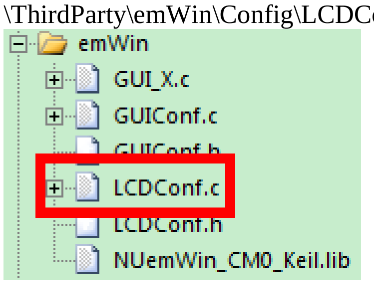

Figure 4.1-1 emWin display define.

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_MIRROR_Y | 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 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;
}

/*********************************************************************
*
*       _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

BSP spi.c defines the driver interface.

\Library\StdDriver\src\ebi.c

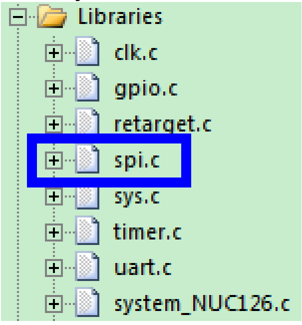

Figure 4.2-1 BSP SPI interface for MPU-type LCD.

5. Resource usage

5.1. System resource usage

For NUC126, emWin_SimpleDemo needs ROM size 81KB and RAM size 7.4KB.
ROM contains code (72.8KB) and read-only data (8.1KB).
RAM contains stack (1.25KB).

emWin utilizes 1 Framewin, 4 buttons, 1 text label and 1 text editor.

System contains BSP related driver, initialization flow and open source library “tslib”.

The resource measured as M0, KEIL V5.20 and emWin V5.46h.1 with optimization.

NUC126	**ROM**		**RAM**	
	Total: 256KB		Total: 20KB	
emWin_SimpleDemo	Used: 81KB		Used: 7.4KB	
	72.8KB	8.1KB	1.25KB	4KB
Comments	CODE of system and emWin	RO of system and emWin	STACK of system and emWin	RW of emWin

Table 5.1-1 System resource usage.

5.2. Widget resource usage

If we only utilize GUI_Init and draw some string. For NUC126, emWin_SimpleDemo needs ROM size 57.9KB and RAM size 6.9KB.
ROM contains code (54.1KB) and read-only data (3.7KB).
RAM contains stack (1.25KB).

GUI componets memory requirements:

The following table shows some GUI components memory requirements and for more details please reference the user manual in \ThirdParty\emWin\Doc\

Component	ROM	RAM
Core	5.2KB	80B
Driver	2~8KB	20B
Window Manager	6.2KB	2.5KB
Memory Devices	4.7KB	7KB
Widgets	4.5KB	
Widget / BUTTON	1KB	40B
Widget / EDIT	2.2KB	28B
Widget / FRAMEWIN	2.2KB	12B
Widget / TEXT	0.4KB	16B
Widget / CHECKBOX	1KB	52B
Widget / DROPDOWN	1.8KB	52B
Widget / GRAPH	2.9KB	48B
Widget / LISTBOX	3.7KB	56B
Widget / LISTVIEW	3.6KB	44B
Widget / MENU	5.7KB	52B
Widget / MULTIEDIT	7.1KB	16B
Widget / MULTIPAGE	3.9KB	32B
Widget / PROGBAR	1.3KB	20B
Widget / RADIOBUTTON	1.4KB	32B
Widget / SCROLLBAR	2KB	14B
Widget / SLIDER	1.3KB	16B

Table 5.2-1 GUI components resource usage.

Stack memory requirements:

The basic stack requirement is approximate 600 bytes. If to utilize the Window Manager additional 600 bytes should be calculated. For Memory Devices further additional 200 bytes are recommended. Please note that the stack requirement also depends on the application, the used compiler and the CPU.

6. Revision History

Version	Date	Description
V1.00.001	Jun. 14, 2018	● Created

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in equipment or systems intended for surgical implantation, atomic energy control instruments, aircraft or spacecraft instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for any other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications whereby failure could result or lead to personal injury, death or severe property or environmental damage.

Nuvoton customers using or selling these products for such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from their improper use or sales.