

Oscilloscope GUI Designed by emWin

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code utilizes the SEGGER emWin GUIBuilder tool to create a graphical user interface (GUI) for an oscilloscope. It is designed to be compatible with the M467 series development platform to showcase a graphical user interface for an oscilloscope.
BSP Version	M460_Series_BSP_CMSIS_V3.00.001
Hardware	NuMaker-HMI-M467

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 example code demonstrates how to design the GUI (Graphical User Interface) for an oscilloscope project on the NuMaker-HMI-M467 evaluation board. It uses SEGGER emWin in development and designs interactive functions, such as Run/Stop button and Trigger button, Reset button, and Zoom in/out button. The application of EADC interrupts to capture analog signals, which are displayed on the screen. The sampling frequency of the EADC is set at 1 kHz and supports analog signal sources up to 3.3V. Additionally, the DAC outputs a 30Hz sine wave for validation.

1.1 UI Design

In this project, the graphical interface design uses SEGGER emWin GUIBuilder V6.30. The GUIBuilder IDE development is divided into different parts: Function window, Editor window, Hierarchic Tree, and Properties, as shown in Figure 1-1. Users can design the UI layout by dragging and dropping components from the Function window onto the Editor window.



Figure 1-1 emWin GUIBuilder IDE

The NuMaker-HMI-M467 evaluation board is equipped with the NuMaker-THT-LCD43 display, which has a window size of 480x272 pixels. Setting up the oscilloscope project involves using the Graph component to draw waveforms. Additionally, a Text element is added to display waveform frequency information. The Radio element provides the option to select between simulation and real-time modes for displaying the current screen. In simulation mode, a simulated sine wave output is generated, while in real-time mode, actual EADC data is sampled. The Buttons element are used to provide various functions. The Run/Stop button

can toggle between scrolling and freezing the oscilloscope screen. The Trigger button captures the first segment of the signal source, matching the size of the screen. The Reset button clears the data on the current page and resets all setting. The Zoom In/Out buttons allow for adjusting the waveform amplitude, providing up to three levels of adjustment.

Both the appearance and size of the element can be configured through the **Properties** section. The design of the GUI interface and its **properties** can be referred to Figure 1-3 to Figure 1-5.

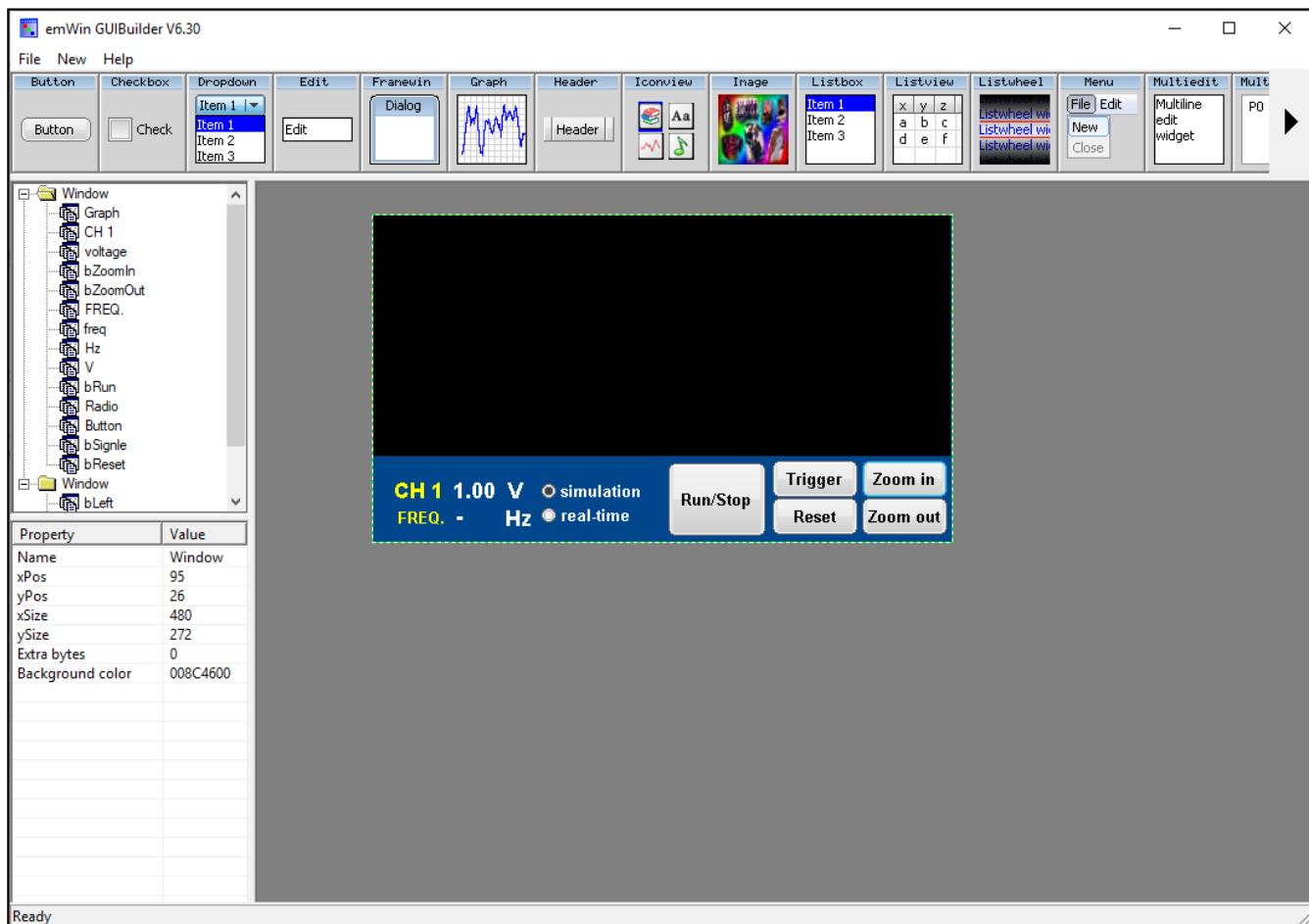


Figure 1-2 Oscilloscope Project Pattern

After file saving, the *FramewinDLG.c* file will be updated, and the same file will be replaced in the oscilloscope project. This section completes the preliminary UI design. In this example file, the *FramewinDLG.c* is located in the Application folder. It contains the pre-designed UI framework, the button events and other settings. Users can customize their own GUI design based on this template.

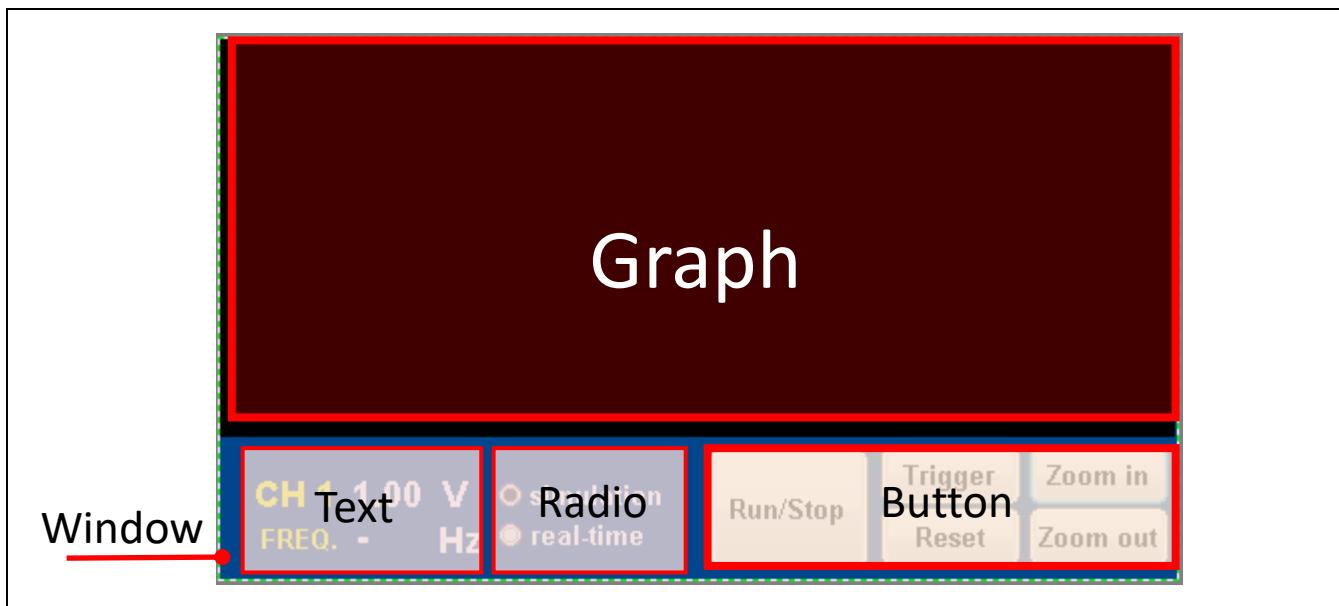


Figure 1-3 Oscilloscope Element Planning And Configuration

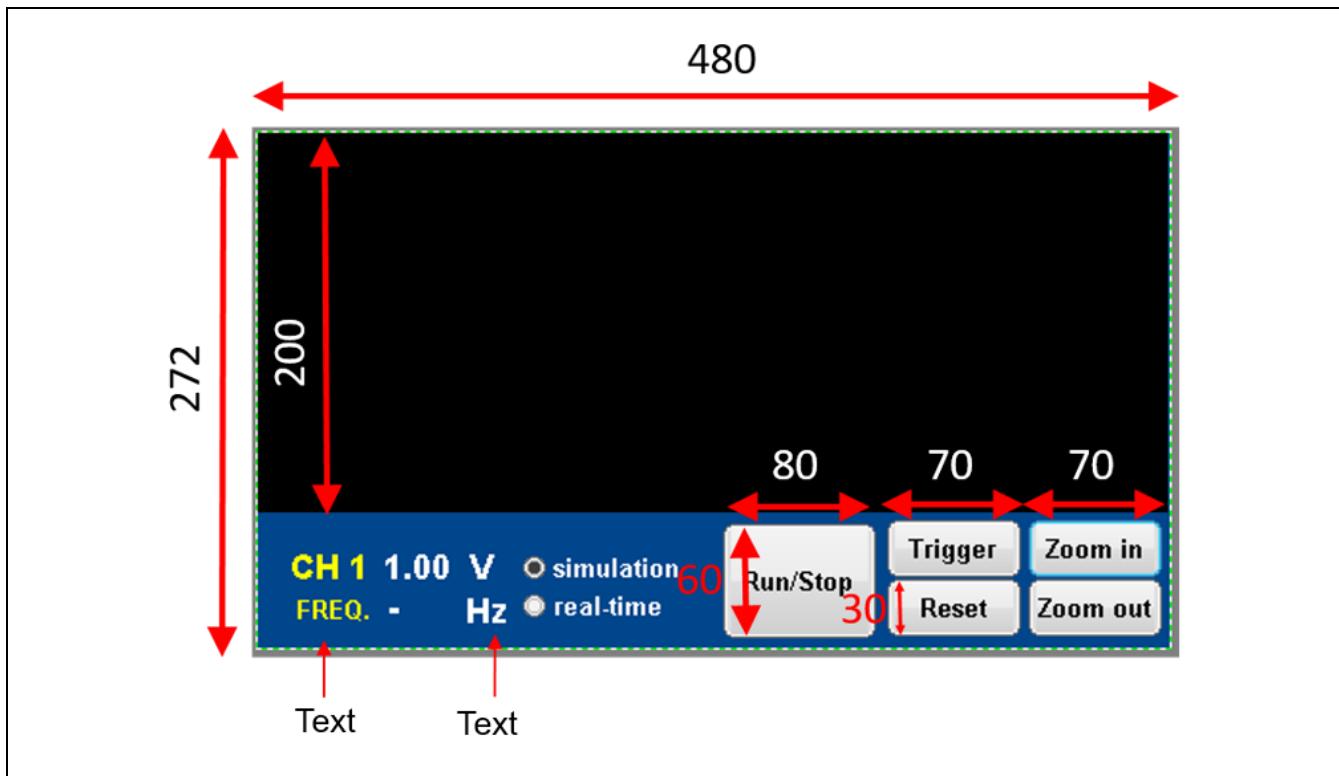


Figure 1-4 Oscilloscope Element Size

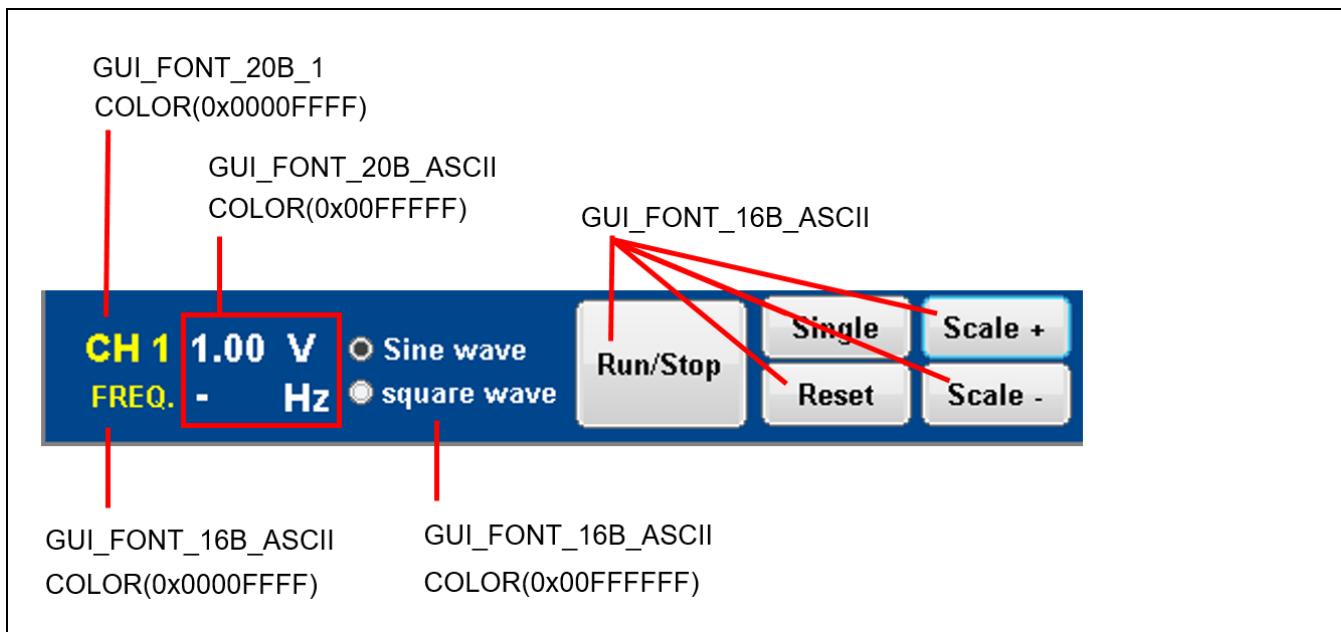


Figure 1-5 Oscilloscope Element Font Color

1.2 UI Code Description

The software framework diagram is shown in Figure 1-6.

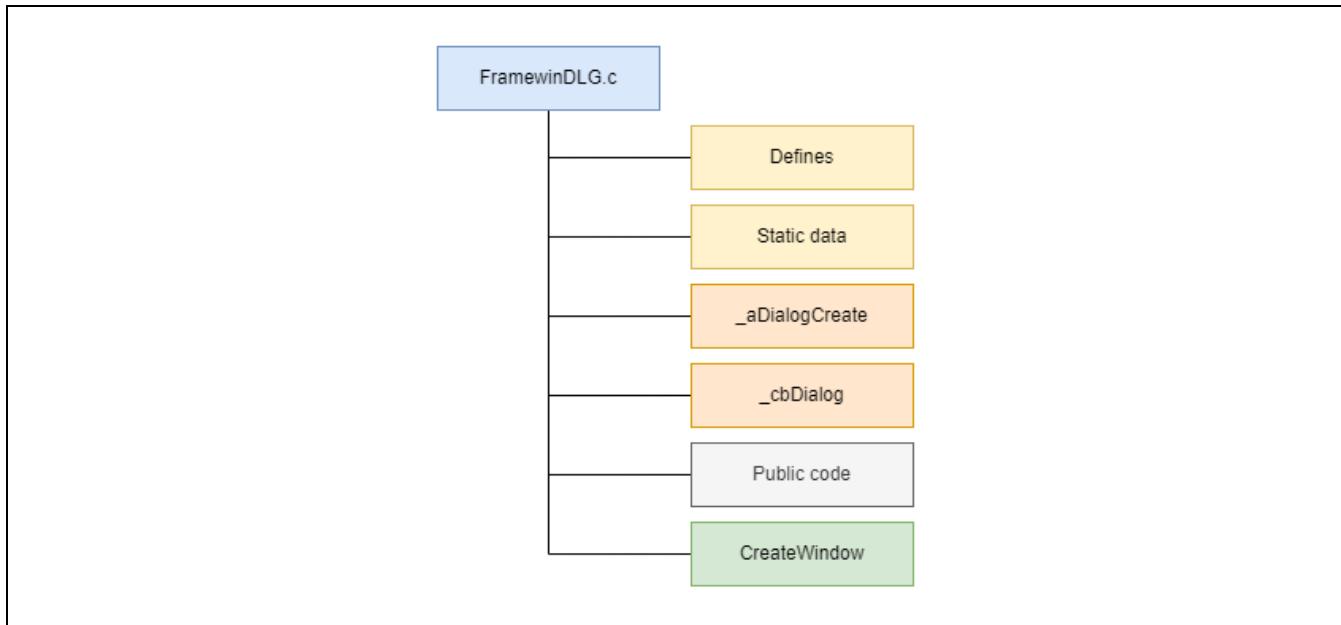


Figure 1-6 Software Framework Diagram

The **_aDialogCreate** part of *FramewinDLG.c* sets the appearance of the element through the API and finds the position of the GUIBuilder in the editor:

```
*****  
*  
*   _aDialogCreate  
*/  
  
static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] =  
{  
    { WINDOW_CreateIndirect, "Window", ID_WINDOW_0, 0, 0, 480, 272, 0, 0x0, 0 },  
    { GRAPH_CreateIndirect, "Graph", ID_GRAPH_0, 0, 0, 480, 200, 0, 0x0, 0 },  
    ...  
    { BUTTON_CreateIndirect, "Button", ID_BUTTON_3, 318, 314, 80, 30, 0, 0x0, 0 },  
    { BUTTON_CreateIndirect, "bSingle", ID_BUTTON_4, 331, 204, 70, 30, 0, 0x0, 0 },  
    { BUTTON_CreateIndirect, "bReset", ID_BUTTON_5, 331, 235, 70, 30, 0, 0x0, 0 },  
};
```

The **_cbDialog** part of *FramewinDLG.c* handles GUI events, and the GRAPH widget sets the configuration of grid lines:

```
*****  
*  
*   _cbDialog  
*/  
  
static void _cbDialog(WM_MESSAGE * pMsg)  
{  
    ...  
    switch (pMsg->MsgId)  
    {  
  
        case WM_INIT_DIALOG:  
            // Initialization of 'Window'  
            hItem = pMsg->hWin;  
            WINDOW_SetBkColor(hItem, GUI_MAKE_COLOR(0x008C4600));  
  
            // Enable multi-buffering  
            WM_MULTIBUF_Enable(1);  
  
            ...  
  
            // Initialize the GRAPH widget  
            hItem = WM_GetDialogItem(pMsg->hWin, ID_GRAPH_0);  
  
            // Set the appearance and grid properties of the graph  
            GRAPH_SetBorder(hItem, 1,1,1,1);  
            GRAPH_SetColor(hItem, GUI_BLACK, GRAPH_CI_BK);  
            GRAPH_SetColor(hItem, GUI_GRAY_E7, GRAPH_CI_BORDER);  
            GRAPH_SetColor(hItem, GUI_GRAY_E7, GRAPH_CI_FRAME);  
            GRAPH_SetColor(hItem, GUI_WHITE, GRAPH_CI_GRID);  
            GRAPH_SetGridVis(hItem, 1);  
            GRAPH_SetGridDistX(hItem, 50);  
            GRAPH_SetGridDistY(hItem, 25);  
            GRAPH_SetLineStyleH(hItem, GUI_LS_DOT);  
            GRAPH_SetLineStyleV(hItem, GUI_LS_DOT);  
  
            // Create and configure a vertical scale, then attach it to the graph  
            hScaleHandle = GRAPH_SCALE_Create(15, GUI_TA_HCENTER | GUI_TA_VCENTER,  
GRAPH_SCALE_CF_VERTICAL, 1);  
            GRAPH_SCALE_SetFont(hScaleHandle, &GUI_Font8x8);  
            GRAPH_SCALE_SetTextColor(hScaleHandle, GUI_BLACK);
```

```
GRAPH_SCALE_SetNumDecs(hScaleHandle, 0);
GRAPH_SCALE_SetTickDist(hScaleHandle, 20);
GRAPH_SCALE_SetOff(hScaleHandle, 100);
GRAPH_AttachScale(hItem,hScaleHandle);

// Introduce a delay of 500 milliseconds
GUI_Delay(500);

// Create and configure graph data, then attach it to the graph
hGraphData = GRAPH_DATA_YT_Create(GUI_YELLOW,LCD_GetXSize(),DrawData,0);
GRAPH_DATA_YT_SetAlign(hGraphData, GRAPH_ALIGN_RIGHT);
GRAPH_DATA_YT_SetOffY(hGraphData, 100);
GRAPH_AttachData(hItem, hGraphData);

// Set a custom callback, here we want to draw a cursor
// WM_SetCallback(hItem, _cbGraph);
break;

// event
//
case WM_NOTIFY_PARENT:
    // Retrieve the ID and notification code of the event sender
    Id      = WM_GetId(pMsg->hWinSrc);
    NCode  = pMsg->Data.v;

    switch(Id)
    {
        case ID_BUTTON_0: // Notifications sent by 'bZoomIn'
            switch(NCode)
            {
                case WM_NOTIFICATION_CLICKED:
                    // Clear existing graph data
                    GRAPH_DATA_YT_Clear(hGraphData);

                    // Increment the 'multiple' factor for zooming in
                    multiple++;
                    if(multiple > 2) multiple=2;

                    // Update graph data with adjusted values
                    for(i=0; i<480; i++)
                    {
                        GRAPH_DATA_YT_AddValue(hGraphData, DrawData[i] *
multipleArray[multiple]);
                    }
                    break;
                case WM_NOTIFICATION_RELEASED:
                    break;
            }
            break;

        case ID_BUTTON_1: // Notifications sent by 'bZoomOut'
            switch(NCode)
            {
                case WM_NOTIFICATION_CLICKED:
                    // Clear existing graph data
                    GRAPH_DATA_YT_Clear(hGraphData);

                    if(multiple == 0)
                        multiple=0;
                    else
                        multiple--;

                    for(i=0; i<480; i++)

```

```
multipleArray[multiple]);
{
    GRAPH_DATA_YT_AddValue(hGraphData, DrawData[i] *
multipleArray[multiple]);
}
break;

case WM_NOTIFICATION_RELEASED:
    // USER START (Optionally insert code for reacting on
notification message)
    break;
}
break;
...
}

}
break;

...
}
```

CreateFramewin() is used to create a dialog and set up multi-buffering:

```
*****
*
*   CreateWindow
*/
WM_HWIN CreateFramewin(void)
{

    WM_HWIN hWin;

    // Enable multi-buffering for the window
    WM_MULTIBUF_Enable(1);
    // Create a dialog box (FrameWin) using the specified dialog creation parameters
    hWin = GUI_CreateDialogBox(_aDialogCreate, GUI_COUNTOF(_aDialogCreate), _cbDialog,
WM_HBKWIN, 0, 0);

    return hWin;
}
```

2. Code Description

The main program begins in the *main.c* file, which initializes the system first. After configuring the touch screen, emWin starts displaying the GUI by executing the **MainTask()** function.

Oscilloscope_init() will start the oscilloscope to capture the EADC data where the sampling rate is about 1 kHz, output a sine wave, and update the data on the GUI.

```
/*
 * Main Function
 */
int main(void)
{
    /* init */
    SYS_Init();

    // Initialize the touch functionality.
    g_enable_Touch = 0;
    tmr_init();
    st1633i_low_level_init();
    st1633i_startup_sequence();
    g_enable_Touch = 1;

    // Print a message indicating the CPU frequency.
    printf("[M]Oscilloscope Demo - CPU @ %d Hz \r\n", SystemCoreClock);

    /* Main Loop */
    MainTask();

    // This loop will keep the program running indefinitely.
    while(1) {}

    return 0;
}
```

```
void MainTask(void)
{
    // Initialize variables
    int i=0;
    int32_t bSingleCount = 0;
    int32_t bSingleFlag = 0;

    // Create a window handle
    WM_HWIN hWin;
    printf("Main Task -> \n");

    // Enable memory device for GUI rendering
    WM_SetCreateFlags(WM_CF_MEMDEV);

    // Initialize the GUI
    GUI_Init();
    WM_MULTIBUF_Enable(1); // Enable multi-buffering for better performance
    hWin = CreateFrameWin(); // Create a window using CreateFrameWin function

    // Initialize the Oscilloscope
    Oscilloscope_init();
```

```
// Enter the main loop of the task
while (1)
{
    GUI_Delay(1);
    ...
}
```

The initialization in `Oscilloscope.c` sets EADC (input pin is PB0), Timer1, Timer2, EPWM (output pin is PC12) and DAC (output pin is PB12). The screen refresh rate is also refreshed once at 100Hz.

```
*****  
/* Oscilloscope init */  
/*  Use EADC  -> TMR1 1kHz */  
/*  DAC   -> TMR2 2kHz */  
/*  EPWM1 -> 30 Hz */  
*****  
void Oscilloscope_init(void)  
{  
    SYS_UnlockReg();  
  
    /* Enable module clock */  
    CLK_EnableModuleClock(EADC0_MODULE);  
    CLK_EnableModuleClock(TMR1_MODULE);  
    CLK_EnableModuleClock(TMR2_MODULE);  
    CLK_EnableModuleClock(EPWM1_MODULE);  
    CLK_EnableModuleClock(DAC_MODULE);  
  
    /* Set module clock */  
    CLK_SetModuleClock(EADC0_MODULE, CLK_CLKSEL0_EADC0SEL_PLL_DIV2,  
CLK_CLKDIV0_EADC0(12));  
    CLK_SetModuleClock(EPWM1_MODULE, CLK_CLKSEL2_EPWM1SEL_PCLK1, 0);  
    CLK_SetModuleClock(TMR1_MODULE, CLK_CLKSEL1_TMR1SEL_HIRC, NULL);  
    CLK_SetModuleClock(TMR2_MODULE, CLK_CLKSEL1_TMR2SEL_HIRC, 0);  
  
    /* EADC0 channels. */  
    SET_EADC0_CH0_PB0();  
    SET_DAC0_OUT_PB12();  
  
    /* Disable digital input path of EADC analog pin to prevent leakage */  
    GPIO_DISABLE_DIGITAL_PATH(PB, BIT0);  
  
    /* EPWM channels. */  
    SET_EPWM1_CH0_PC12();  
    EPWM_ConfigOutputChannel(EPWM1, 0, 30, 50);  
    EPWM_EnableOutput(EPWM1, 0x3F);  
    EPWM_Start(EPWM1, 0x3F);  
  
    /* module init */  
    TMR1_Init();  
    EADC_Init();  
    DAC_Init();  
    SYS_LockReg();  
}
```

To modify the EADC sampling time, adjust the settings in Timer1. Currently, the sampling frequency is set to 1 kHz and triggers the EADC to interrupt the measurement.

```
*****  
/* TIMER1 init */  
*****  
  
void TMR1_Init(void)  
{  
    TIMER_Open(TIMER1, TIMER_PERIODIC_MODE, 1000);  
    TIMER_SetTriggerTarget(TIMER1, TIMER_TRG_TO_EADC);  
  
    // Start Timer 1  
    TIMER_Start(TIMER1);  
}
```

```
*****  
/* EADC init */  
*****  
  
void EADC_Init(void)  
{  
    int32_t i32Err;  
  
    i32Err = EADC_Open(EADC0, EADC_CTL_DIFFEN_SINGLE_END);  
  
    ... ...  
  
    /* Configure the sample module 0 for analog input channel 0 and enable Timer1 trigger source */  
    EADC_ConfigSampleModule(EADC0, 0, EADC_TIMER1_TRIGGER, 0);  
  
    /* Set sample module external sampling time to 0 */  
    EADC_SetExtendSampleTime(EADC0, 0, 0);  
  
    /* Enable Accumulate feature */  
    EADC_ENABLE_ACU(EADC0, 0, EADC_MCTL1_ACU_32);  
  
    /* Enable Average feature */  
    EADC_ENABLE_AVG(EADC0, 0);  
  
    /* Clear the A/D ADINT0 interrupt flag for safe */  
    EADC_CLR_INT_FLAG(EADC0, EADC_STATUS2_ADIF0_Msk);  
  
    /* Enable the sample module interrupt. */  
    EADC_ENABLE_INT(EADC0, BIT0);  
    EADC_ENABLE_SAMPLE_MODULE_INT(EADC0, 0, BIT0);  
    NVIC_EnableIRQ(EADC00 IRQn);  
  
    /* Reset the ADC interrupt indicator and trigger sample module to start A/D conversion */  
    EADC_START_CONV(EADC0, BIT0);  
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
 - M460_Series_BSP_CMSIS_V3.00.001
- IDE version
 - Keil uVersion 5.36

3.2 Hardware Requirements

- Circuit components
 - NuMaker-HMI-M467HJ V1.1
- Connect NuMaker-HMI-M467 and PC
 - NuMaker-HMI-M467HJ connects to PC via USB cable, as shown in Figure 3-1.
 - Input pin: EADC pin is PB0. Please use DuPont wires to connect to PB12 or PC12.
 - Output pin: DAC pin is PB12; EPWM pin is PC12.

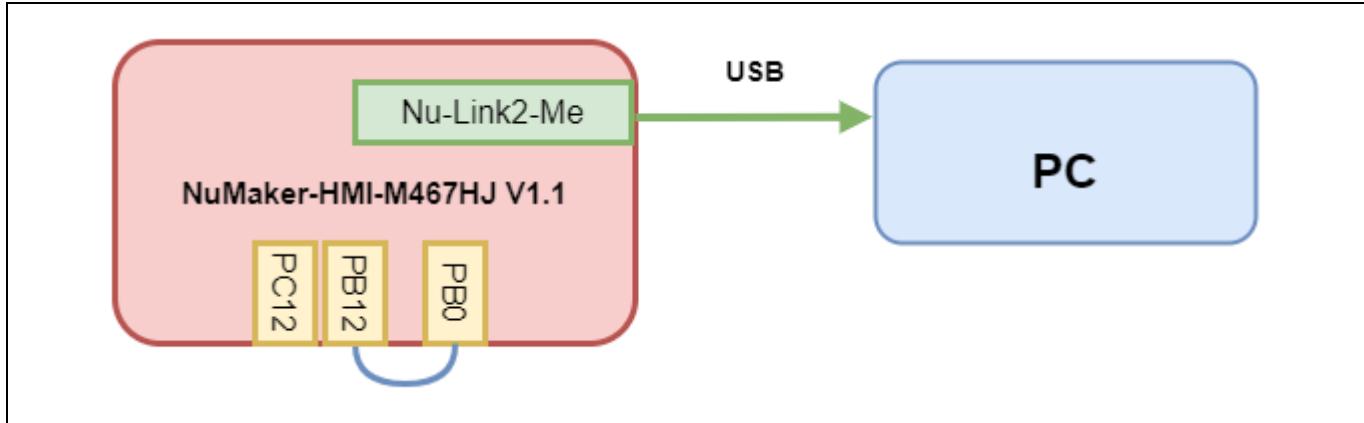


Figure 3-1 Connection Diagram of Numaker-HMI-M467HJ and PC

4. Directory Information

The directory structure is shown below.

EC_M467_emWin_Oscilloscope_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
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	
emWin_Oscilloscope	Source file of example code
ThirdParty	Header files for emWin project
Tool	Tool : GUIBuilder.exe

Figure 4-1 Directory Structure

5. Example Code Execution

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

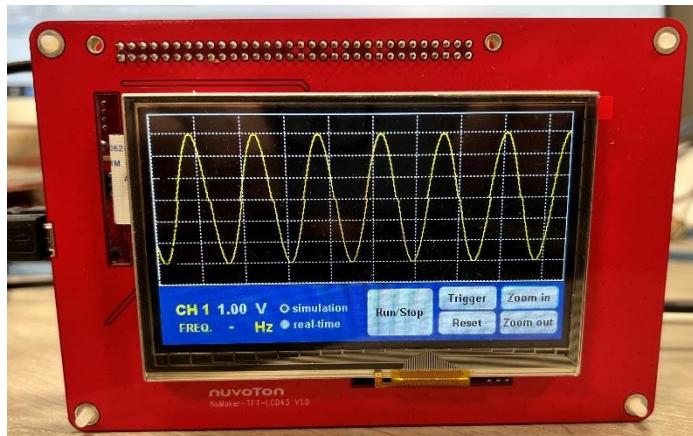


Figure 5-1 NuMaker-HMI-M467HJ Execution Result

6. Revision History

Date	Revision	Description
2023.08.13	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.