M2L31 Internal Temperature Sensor

NuMicro® 32 位系列微控制器范例代码介绍

文件信息

应用简述	本示例程序代码读取 M2L31 的内部温度传感器数值，并显示在 GUI。
BSP 版本	M2L31_Series_BSP_CMSIS_V3.03.000
开发平台	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

本示例程序代码展示如何读取 NuMicro® M2L31 系列微控制器的内部温度传感器。系统启动时，会先从 RMC Flash 读取出厂刻录的 25°C 与 125°C 两点校正数据，随后由 Timer0 每秒触发一次测量。测量时分别通过 EADC 通道 28 测量带隙电压 (Bandgap Voltage)以计算实际 AVDD 电压，并通过 EADC 通道 29 测量温度传感器电压。两组数值经中值滤波滤除噪声后，会套用两点线性插值公式换算为摄氏温度，最后打包成自定义 UART 数据包，以 115200 bps 传送至 PC 端 UI 软件，供用户实时观看温度曲线。

1.1 原理

本示例首先通过 RMC ISP 指令，读取 M2L31 芯片出厂的两点温度校正值，借此决定温度基底斜率。随后在运行时，依序通过 EADC 通道 28 与 29 采集内部带隙电压与温度传感器信号，并套用 10 笔数据的中值滤波器（Median Filter）以剔除突波噪声；接着利用该数据反推出精准的动态 AVDD 电源电压，并通过双点线性插值公式，计算出精确至小数点后两位的温度定点数。最终，系统将电压与温度数据进行封装，形成一个以 0x49 为帧头、0x19 为帧尾的 7 字节自定义通信数据包，其格式如图 1-1 所示，通过 UART0 以 115200 bps 的速率实时推送到 PC 端 UI，以绘制出稳定的监测曲线。

带隙电压计算：

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

摄氏温度计算：

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

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

图 1-1 UART 数据输出

1.2 执行结果

本示例程序代码支持通过计算机 UI 进行实时数据监控(如图 1-2)。操作时，首先于「Serial Port」区块选择正确的 COM 端口并建立通讯；连接成功后，UI 接口将实时解析并显示由 EADC 测量出的 AVDD 电压与精确温度，并于下方动态绘制出平滑的温度监测折线图。最后，用户可通过点击「Log」区块的 Start 按钮，将含有精确时间戳、电压与温度的历史数据，自动且连续地写入并导出为.csv 文件，以利后续进行长期的稳定性测试与数据分析。

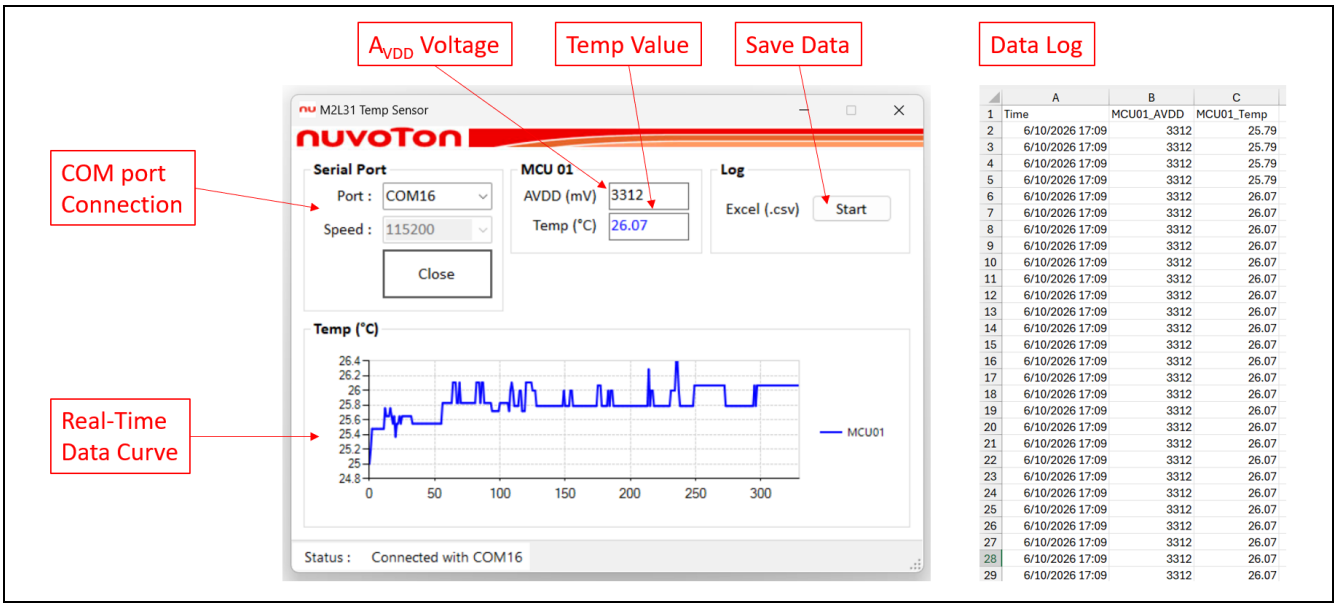


图 1-2 UI 连接说明

1.3 同时测量 80 颗 M2L31 内部温度传感器的误差大小

本实验以高精度的 TMP117 温度传感器数值作为基准标准（Golden Sample），在 -40°C 至 125°C 的宽温范围内，同时对大量（共 80 颗）M2L31 芯片进行内部温度传感器的误差测量。实验结果显示（如图 1-3、图 1-4、图 1-5、图 1-6、图 1-7 所示），各芯片在全温区的测量误差曲线皆呈现稳定的弧形飘移趋势，且整体的温度误差全数成功控制在 ±5°C 的范围以内。这证实了通过出厂两点校正与动态 AVDD 电源补偿后，该芯片内置的温度传感器在极端温度环境下仍具备极高的准确度与优异的可靠性。

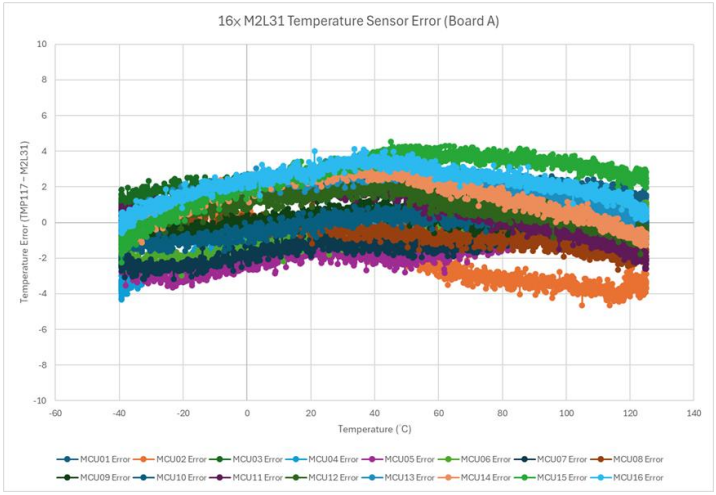


图 1-3 16 颗 M2L31 温度数值误差(Board A)

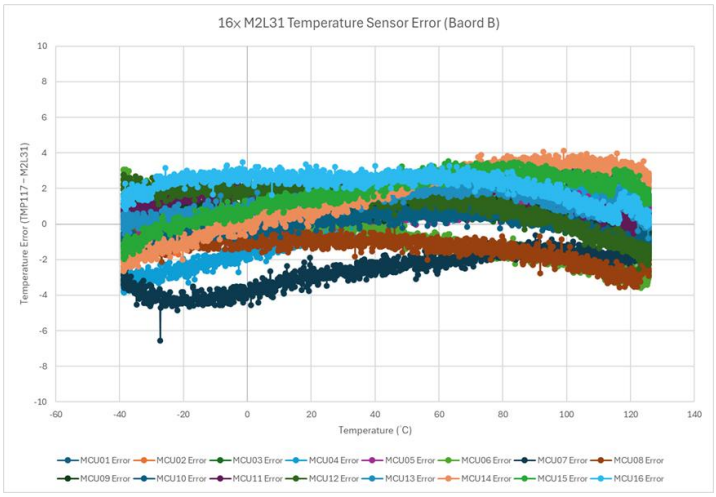


图 1-4 16 颗 M2L31 温度数值误差(Board B)

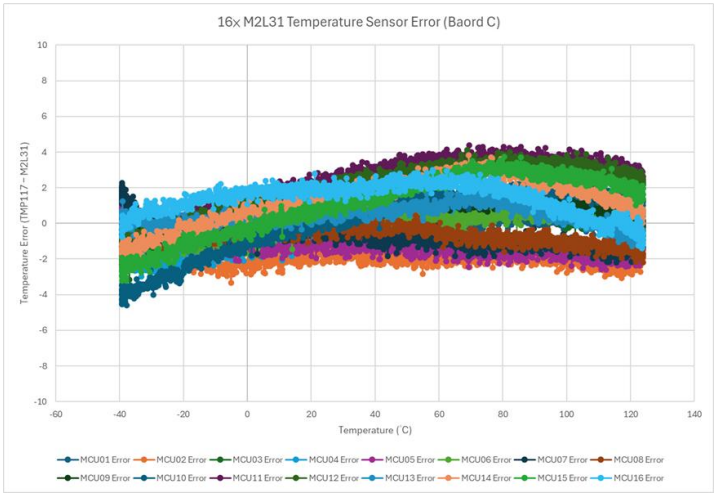


图 1-5 16 颗 M2L31 温度数值误差(Board C)

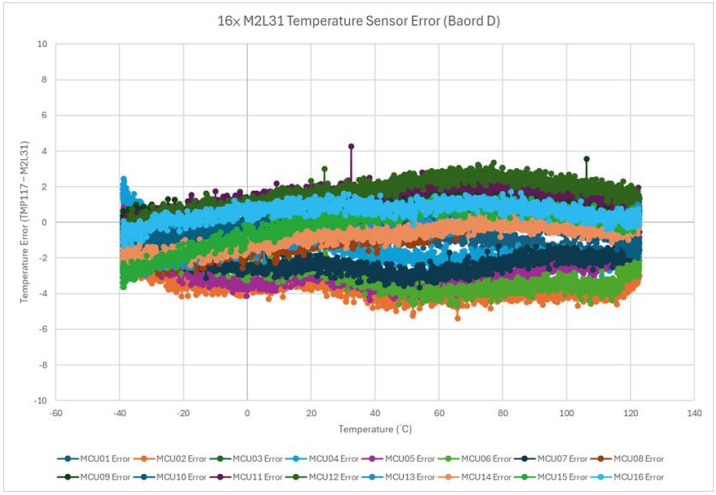


图 1-6 16 颗 M2L31 温度数值误差(Board D)

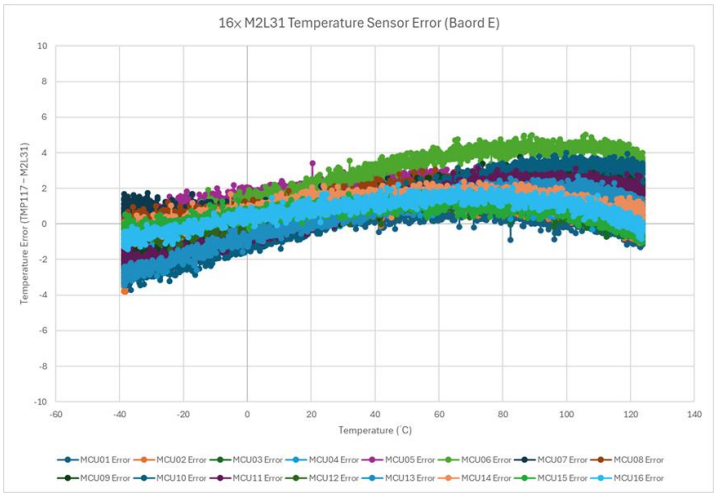


图 1-7 16 颗 M2L31 温度数值误差(Board E)

2. 程序代码介绍

主程序首先初始化系统并读取出厂校正值，随后启动 1 Hz 的 Timer0 定时中断；在循环中，每秒检测到 Timer_Flag 后便依序测量 AVDD 电压与芯片温度，并将 16 位数据拆分为高低字节，最终通过自定义协议将数据包实时传送至 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;
        }
    }
}
```

此函数首先解除缓存器保护以启用内部带隙电压，接着通过 EADC 通道 28 进行单端取样与转换，并利用 RMC ISP 指令读取出厂时的带隙电压参考标记值，借此以公式反推出当前的动态 AVDD 毫伏特数值，最后将结果送入中值滤波器以纯化信号并更新全局变量 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);
}
```

此函数通过 EADC 通道 29 采集内部温度传感器的原始 ADC 值，经由中值滤波器消除突波后，将其与实时的 AVDD 电压结合以换算出当前传感器电压 V_Curr，随后带入出厂的 25°C 与 125°C 双点线性插值公式计算出真实摄氏温度，最后将浮点数结果乘以 100 转为定点数整数，以供 PC UI 传输与显示。

```
/* 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);
}
```

此函数将传入的电压与温度高低字节，严密封装进以 0x49 为帧头、0x19 为帧尾且数据长度固定为 5 的 7 字节数组中，随后通过循环轮询 UART0 的传送 FIFO 状态，在确保硬件缓冲区未满的前提下，依序将每个字节推向 TX 引脚输出。

```
/* 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. 软件与硬件需求

3.1 软件需求

- 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 硬件需求

- 电路组件
  - NuMaker-M2L31SG_V1.1
- 线路示意图
  - 将 NuMaker-M2L31SG_V1.1 的 USB 连接到 PC 的 USB，以显示示例程序代码的执行结果，如图 3-1 所示。

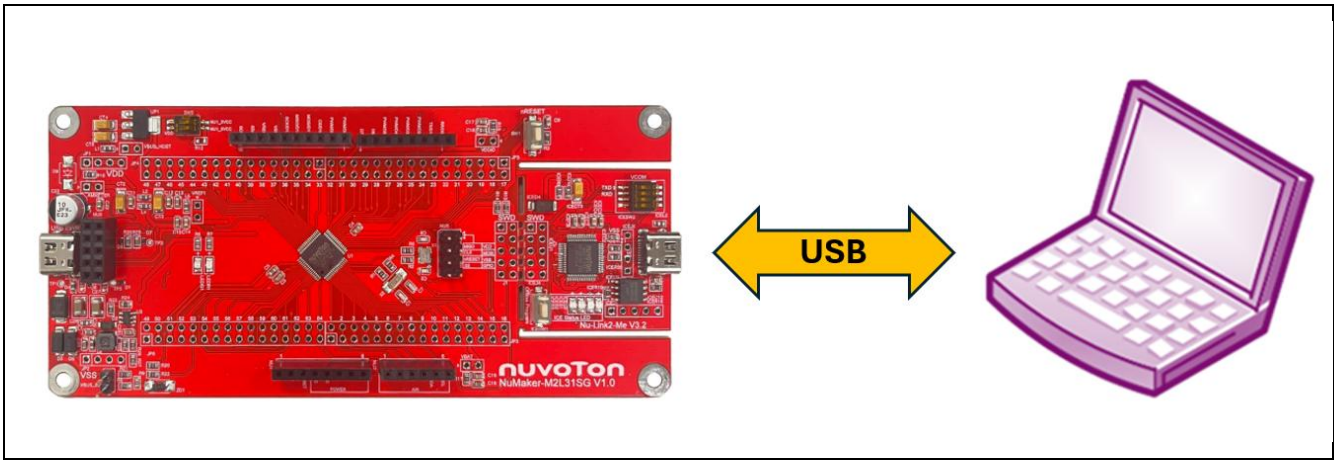


图 3-1 线路示意图

4. 目录信息

目录结构如下图所示。

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

图 4-1 目录信息

5. 示例程序执行

5.1 Keil MDK

1. 在 Windows 档案总管中开启 **Keil** 项目文件夹，并开启 *Internal_Temperature_Sensor.uvprojx* 专案档。
2. 执行建置（Build）。
3. 下载程序。
4. 进入除错模式。
5. 开启 **UI**（M2L31_Temp_Sensor.exe）。
   - 选择 **COM Port**。
   - 点击 **Open**。
   - 显示 **AV_{DD}** 电压值、温度值及温度曲线。

5.2 VS Code

1. 在 **VS Code IDE** 中选择 File。
   - 点击 **Open Folder**。
   - 选择项目目录下的 **VSCode** 文件夹。
2. 选择 **CMSIS**。
   - 点击 **Build solution**，编译项目。
   - 点击 **Load & Run application**，将程序刻录至目标芯片并执行。
   - 点击 **Load & Debug application**，进入除错模式。
3. 进入除错界面。
4. 开启 **UI**（M2L31_Temp_Sensor.exe）。
   - 选择 **COM Port**。
   - 点击 **Open**。

- 显示 **AV_{DD}** 电压值、温度值及温度曲线。

6. 修订纪录

Date	Revision	Description
2026.06.11	1.00	初始发布。

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.