# M2L31 Internal Temperature Sensor

Example Code Introduction for 32-bit NuMicro® Family

## Document Information

| | |
|---|---|
| Application | This example code reads the value from the M2L31 internal temperature sensor and displays it on the GUI. |
| BSP Version | M2L31_Series_BSP_CMSIS_V3.03.000 |
| Hardware | 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. Overview

This example code demonstrates how to read the internal temperature sensor of the NuMicro M2L31 series microcontroller. Upon system startup, the factory-calibrated two-point data for 25°C and 125°C is retrieved from the RMC memory. Timer0 then triggers a measurement once every second. During measurement, EADC Channel 28 measures the band-gap voltage to calculate the actual $AV_{DD}$ voltage, while EADC Channel 29 measures the temperature sensor voltage. After filtering out noise using a median filter, both values are converted into Celsius using a two-point linear interpolation formula. Finally, the data is packed into a custom UART packet and transmitted to the PC-side UI at 115200 bps, allowing users to view the real-time temperature curve.

## 1.1 Principle

This example first uses RMC ISP commands to read the two-point temperature calibration values pre-programmed in the M2L31 series, thereby determining the base temperature slope. During runtime, it sequentially captures the internal band-gap voltage and temperature sensor signals via EADC channels 28 and 29. A 10-sample median filter is applied to eliminate spike noise. This data is then used to derive the precise dynamic $AV_{DD}$ supply voltage. Using a two-point linear interpolation formula, the system calculates a fixed-point temperature value accurate to two decimal places. Finally, the voltage and temperature data are strictly encapsulated into a 7-byte custom communication packet (as shown in Figure 1-1) with a 0x49 header and a 0x19 trailer. This packet is transmitted via UART0 at a baud rate of 115200 bps to the PC-side UI to plot a stable monitoring curve.

Band-gap Voltage Calculation:

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

Celsius Temperature Calculation:

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

| 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 |

Figure 1-1 UART Data Output

## 1.2 Demo Result

This example code supports real-time data monitoring through the PC-side UI (as shown in Figure 1-2). To operate, first select the correct COM port in the 'Serial Port' section to establish communication. Once connected successfully, the UI will parse and display the $AV_{DD}$ voltage

and precise temperature measured by the EADC in real time, while dynamically plotting a smooth temperature monitoring line graph below. Finally, users can click the '**Start**' button in the 'Log' section to automatically and continuously record the historical data—with precise timestamps, voltages, and temperatures—and export it to a *.csv* file for subsequent long-term stability testing and data analysis.

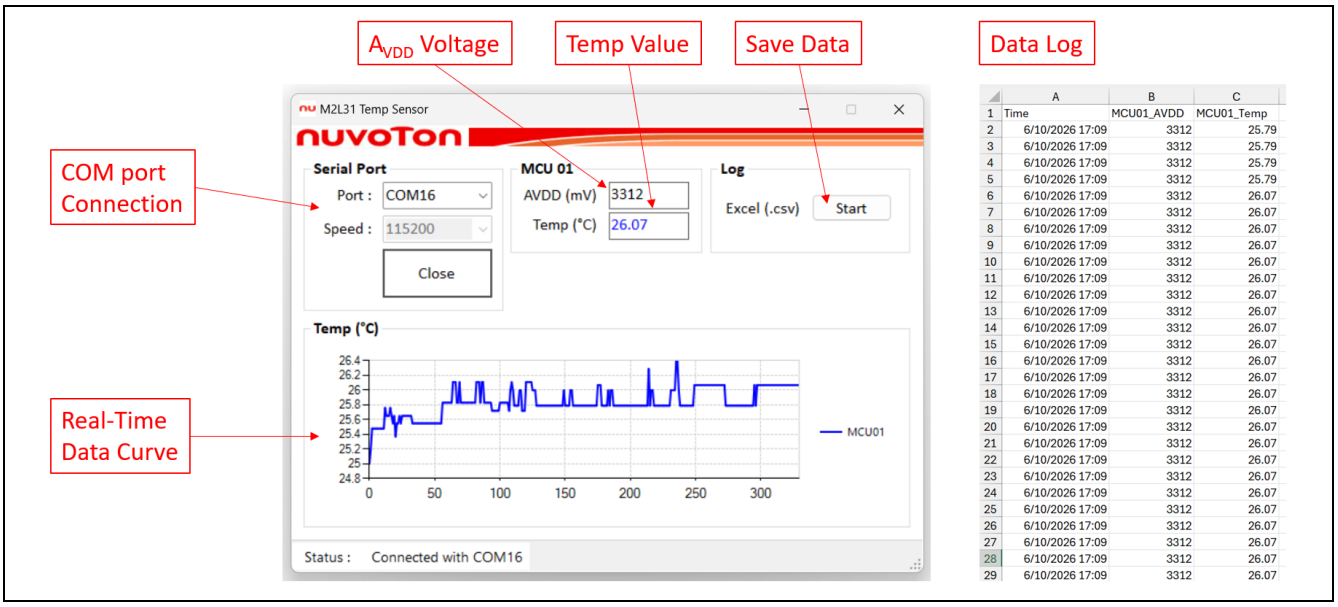



Figure 1-2 UI Connection Instructions

## 1.3 Simultaneous Error Measurement of Internal Temperature Sensors for 80 M2L31 Chips

In this experiment, the readings from a high-precision TMP117 temperature sensor are utilized as the baseline reference (golden sample) to simultaneously measure the internal temperature sensor errors of a large batch (80 chips in total) of M2L31 series microcontrollers across a wide temperature range from -40°C to 125°C. The experimental results (as illustrated in Figure 1-3, Figure 1-4, Figure 1-5, Figure 1-6, and Figure 1-7) indicate that the measurement error curves of all chips exhibit a stable, curved drift profile across the entire temperature spectrum, with the overall temperature error successfully contained within a ±5°C window. This demonstrates that by leveraging factory two-point calibration alongside dynamic $AV_{DD}$ power supply compensation, the chip's built-in temperature sensor maintains exceptionally high accuracy and outstanding reliability even under extreme temperature environments.

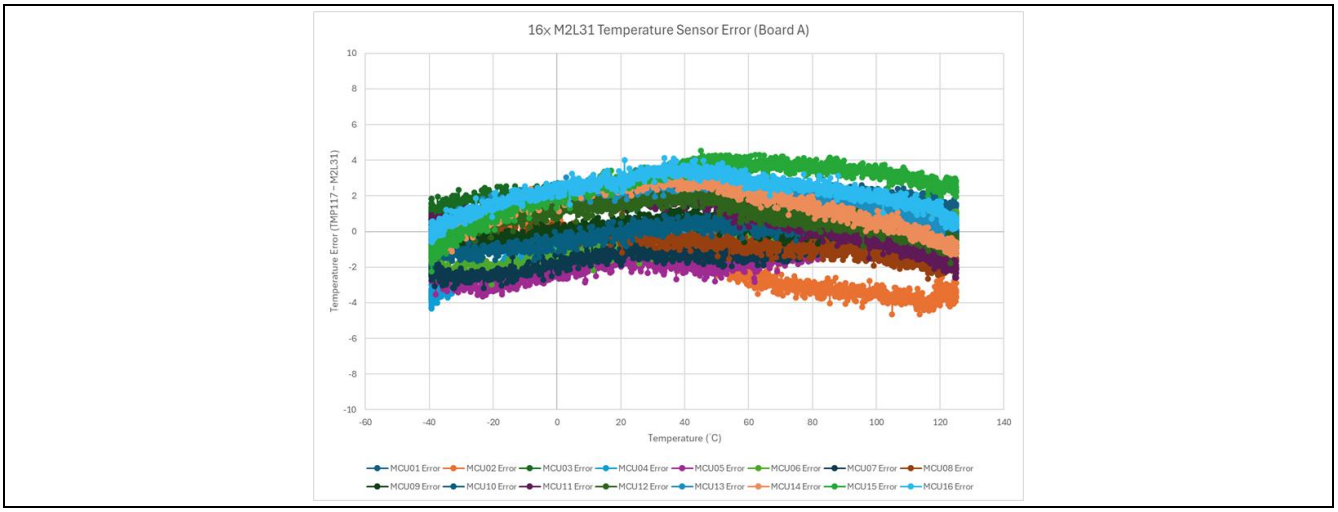



Figure 1-3 Temperature Measurement Errors for 16 M2L31 Chips (Board A)

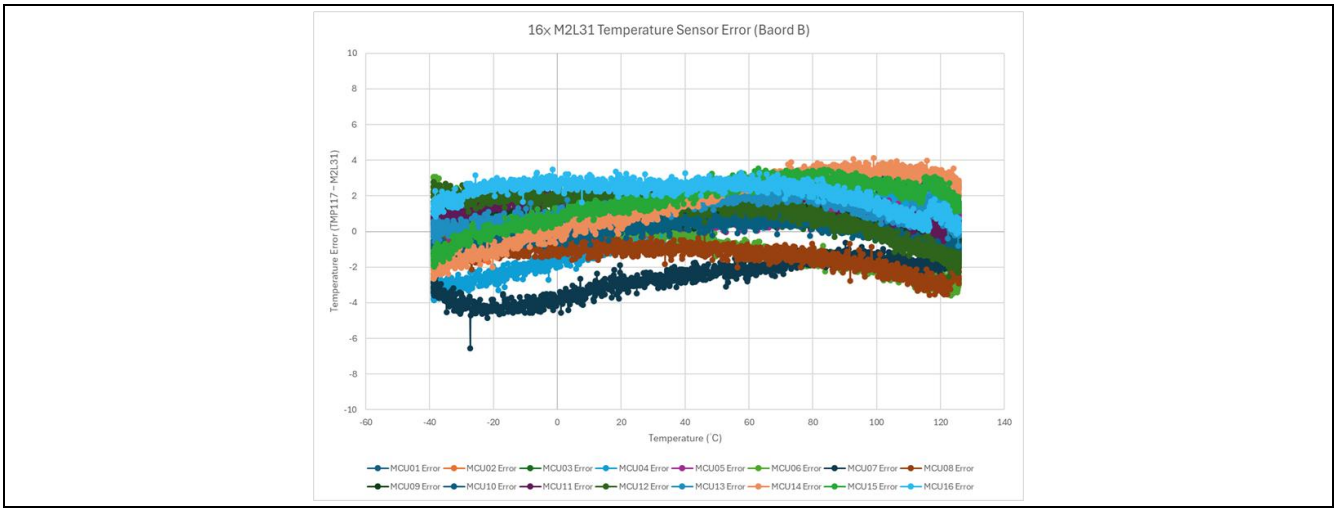



Figure 1-4 Temperature Measurement Errors for 16 M2L31 Chips (Board B)

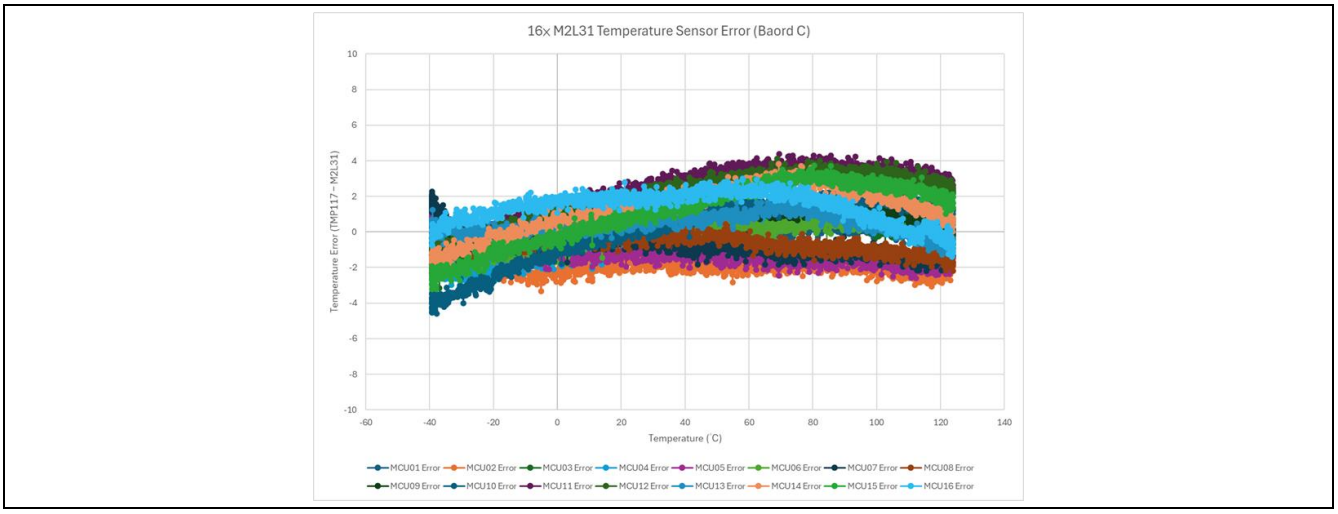



Figure 1-5 Temperature Measurement Errors for 16 M2L31 Chips (Board C)

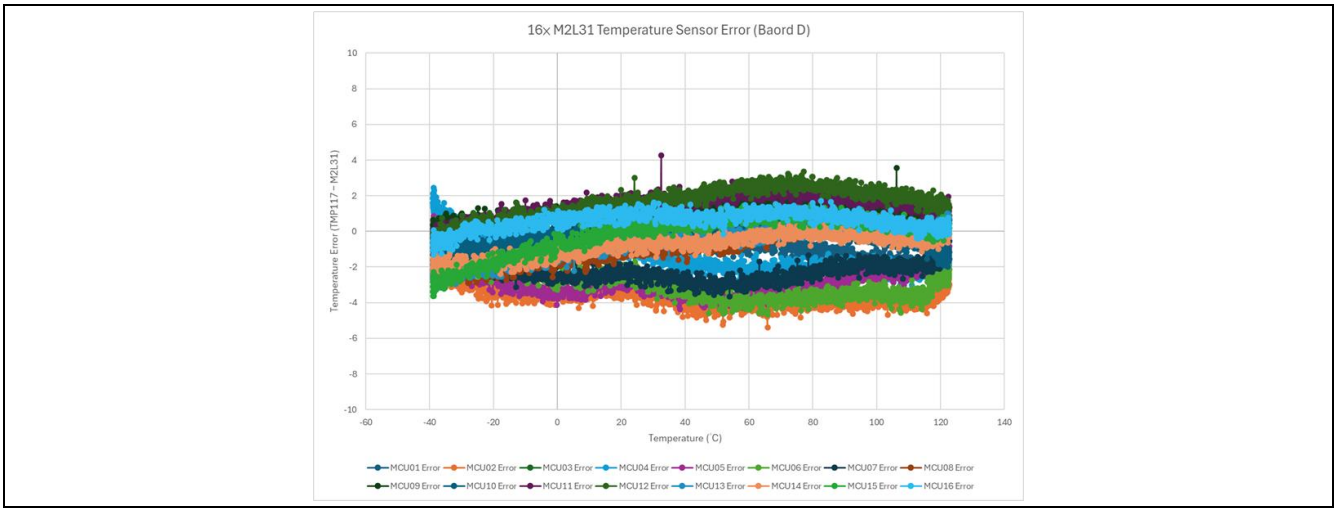



Figure 1-6 Temperature Measurement Errors for 16 M2L31 Chips (Board D)

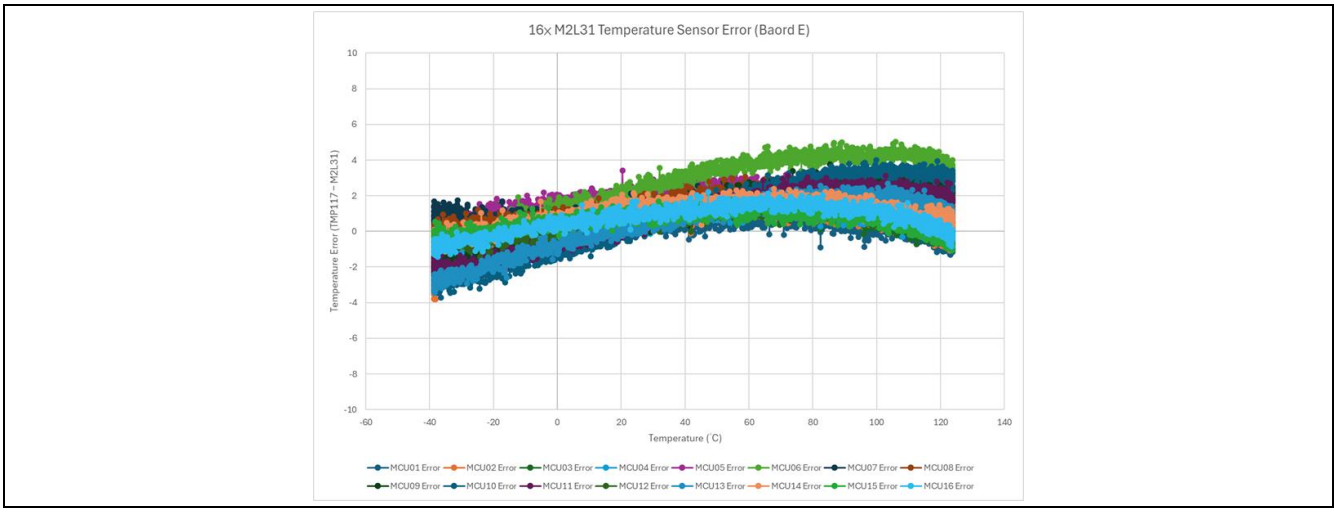



Figure 1-7 Temperature Measurement Errors for 16 M2L31 Chips (Board E)

## 2. Code Description

The main program first initializes the system and retrieves the factory calibration values, subsequently starting a 1 Hz Timer0 periodic interrupt. In the main loop, upon detecting the Timer_Flag every second, the system sequentially measures the $AV_{DD}$ voltage and chip temperature. It then splits the 16-bit data into high and low bytes and ultimately transmits the packet in real time to the PC-side UI via the custom protocol.

```
/*---------------------------------------------------------------------------------------------*/
/*  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;
        }
    }
}
```

This function first disables register write-protection to enable the internal band-gap voltage reference source. It then performs single-ended sampling and conversion via EADC channel 28 and uses RMC ISP commands to read the factory-programmed band-gap calibration reference value. Using this data, the current dynamic $AV_{DD}$ value in millivolts (mV) is back-calculated via the formula. Finally, the result is passed into a median filter to remove signal noise and update the global variable 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);
}
```

This function captures the raw ADC value of the internal temperature sensor via EADC channel 29. After eliminating spike noise through a median filter, it combines the filtered reading with the real-time $AV_{DD}$ voltage to convert it into the current sensor voltage, V_Curr. Subsequently, this voltage is applied to the factory 25°C and 125°C two-point linear interpolation formula to calculate the true Celsius temperature. Finally, the floating-point result is multiplied by 100 to convert it into a fixed-point integer for transmission and display on the PC-side UI.

```
/* 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);
}
```

This function packs the high and low bytes of voltage and temperature values into a 7-byte array with a 0x49 header, a 0x19 trailer, and a fixed data length of 5. It then sequentially polls the UART0 TX FIFO status in a loop, ensuring that each byte is transmitted only when the hardware buffer is not full.

```
/* 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. Software and Hardware Requirements

## 3.1 Software Requirements

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

## 3.2 Hardware Requirements

- Circuit components
  - NuMaker-M2L31SG_V1.1
- Pin connect
  - Connect the USB port of the NuMaker-M2L31SG to the PC's USB port to display the execution results of the example code as shown in Figure 3-1.

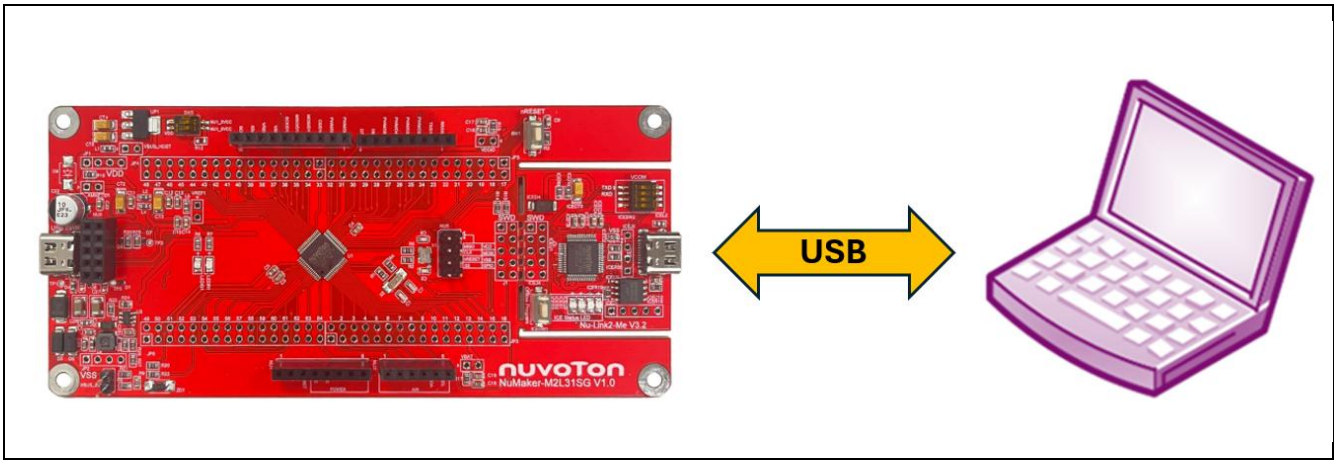



Figure 3-1 Pin Connect

# 4. Directory Information

The directory structure is shown below.

| | |
|---|---|
| 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 |

Figure 4-1 Directory Structure

# 5. Example Code Execution

## 5.1 Keil MDK

1. Navigate to the **Keil** project folder in Windows Explorer and open the *Internal_Temperature_Sensor.uvprojx* project file.
2. Build.
3. Download.
4. Enter debug mode.
5. Launch the **UI** (*M2L31_Temp_Sensor.exe*)
   - Select the **COM Port**.
   - Click "**Open**".
   - Display the **$AV_{DD}$** voltage value, temperature value, and temperature curve.

## 5.2 VS Code

1. Select **File** in VS Code IDE
   - Click **Open Folder**
   - Select the **VSCode** folder under the project directory
2. Select **CMSIS**
   - Click **Build solution** to compile the project
   - Click **Load & Run application** to program the target chip
   - Click **Load & Debug application** to enter debug mode
3. Enter the debug interface:
4. Launch the **UI** (*M2L31_Temp_Sensor.exe*)
   - Select the **COM Port.**
   - Click "**Open**".
   - Display the $AV_{DD}$ voltage value, temperature value, and temperature curve.

# 6. Revision History

| Date | Revision | Description |
|---|---|---|
| 2026.06.11 | 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.*