M2L31 Internal Temperature Sensor

NuMicro® 32 位元系列微控制器範例程式碼介紹

文件資訊

應用簡述	本範例程式碼讀取 M2L31 的內部溫度感測器數值，並顯示在 GUI。
BSP 版本	M2L31_Series_BSP_CMSIS_V3.03.000
開發平台	NuMaker-M2L31SG_V1.1

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. 概述

本範例程式碼展示如何讀取 NuMicro® M2L31 系列微控制器的內部溫度感測器。系統啟動時，會先從 RMC Flash 讀取出廠燒錄的 25°C 與 125°C 兩點校正資料，隨後由 Timer0 每秒觸發一次量測。量測時分別透過 EADC 通道 28 量測帶隙電壓（Bandgap Voltage）以計算實際 AV_{DD} 電壓，並透過 EADC 通道 29 量測溫度感測器電壓。兩組數值經中值濾波濾除雜訊後，會套用兩點線性插值公式換算為攝氏溫度，最後打包成自定義 UART 封包，以 115200 bps 傳送至 PC 端 UI 軟體，供使用者即時觀看溫度曲線。

1.1 原理

本範例首先透過 RMC ISP 指令，讀取 M2L31 晶片出廠時的兩點溫度校正值，藉此決定溫度基底斜率。隨後在執行時，依序透過 EADC 通道 28 與 29 擷取內部帶隙電壓與溫度感測器訊號，並套用 10 筆資料的中值濾波器（Median Filter）以剔除突波雜訊；接著利用該資料反推出精準的動態 AV_{DD} 電源電壓，並透過雙點線性插值公式，計算出精確至小數點後兩位的溫度定點數。最終，系統將量測所得的電壓與溫度資料封裝於自定義通訊封包中。該封包長度為 7 個位元組，並以 0x49 作為封包標頭（Header）、0x19 作為封包結尾（Trailer），其格式如圖 1-1 所示，透過 UART0 以 115200 bps 的速率即時推送到 PC 端 UI，以繪製出穩定的監測曲線。

帶隙電壓計算：

$$AV_{DD} = \frac{3072 \times Bandgap_{factory}}{ADC_{measured}} \ (\mathrm{mV})$$

攝氏溫度計算：

$$T = \frac{V_{curr} - V_{25^\circ C}}{V_{125^\circ C} - V_{25^\circ C}} \times (125 - 25) + 25$$

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Header	Length	AVDD High	AVDD Low	Temp High	Temp Low	Trailer
0x49	0x05	A_{VDD} High Byte	A_{VDD} Low Byte	Temp High Byte	Temp Low Byte	0x19

圖 1-1 UART 資料輸出

1.2 執行結果

本範例程式碼支援透過電腦 UI 進行即時資料監控(如圖 1-2)。操作時，首先於「Serial Port」區塊選擇正確的 COM 埠並建立通訊；連線成功後，UI 介面將即時解析並顯示由 EADC 量測出的 AV_{DD} 電壓與精確溫度，並於下方動態繪製出平滑的溫度監測折線圖。最後，使用者可透過點選「Log」區塊的 Start 按鈕，將含有精確時間戳記、電壓與溫度的歷史資料，自動且連續地寫入並匯出為.csv 檔案，以利後續進行長期的穩定性測試與資料分析。

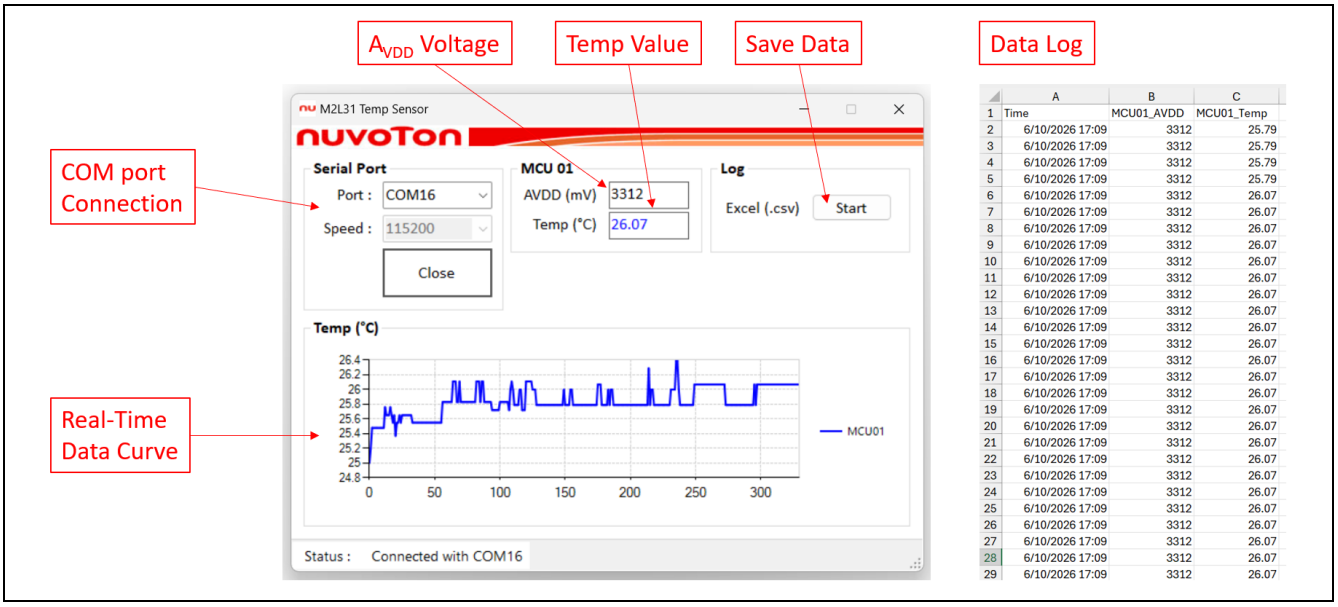


圖 1-2 UI 連線說明

1.3 同時量測 80 顆 M2L31 內部溫度感測器的誤差大小

本實驗以高精度的 TMP117 溫度感測器數值作為基準標準（Golden Sample），在 -40°C 至 125°C 的寬溫範圍內，同時對大量（共 80 顆）M2L31 晶片進行內部溫度感測器的誤差量測。實驗結果顯示（如圖 1-3、圖 1-4、圖 1-5、圖 1-6、圖 1-7 所示），各晶片在全溫區的量測誤差曲線皆呈現穩定的弧形飄移趨勢，且整體的溫度誤差全數成功控制在 ±5°C 的範圍以內。這證實了透過出廠兩點校正與動態 AV_{DD} 電源補償後，該晶片內建的溫度感測器在極端溫度環境下仍具備極高的準確度與優異的可靠性。

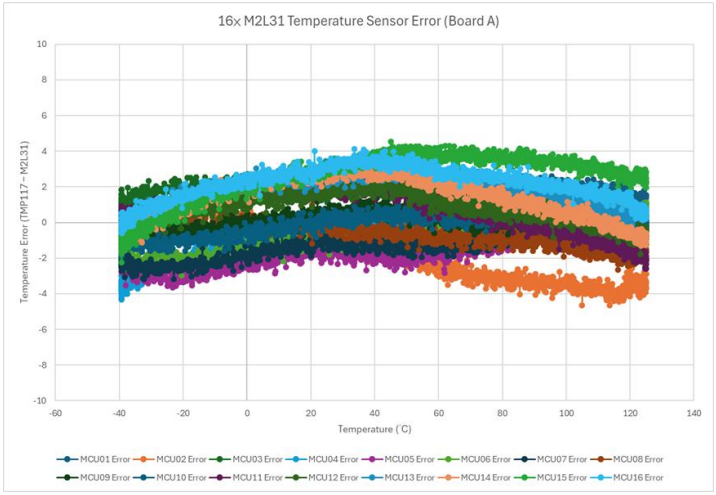


圖 1-3 16 顆 M2L31 溫度數值誤差(Board A)

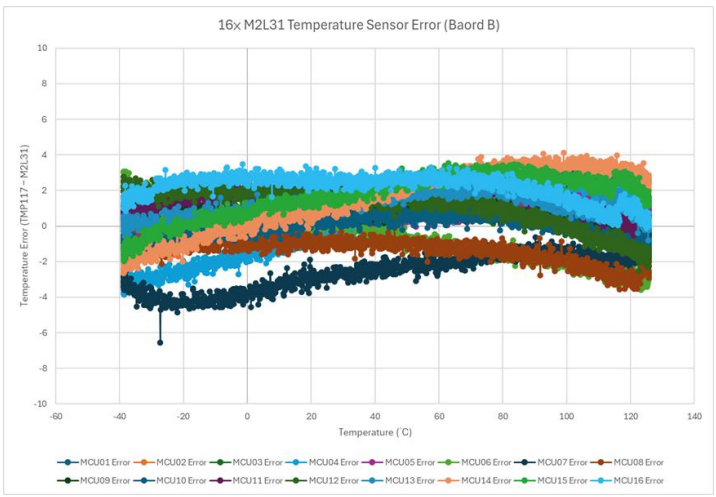


圖 1-4 16 顆 M2L31 溫度數值誤差(Board B)

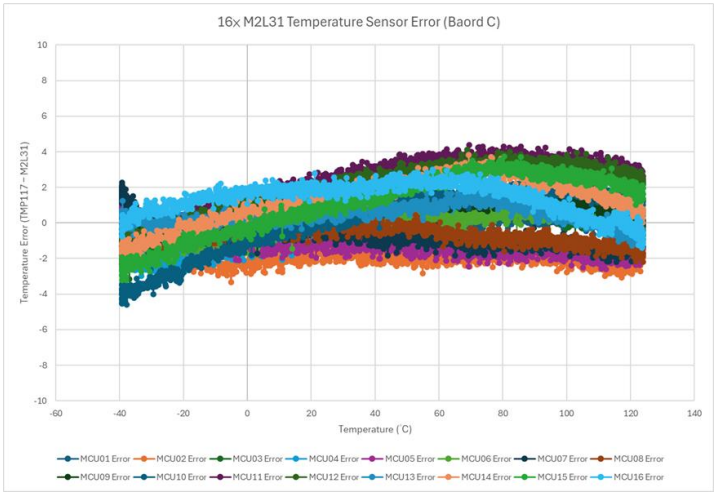


圖 1-5 16 顆 M2L31 溫度數值誤差(Board C)

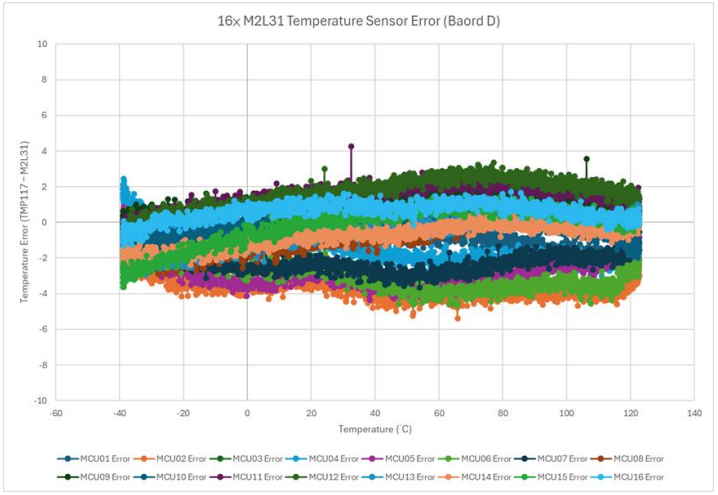


圖 1-6 16 顆 M2L31 溫度數值誤差(Board D)

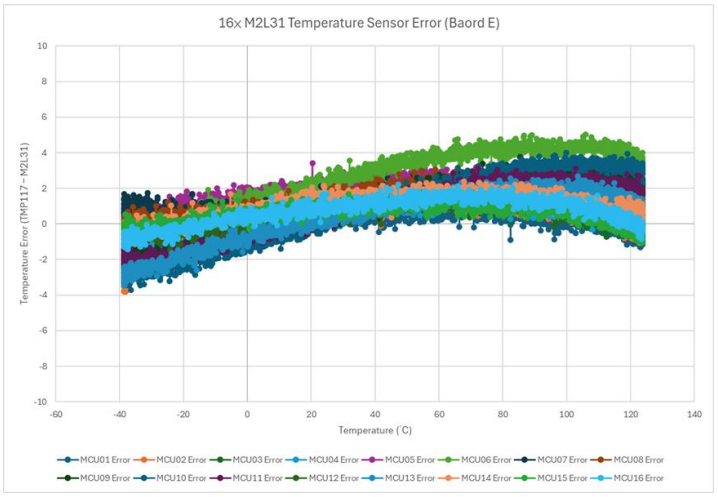


圖 1-7 16 顆 M2L31 溫度數值誤差(Board E)

2. 程式碼介紹

主程式首先初始化系統並讀取出廠校正值，隨後啟動 1 Hz 的 Timer0 定時中斷；在迴圈中，每秒偵測到 Timer_Flag 後便依序量測 AV_{DD} 電壓與晶片溫度，並將 16 位元資料拆分為高低位元組，最終透過自定義協定將封包即時傳送至 PC UI。

```
/*---------------------------------------------------------------------------------------------*/
/*  Main Function                                                                              */
/*---------------------------------------------------------------------------------------------*/
int32_t main(void)
{
    /* Init System, IP clock and multi-function I/O. */
    SYS_Init();

    /* Init UART0 for printf */
    UART0_Init();

    DEBUG_MSG("\n\n");

    DEBUG_MSG("+-------------------------------------------------+\n");

    DEBUG_MSG("|  Reads the M2L31's Internal Temperature Sensor  |\n");

    DEBUG_MSG("+-------------------------------------------------+\n");

    DEBUG_MSG("\nSystem clock rate: %d Hz\n\n", SystemCoreClock);

    /* Fetch factory calibration markers */
    Read_Temp_Trim_Data();

    /* Set timer frequency to 1HZ */
    TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, 1);

    /* Enable timer interrupt */
    TIMER_EnableInt(TIMER0);
    NVIC_EnableIRQ(TMR0_IRQn);

    /* Start Timer 0 */
    TIMER_Start(TIMER0);

    /* Main processing loop */
    while (1)
    {
        if (Timer_Flag)
        {
            /* Read Bandgap Voltage */
            Read_Bandgap_Voltage();

            /* EADC function test */
            Read_Temp_Sensor();

            /* Split 16-bit parameters into 8-bit streaming buffers */
```

```
            u8_Data[0] = AVDD_Value >> 8;
            u8_Data[1] = AVDD_Value;
            u8_Data[2] = Temp_Int_Value  >> 8;
            u8_Data[3] = Temp_Int_Value;

            /* Ship frame to UI if debug printing is suppressed */
            if (!DEBUG_MODE)
            {
                Output_UART0_UI_Data(u8_Data);
            }

            /* Reset the Flag */
            Timer_Flag = 0;
        }
    }
}
```

此函式首先解除暫存器保護以啟用內部帶隙電壓，接著透過 EADC 通道 28 進行單端取樣與轉換，並利用 RMC ISP 指令讀取出廠時的帶隙電壓參考標記值，藉此以公式反推出目前的動態 AV_{DD} 毫伏特數值，最後將結果送入中值濾波器以純化訊號並更新全域變數 AVDD_Value。

```
/* Measure Band-gap voltage to reverse-calculate the actual AVdd power rail */
void Read_Bandgap_Voltage(void)
{
    int32_t i32ConversionData;
    int32_t i32BuiltInData;

    /* Enable internal voltage band-gap reference */
    SYS_UnlockReg();
    SYS->VREFCTL |= SYS_VREFCTL_VBGFEN_Msk;
    SYS_LockReg();

    /* Configure ADC settings for channel 28 */
    EADC_Open(EADC, EADC_CTL_DIFFEN_SINGLE_END);
    EADC_SetExtendSampleTime(EADC, BANDGAP_CH, EXTEND_TIME);
    EADC_CLR_INT_FLAG(EADC, EADC_STATUS2_ADIF0_Msk);

    /* Enable ADC channel 28 interrupts and NVIC */
    EADC_ENABLE_INT(EADC, BIT0);
    EADC_ENABLE_SAMPLE_MODULE_INT(EADC, 0, BIT28);
    NVIC_EnableIRQ(EADC0_INT0_IRQn);

    /* Trigger ADC conversion and poll for completion */
    g_u32AdcIntFlag = 0;
    EADC_START_CONV(EADC, BIT28);

    while (g_u32AdcIntFlag == 0);

    /* Disable ADC interrupt and read conversion result */
    EADC_DISABLE_INT(EADC, BIT0);
    i32ConversionData = EADC_GET_CONV_DATA(EADC, BANDGAP_CH);

    /* Read factory calibrated band-gap value from FMC/RMC */
    SYS_UnlockReg();
```

```
    RMC_Open();
    i32BuiltInData = RMC_ReadBandGap();

    /* Reverse-calculate AVDD (mV) and update the ring buffer */
    AVDD_Data[AVDD_Count++] = 3072 * i32BuiltInData / i32ConversionData;

    if (AVDD_Count >= MF_NUM)
        AVDD_Count = 0;

    /* Apply Median Filter to determine stable AVDD value */
    AVDD_Value = GetMedianNum((uint32_t *)AVDD_Data, MF_NUM);

    DEBUG_MSG("AVdd = %d mV, ", AVDD_Value);
}
```

此函式透過 EADC 通道 29 擷取內部溫度感測器的原始 ADC 值，經由中值濾波器消除突波後，將其與即時的 AV_{DD} 電壓結合以換算出目前感測器電壓 V_Curr，隨後帶入出廠的 25°C 與 125°C 雙點線性插值公式計算出真實攝氏溫度，最後將浮點數結果乘以 100 轉為定點數整數，以供 PC UI 傳輸與顯示。

```
/* Sample temperature sensor and process using two-point linear interpolation */
void Read_Temp_Sensor(void)
{
    int32_t i32ConversionData;

    /* Configure ADC hardware settings */
    EADC_Open(EADC, EADC_CTL_DIFFEN_SINGLE_END);
    EADC_SetExtendSampleTime(EADC, TEMP_CH, EXTEND_TIME);
    EADC_CLR_INT_FLAG(EADC, EADC_STATUS2_ADIF0_Msk);

    /* Enable ADC interrupts and NVIC */
    EADC_ENABLE_INT(EADC, BIT0);
    EADC_ENABLE_SAMPLE_MODULE_INT(EADC, 0, BIT29);
    NVIC_EnableIRQ(EADC0_INT0_IRQn);

    /* Trigger ADC conversion and poll for completion */
    g_u32AdcIntFlag = 0;
    EADC_START_CONV(EADC, BIT29);

    while (g_u32AdcIntFlag == 0);

    /* Read conversion result */
    i32ConversionData = EADC_GET_CONV_DATA(EADC, TEMP_CH);

    /* Update ring buffer and apply Median Filter */
    TEMP_Data[TEMP_Count++] = i32ConversionData;

    if (TEMP_Count >= MF_NUM)
        TEMP_Count = 0;

    TEMP_Value = GetMedianNum((uint32_t *)TEMP_Data, MF_NUM);

    DEBUG_MSG("Temp_ADC = %d,", TEMP_Value);
```

```
    /* Calculate current voltage and interpolate temperature */
    V_Curr = (float)TEMP_Value / 4095.0 * (AVDD_Value / 1000.0);
    Temp_Curr = (V_Curr - V_25dC) / (V_125dC - V_25dC) * (TEMP_HIGH - TEMP_LOW) + 25.0;

    DEBUG_MSG(" Temp_Celsius = %3.2f\n", Temp_Curr);

    /* Convert float to fixed-point integer (2 decimal places) for UI packing */
    Temp_Int_Value = (int32_t)(Temp_Curr * 100);
}
```

此函式將傳入的電壓與溫度高低位元組，嚴密封裝進以 0x49 為標頭、0x19 為標尾且資料長度固定為 5 的 7 位元組陣列中，隨後透過迴圈輪詢 UART0 的傳送 FIFO 狀態，在確保硬體緩衝區未滿的前提下，依序將每個位元組推向 TX 腳位輸出。

```
/* A custom UART protocol frame and transmit */
void Output_UART0_UI_Data(uint8_t *Data)
{
    volatile uint8_t i;
    volatile uint8_t Out_Str[TX_LENGTH];

    /* Format 7-byte protocol packet */
    Out_Str[0] = TX_HEADER;      // Header
    Out_Str[1] = TX_LENGTH - 2;  // Length
    Out_Str[2] = Data[0];        // AVDD High-Byte
    Out_Str[3] = Data[1];        // AVDD Low-Byte
    Out_Str[4] = Data[2];        // Temp High-Byte
    Out_Str[5] = Data[3];        // Temp Low-Byte
    Out_Str[6] = TX_END;         // End

    /* Transmit packet via UART0 */
    for (i = 0; i < TX_LENGTH; i++)
    {
        while (UART_IS_TX_FULL(UART0)); // Wait if TX FIFO is full

        UART_WRITE(UART0, Out_Str[i]);  // Send byte
    }
}
```

3. 軟體與硬體需求

3.1 軟體需求

- BSP 版本
  - M2L31_Series_BSP_CMSIS_V3.03.000
- IDE 版本
  - Keil uVision 5.43
  - Visual Studio Code 1.125.1

3.2 硬體需求

- 電路元件
  - NuMaker-M2L31SG_V1.1
- 線路示意圖
  - 將 NuMaker-M2L31SG_V1.1 的 USB 連接到 PC 的 USB，以顯示範例程式碼的執行結果，如圖 3-1 所示。

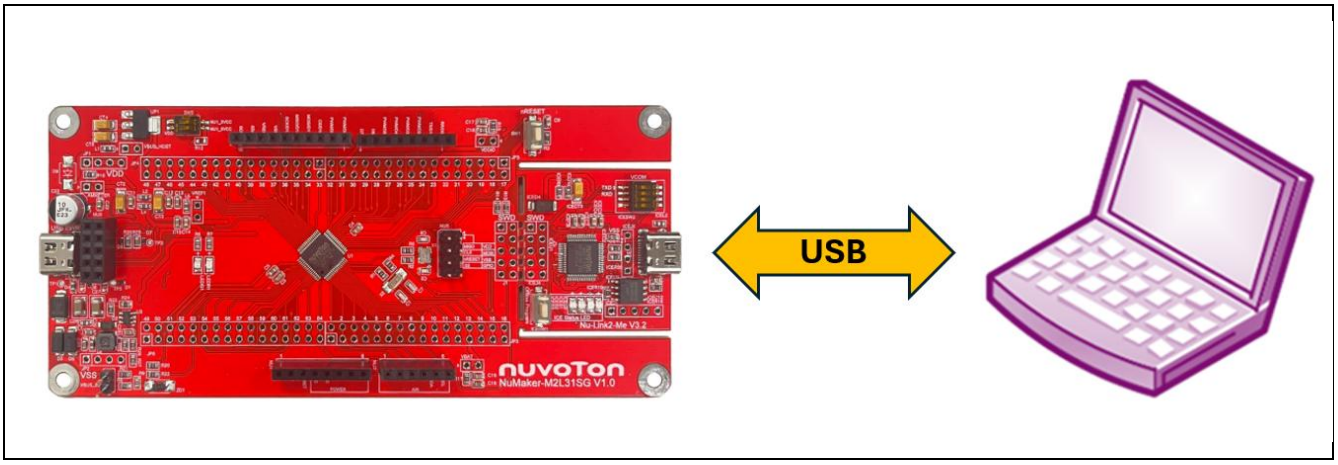


圖 3-1 線路示意圖

4. 目錄資訊

目錄結構如下圖所示。

EC_M2L31_Internal_Temperature_Sensor_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	
Internal_Temperature_Sensor	Source file of example code
WindowsTool	User Control GUI Tool

圖 4-1 目錄資訊

5. 範例程式執行

5.1 Keil MDK

1. 在 Windows 檔案總管中開啟 **Keil** 專案資料夾，並開啟 *Internal_Temperature_Sensor.uvprojx* 專案檔。
2. 執行建置（Build）。
3. 下載程式。
4. 進入除錯模式。
5. 開啟 **UI**（M2L31_Temp_Sensor.exe）。
   - 選擇 **COM Port**。
   - 點擊 **Open**。
   - 顯示 **AV_{DD}** 電壓值、溫度值及溫度曲線。

5.2 VS Code

1. 在 VS Code IDE 中選擇 **File**。
   - 點擊 **Open Folder**。
   - 選擇專案目錄下的 **VSCode** 資料夾。
2. 選擇 **CMSIS**。
   - 點擊 **Build solution**，編譯專案。
   - 點擊 **Load & Run application**，將程式燒錄至目標晶片並執行。
   - 點擊 **Load & Debug application**，進入除錯模式。
3. 進入除錯介面。
4. 開啟 **UI**（M2L31_Temp_Sensor.exe）。
   - 選擇 **COM Port**。
   - 點擊 **Open**。

- 顯示 **AV_{DD}** 電壓值、溫度值及溫度曲線。

6. 修訂紀錄

Date	Revision	Description
2026.06.11	1.00	初始發布。

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.