GUI emWin
Start Guide

V1.00.002

Publication Release Date: Jul. 2018

Support Chips:

M480 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.

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.

Figure 1.1-1 emWin runs on M480 series.

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 click button by touch and check the result that shown on the text editor.
Here is the project path and structure:

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

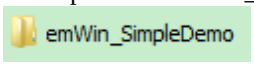

Figure 2.1-1 "emWin_SimpleDemo" sample path.

Sample project structure: \SampleCode\emWin_SimpleDemo\KEIL\emWin_SimpleDemo.uvproj
The scope of BSP is in the blue part. (enable sdh.c if project emWin_SimpleDemo_SD selected)
The scope of emWin is in the red part.

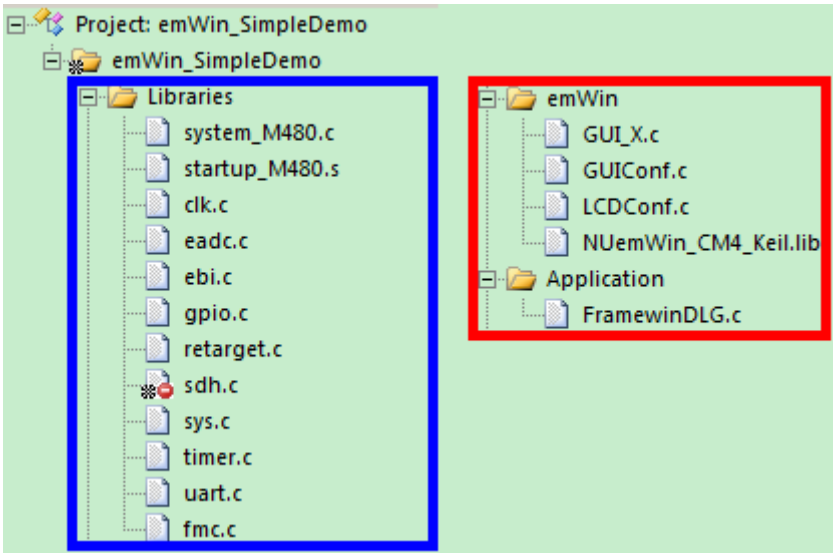

Figure 2.1-2 "emWin_SimpleDemo" project structure.

2.2. Step 2: BSP Initilization

Initialize M480 series non-OS BSP to utilize the device system, e.g., Uart debug port, display output panel, and resistor-type touch panel.

BSP initization descripted in \emWin_SimpleDemo\main.c.

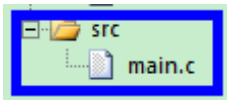

Figure 2.2-1 BSP initialization on main.c.

```
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_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”.

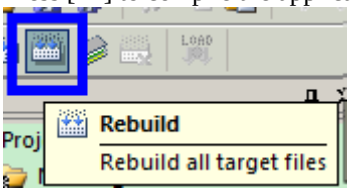

Figure 2.4-1 Build project.

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.

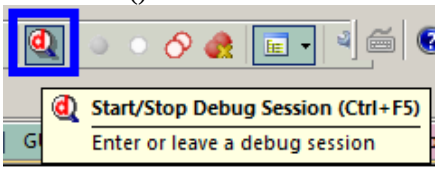

Figure 2.5-1 Download and run application.

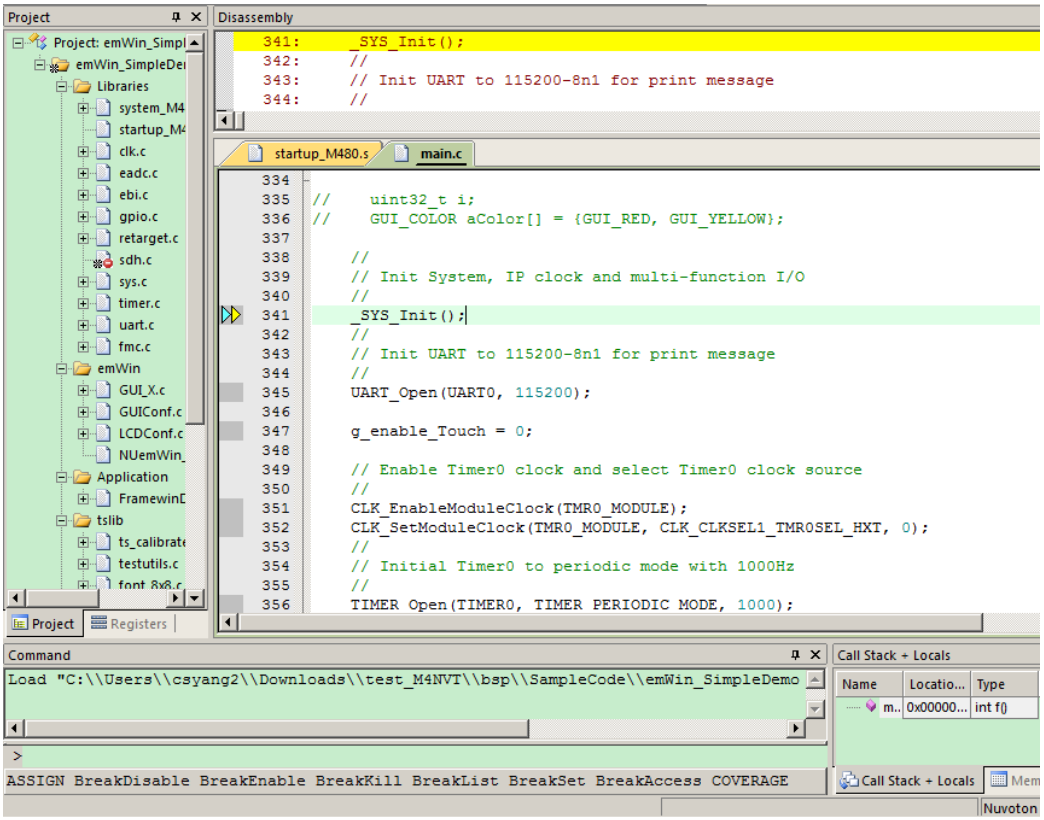

Figure 2.5-2 Debug session.

2.6. Touch screen

For resistor-type touch panel, we can utilize ADC to convert the reference voltage of x and y. Then, we utilize open source “tslib” to convert into the coordinates of x and y.

\Library\StdDriver\src\eadc.c

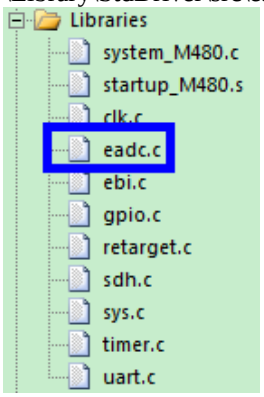

Figure 2.6-1 utilize eadc to convert the position of x and y.

We store the calibration results in the APROM or SD card. (default project is “emWin_SimpleDemo” and store the calibration results in APROM)

Project “emWin_SimpleDemo”: the calibration results in the APROM address 0x00040000.
Project “emWin_SimpleDemo_SD”: the calibration results in SD card called “TS_CALIB”.

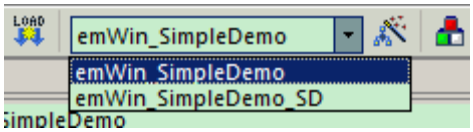

Figure 2.6-2 Two project settings for the location of calibration results.

In the project “emWin_SimpleDemo_SD”, we define “__USE_SD__” to utilize SD card to store the calibration results:

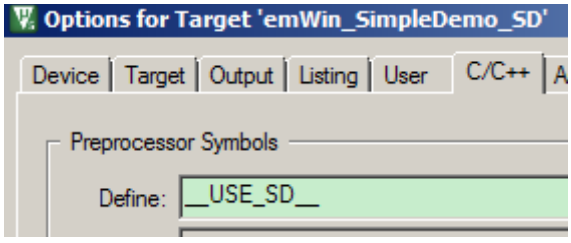

Figure 2.6-3 SD definition for the project “emWin_SimpleDemo_SD”.

In “M48XTouchPanel.h”, we defined the width and height of the touch screen and APROM address for the calibration results:

```
#ifndef __M48XTOUCHPANEL_H__
#define __M48XTOUCHPANEL_H__

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

#define __DEMO_TS_WIDTH__       320
#define __DEMO_TS_HEIGHT__      240

#endif
```

In “main.c”, to utilize APROM to store the calibration results:

```
/* 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();
        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();
```

In “main.c”, to utilize SD card to store the calibration results:

Please note that before access SD card, we need to enable SDH module and its callback function. We utilize open source “FatFs” as a generic FAT filesystem module.

```
#ifdef __USE_SD__
    SDH_Open_Disk(TEST_SDH, CardDetect_From_GPIO);
    printf("rc=%d\n", (WORD)disk_initialize(0));
    disk_read(0, Buff, 2, 1);
    //f_mount(&FatFs[0], "", 0);  // for FATFS v0.11
    GUI_Init();
    res = f_open(&hFile, "0:\\ts_calib", FA_OPEN_EXISTING | FA_READ);
    if (res)
    {
        // file does not exists, so do calibration
        res = f_open(&hFile, "0:\\ts_calib", FA_CREATE_ALWAYS | FA_WRITE);
        if ( res )
        {
            f_close(&hFile);
            GUI_DispStringAt("CANNOT create the calibration file.\nPlease
insert a SD card then reboot.", 0, 0);
            while(1);
        }

        ts_calibrate(__DEMO_TS_WIDTH__, __DEMO_TS_HEIGHT__);
        //GUI_SetDrawMode(GUI_DRAWMODE_NORMAL);
        ts_writefile();
    }
    else
    {
        ts_readfile();
    }
    f_close(&hFile);
#else
```

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:

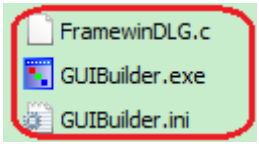

Figure 3.1-1 emWin GUIBuilder.

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

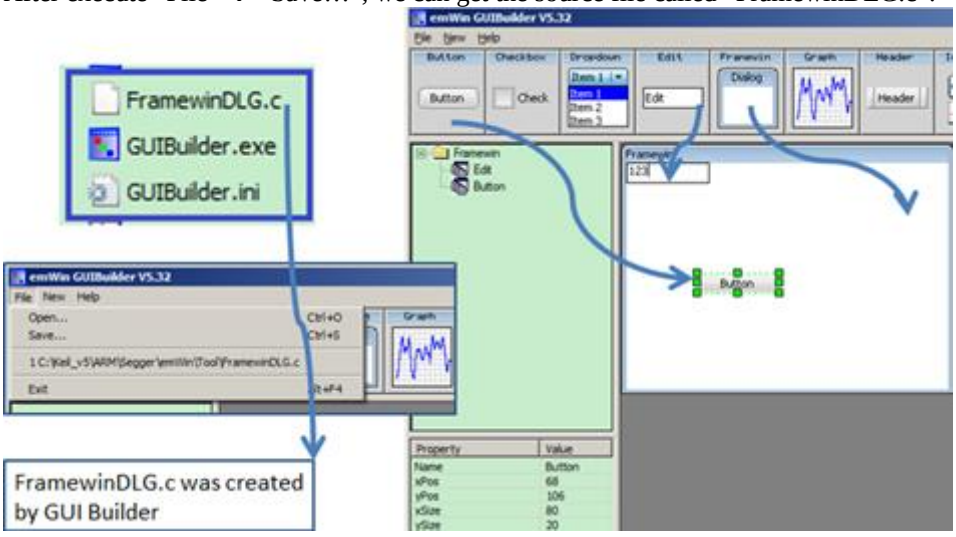

Figure 3.1-2 emWin GUIBuilder can generate a GUI layout and source file.

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:

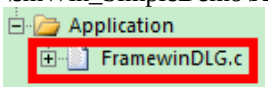

Figure 3.2-1 emWin GUI application source file.

```
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

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

\ThirdParty\emWin\Config\LCDConf.c

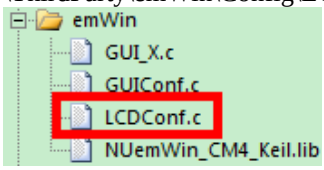

Figure 4.1-1 emWin display define.

```
/*********************************************************************
*
*       Layer configuration
*
**********************************************************************
*/
//
// Physical display size
//
#define XSIZE_PHYS 240
#define YSIZE_PHYS 320
```

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

```
//
// Orientation
//
Config.Orientation = GUI_MIRROR_X | GUI_MIRROR_Y | GUI_SWAP_XY;
GUIDRV_FlexColor_Config(pDevice, &Config);
//
// Set controller and operation mode
```

```
//
PortAPI.pfWrite16_A0  = LCD_WR_REG;
PortAPI.pfWrite16_A1  = LCD_WR_DATA;
PortAPI.pfWriteM16_A1 = LcdWriteDataMultiple;
PortAPI.pfReadM16_A1  = LcdReadDataMultiple;
GUIDRV_FlexColor_SetFunc(pDevice, &PortAPI, GUIDRV_FLEXCOLOR_F66709,
GUIDRV_FLEXCOLOR_M16C0B16);
```

```
/*-----------------------------------------------*/
// Write control registers of LCD module
//
/*-----------------------------------------------*/
void LCD_WR_REG(uint8_t cmd)
{
   EBI0_WRITE_DATA16(0x00030000, dat);
}
```

```
/*-----------------------------------------------*/
// Read data from SRAM of LCD module
//
/*-----------------------------------------------*/
uint8_t LCD_RD_DATA(void)
{
   return EBI0_READ_DATA16(0x00030000);
}
```

4.2. Step 2: BSP display

BSP ebi.c defines the EBI driver interface. In \ThirdParty\emWin\Config\LCDConf.c, we utilize EBI to configurate MPU-type LCD as an output device.

\Library\StdDriver\src\ebi.c

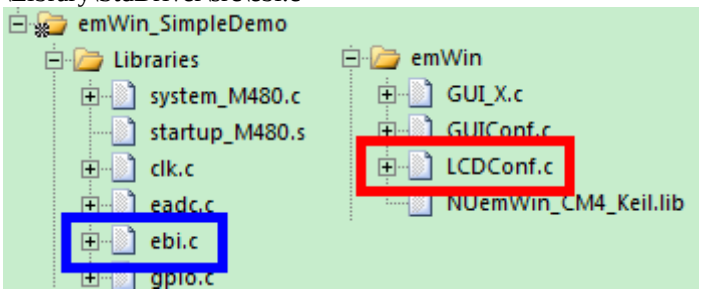

Figure 4.2-1 To utilize BSP EBI interface and emWin LCDConf to configurate the MPU-type LCD.

5. Revision History

Version	Date	Description
V1.00.002	July. 5, 2018	● Modify project “emWin_SimpleDemo” to store calibration results in APROM ● Add new project “emWin_SimpleDemo_SD” to store calibration results in SD card
V1.00.001	Mar. 30, 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.