M480 LCD with Touch

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This example code uses M480 External Bus Interface (EBI) hardware to drive i80-mode LCD module for display and scans the ADC input to recognize the finger position on the touch screen.
BSP Version	M480 Series BSP CMSIS V3.03.001
Hardware	NuMaker-M487D board (NuMaker-PFM-M487 V3.0 board + NuMaker-M487-Advance V4.0 board)

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Function Description

1.1 Introduction

This example code utilizes the NuMaker-M487D board based on the M487 series as the main controller to develop the display and touch application of the resistive-touch LCD module, which uses the External Bus Interface (EBI) and three GPIOs to drive the LCD module for display, four GPIO pins (two of which are ADC channels) to scan periodically the touch position and instantly displays the touch position on the LCD screen, and a GPIO pin to drive the LED to display the status when touched.

The hardware NuMaker-M487D consists of two boards, NuMaker-PFM-M487 V3.0 board and NuMaker-M487-Advacne V4.0 board. For more information on the NuMaker-M487D board, please visit our Nuvoton website to download the User Manual UM_NuMaker-PFM-M487_User_Manual_EN and UM_NuMaker_M487_Advance_User_Manual_EN documents.

https://www.nuvoton.com/hq/products/microcontrollers/arm-cortex-m4-mcus/User-Manual/?__locale=en&resourcePage=Y&category=&pageIndex=3

NuMaker-M487D Board

1.2 Principle

M487 MCU utilizes the following peripherals and GPIO pins to drive the LCD panel module:

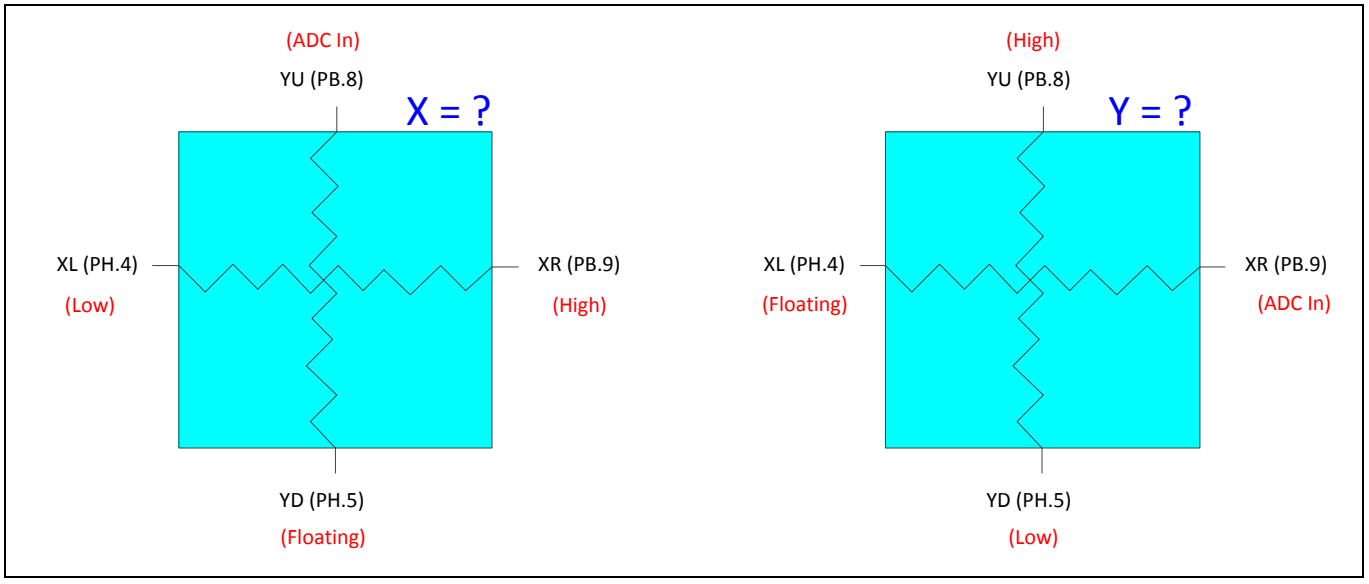
1) Sixteen EBI Address/Data pins (EBI_AD0 ~ EBI_AD15) are used as the transmission of commands and data.
2) Three EBI control pins (EBI_nCS0, EBI_nWR, EBI_nRD) as control signals for chip selection, writing and reading.

3) One GPIO pin (LCD_RS) indicates the current data on the EBI_AD0 ~ EBI_AD15 bus is a command or data.
4) One GPIO pin (LCD_RST) controls the reset of the LCD module.
5) One GPIO pin (LCD_BL) controls the backlight of the LCD module.

For the resistive-touch scanning, M487 MCU uses four GPIO pins (PB.8, PB.9, PH.4 and PH.5):

1) For detecting the x or y position of finger, two of these four pins need to be forced on High and Low states, one is be forced on floating state and the last one use the ADC function to convert the input resistance voltage value of the resistive-touch LCD screen. Finally, the firmware code calculates the x or y position of the touch finger on LCD screen.

The following figure shows these four used GPIO pins and the resistive-touch LCD screen that be simplified as a divider resistor.



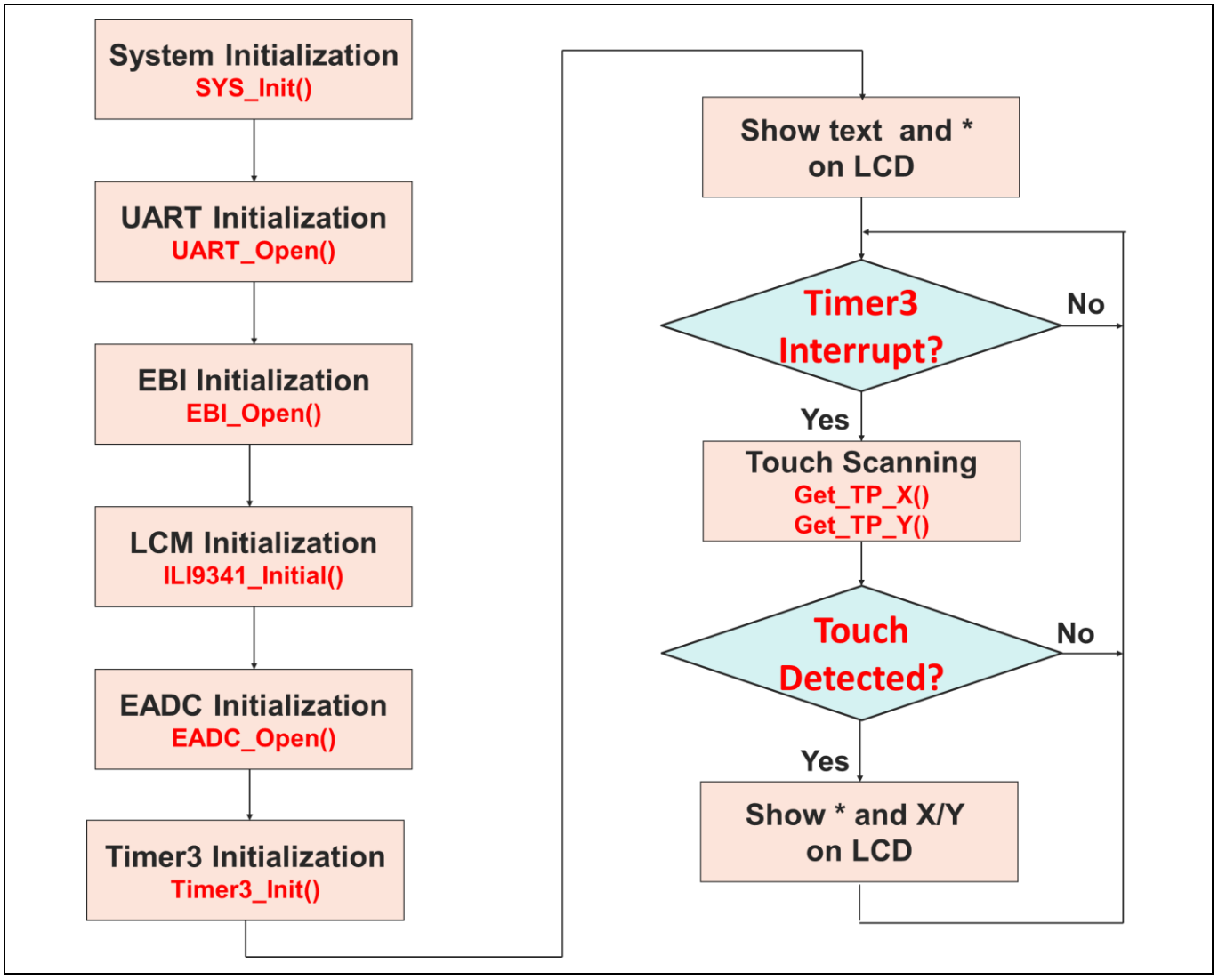
Used Pins and Simplified Resistive-Touch LCD Screen Diagram

1.3 Demo Result

After executed this example code, user can touch the LCD screen and the x and y position of touch finger will be shown on the corner of LCD screen.

2 Code Description

The main() function flow chart is shown below.



Main() Flow Chart

The position scanning of the resistive-touch LCD screen uses four GPIO pins of the M487 MCU, namely PB.8, PB.9, PH.4 and PH.5. Among them, PB.8 and PB.9 can be set as the ADC input function and need to be dynamically switched to GPIO or ADC input in a timely manner.

In the X-axis scanning, the firmware code needs to force XR (PB.9) on High state, XL (PH.4) on Low state, YD (PH.5) on floating state, and YU (PB.8) to be as ADC input function. When the finger touches the touch screen, it will cause the voltage changed on the X-axis divider resistor. This voltage value is converted to a 12-bit value via the ADC input, and then it will be mapped to a relative position on the X-axis of LCD screen.

```
/**
 * @brief       Get X position from touch panel thru the ADC input
 *
 * @param       None
 *
 * @return      The X position on LCD screen
 *
 * @details     To get the X position when finger touching on the LCD screen
 */
uint16_t Get_TP_X(void)
{
    uint16_t    x_adc_in;
    uint16_t    X_pos;

    /*=== Get X from ADC input ===*/
    PB9 = 1;
    PH4 = 0;
    GPIO_SetMode(PB, BIT9, GPIO_MODE_OUTPUT);   // XR
    GPIO_SetMode(PH, BIT4, GPIO_MODE_OUTPUT);   // XL
    GPIO_SetMode(PH, BIT5, GPIO_MODE_INPUT);    // YD

    /* Configure the GPB8 ADC analog input pins. */
    SYS->GPB_MFPH &= ~(SYS_GPB_MFPH_PB8MFP_Msk | SYS_GPB_MFPH_PB9MFP_Msk);
    SYS->GPB_MFPH |= SYS_GPB_MFPH_PB8MFP_EADC0_CH8;

    /* Disable the GPB8 digital input path to avoid the leakage current. */
    GPIO_DISABLE_DIGITAL_PATH(PB, BIT8);

    /* Configure the sample module 1 for analog input channel 8 and software trigger
source.*/
    EADC_ConfigSampleModule(EADC, 1, EADC_SOFTWARE_TRIGGER, 8); // YU

    /* Clear the A/D ADINT1 interrupt flag for safe */
    EADC_CLR_INT_FLAG(EADC, EADC_STATUS2_ADIF1_Msk);

    /* Enable the sample module 1 interrupt. */
    EADC_ENABLE_INT(EADC, BIT1);    //Enable sample module A/D ADINT1 interrupt.
    EADC_ENABLE_SAMPLE_MODULE_INT(EADC, 1, BIT1);    //Enable sample module 1 interrupt.
    NVIC_EnableIRQ(ADC1_IRQn);
```

```
    /* Reset the ADC interrupt indicator and trigger sample module 1 to start A/D
conversion */
    g_u32AdcIntFlag_TP = 0;
    EADC_START_CONV(EADC, BIT1);

    /* Wait ADC interrupt (g_u32AdcIntFlag_TP will be set at IRQ_Handler function) */
    while(g_u32AdcIntFlag_TP == 0) {};
    x_adc_in = EADC_GET_CONV_DATA(EADC, 1)>>2;

    /*=== Calculate the X position ===*/
    X_pos = (x_adc_in - 170)/2.8;

    if(X_pos >= (LCD_W - 1)) X_pos = LCD_W - 1;

    printf("Position X: %d\n", X_pos);
    return X_pos;

}
```

In the Y-axis scanning, the firmware code needs to force YU (PB.8) on High state, YD (PH.5) on Low state, XL (PH.4) on floating state, and XR (PB.9) to be as ADC input function. When the finger touches the touch screen, it will cause the voltage changed on the Y-axis divider resistor. This voltage value is converted to a 12-bit value via the ADC input, and then it will be mapped to a relative position on the Y-axis of LCD screen.

```
/**
 * @brief       Get Y position from touch panel thru the ADC input
 *
 * @param       None
 *
 * @return      The Y position on LCD screen
 *
 * @details     To get the Y position when finger touching on the LCD screen
 */
uint16_t Get_TP_Y(void)
{
    uint16_t    y_adc_in;
    uint16_t    Y_pos;

    /*=== Get Y from ADC input ===*/
    PB8 = 1;
    PH5 = 0;
```

```
    GPIO_SetMode(PB, BIT8, GPIO_MODE_OUTPUT);    // YU
    GPIO_SetMode(PH, BIT5, GPIO_MODE_OUTPUT);    // YD
    GPIO_SetMode(PH, BIT4, GPIO_MODE_INPUT);     // XL

    /* Configure the GPB9 ADC analog input pins. */
    SYS->GPB_MFPH &= ~(SYS_GPB_MFPH_PB8MFP_Msk | SYS_GPB_MFPH_PB9MFP_Msk);
    SYS->GPB_MFPH |= SYS_GPB_MFPH_PB9MFP_EADC0_CH9;

    /* Disable the GPB9 digital input path to avoid the leakage current. */
    GPIO_DISABLE_DIGITAL_PATH(PB, BIT9);

    /* Configure the sample module 2 for analog input channel 9 and software trigger
source.*/
    EADC_ConfigSampleModule(EADC, 2, EADC_SOFTWARE_TRIGGER, 9); // XR

    /* Clear the A/D ADINT1 interrupt flag for safe */
    EADC_CLR_INT_FLAG(EADC, EADC_STATUS2_ADIF1_Msk);

    /* Enable the sample module 2 interrupt. */
    EADC_ENABLE_INT(EADC, BIT2);    //Enable sample module A/D ADINT1 interrupt.
    EADC_ENABLE_SAMPLE_MODULE_INT(EADC, 1, BIT2);    //Enable sample module 2 interrupt.
    NVIC_EnableIRQ(ADC1_IRQn);

    /* Reset the ADC interrupt indicator and trigger sample module 2 to start A/D
conversion */
    g_u32AdcIntFlag_TP = 0;
    EADC_START_CONV(EADC, BIT2);

    /* Wait ADC interrupt (g_u32AdcIntFlag_TP will be set at IRQ_Handler function) */
    while(g_u32AdcIntFlag_TP == 0) {};
    y_adc_in = EADC_GET_CONV_DATA(EADC, 2)>>2;

    /*=== Calculate the Y position ===*/
    Y_pos = (y_adc_in - 150)/2.34;

    if(Y_pos >= (LCD_H - 1)) Y_pos = LCD_H - 1;

    printf("Position Y : %d \n", Y_pos);
    return Y_pos;

}
```

This ADC01_IRQHandler() function is the interrupt service routine of ADC01 and it will clear the interrupt event ADIF1 flag and set the g_u32AdcIntFlag_TP flag to 1.

```
/**
 * @brief       ADC01 IRQ handler
 *
 * @param       None
 *
 * @return      None
 *
 * @details     The ADC01 default IRQ, declared in startup_M480.s
 */
void ADC01_IRQHandler(void)
{
    /* Clear the A/D ADINT1 interrupt flag */
    EADC_CLR_INT_FLAG(EADC, EADC_STATUS2_ADIF1_Msk);

    g_u32AdcIntFlag_TP = 1;

}
```

3 Software and Hardware Environment

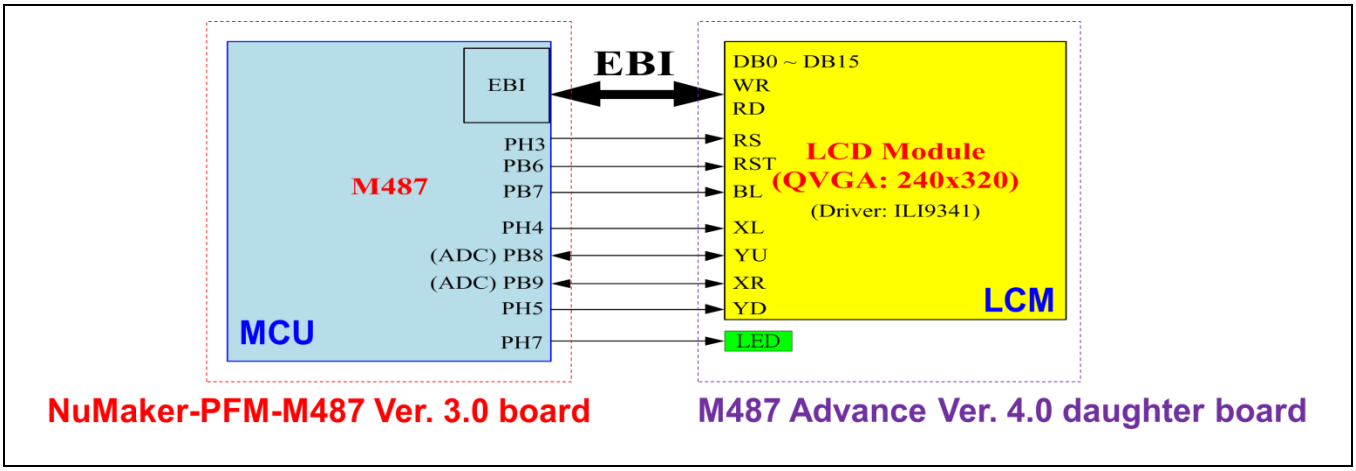
Software Environment

- BSP version
  - M480 Series BSP CMSIS V3.03.001
- IDE version
  - Keil uVersion 4.74

Hardware Environment

- Circuit components
  - NuMaker-M487D board
- Diagram

The following figure shows the hardware block diagram of the M487 with the resistive LCD touch screen module and the related peripherals and GPIO pins used in this example.



Hardware Block Diagram

4 Directory Information

- EC_M480_LCD_with_Touch_V1.00
  - Library — Sample code header and source code files
    - CMSIS — Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
    - Device — CMSIS compliant device header files
    - StdDriver — All peripheral driver header and source code files
  - SampleCode
    - ExampleCode — Source code files of this example code

5 How to Execute Example Code

1. Browsing into the sample code folder by Directory Information (section 4) and double click this sample code project M480_LCD_with_Touch.uvproj.
2. Enter Keil compile mode.
   a. Build.
   b. Download.
   c. Start/Stop debug session.
3. Enter debug mode.
   a. Run.

6 Revision History

Date	Revision	Description
July 31, 2019	1.00	1. Initially issued.

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.