Receiving Variable Length Data by LPUART in Power-down Mode

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code implements receiving data of variable length through LPUART in Power-down mode, and then wakes up the MCU for data processing. This realizes the operating characteristics of M2L31 low power LPUART.
BSP Version	M2L31_Series_BSP_CMSIS_V3.00.000
Hardware	NuMaker-M2L31KI V2.1

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 will demonstrate how to use M2L31's LPUART and LPPDMA to receive data of variable length in Power-down mode, and then wake up the MCU. The MCU will process the data and then enter Power-down mode again, and so on. This reduces system power consumption.

1.1 Principle

This example code mainly uses three peripherals: LPUART0, LPPDMA0 and UART0.

- LPUART0 is set to auto-operation mode to receive data in Power-down mode. At the same time, LPUART RX is set to trigger LPPDMA. When LPUART receives data, LPPDMA is triggered to transfer the data to LPSRAM. The bus idle time-out wake-up is also enabled. When data starts to be received, the time-out counter starts counting. Receiving new data will reset the counter. The MCU will be woken up after the counter times out. This function can be used to wake up the system after receiving data of variable length. LPUART Auto-operation mode RX state Block Diagram is shown in Figure 1-1.

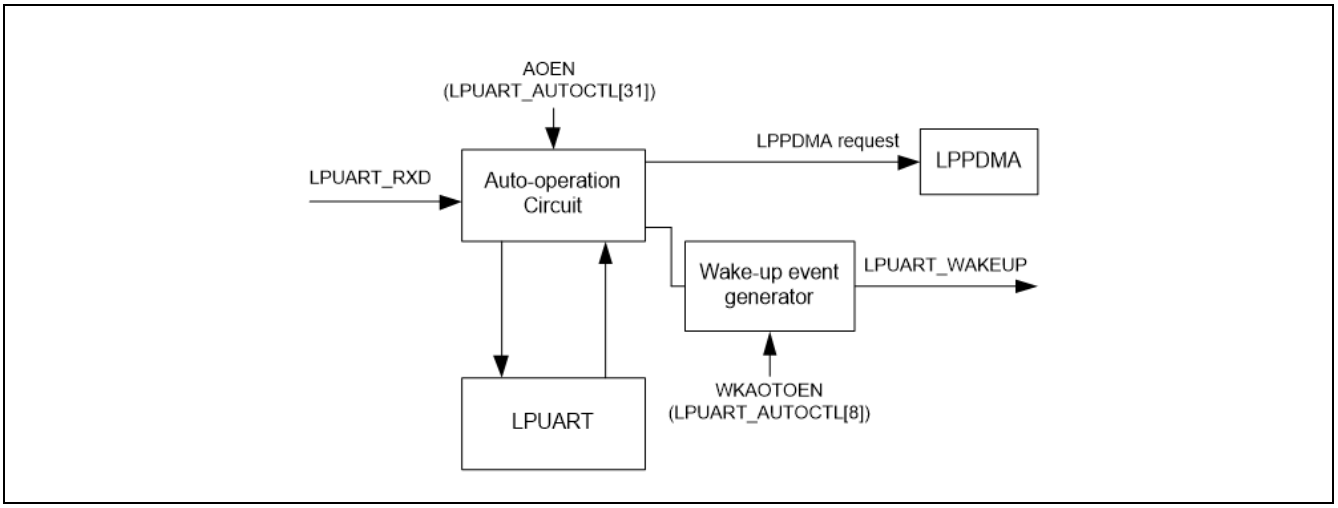

Figure 1-1 LPUART Auto-operation Mode RX State Block Diagram

- LPPDMA can transfer data in Power-down mode. Configure the basic operation mode to transfer the data received by LPUART RX to LPSRAM. LPPDMA will stop after completing the data transfer of the set length.
- As a debugging serial port, UART0 is mainly used to print debugging information and input interactive information.

1.2 Demo Result

The programming results of this example code are shown in Figure 1-2. Open two UART tools on PC. The UART tool in the left of Figure 1-2 connects to UART0 for printing debugging information. The UART tool in the right of Figure 1-2 connects to LPUART to send data to the MCU. The operation steps are as follows:

1. PC sends any data to UART0 of MCU through UART tool to start test.
2. MCU enters Power-down mode.
3. PC sends 26 bytes to LPUART of MCU through UART tool.
4. MCU wakes up after receiving 26 bytes and prints the received data.
5. MCU enters Power-down mode again.
6. PC sends 1066 bytes to LPUART of MCU through UART tool.
7. When the number of data received by MCU exceeds the buffer size, MCU prints a warning message.
8. MCU wakes up and prints the received data.

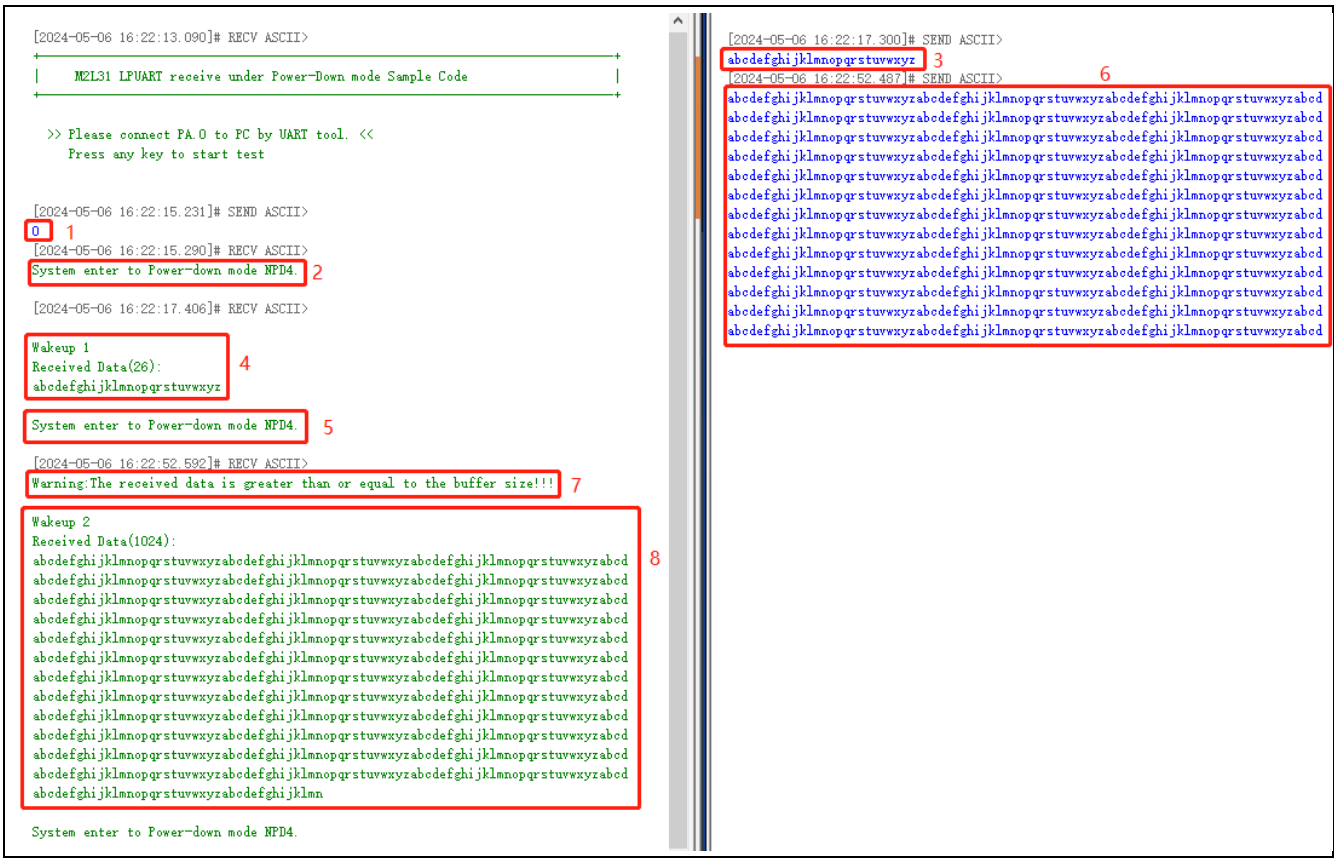

Figure 1-2 Programming Results

The average current is 9 µA when no data is received in NPD4 mode, and the average current is 244 µA when LPUART receives data in NPD4 mode.

2. Code Description

Define the LPPDMA channel, the size of the data buffer, and the level of Power-down mode. The buffer is set to 1024 bytes to avoid data loss caused by receiving too much data at one time.

```
#define LPUART_RX_LPPDMA_CH  0      /* LPPDMA channel for LPUART RX */
#define RXBUFSIZE            1024   /* Buffer size */
#define POWER_DOWN_MODE      CLK_PMUCTL_PDMSEL_NPD4   /* Power-down mode */
```

Specify the address of the data receiving buffer in LPSRAM.

```
#if (defined(__GNUC__) && !defined(__ARMCC_VERSION))
uint8_t g_u8RecData[RXBUFSIZE] __attribute__((section(".lpSram")));
#else
uint8_t g_u8RecData[RXBUFSIZE] __attribute__((section(".ARM.__at_0x28000000")));
#endif
```

LPPDMA initialization

Configure LPPDMA channel 0 to transmit the data received by LPUART RX. In the basic operating mode, LPPDMA will stop after completing the data transfer of the set length.

```
void LPPDMA_RX_Init(void)
{
/*---------------------------------------------------------------------------------------------
        LPPDMA transfer configuration:
        Channel = 0
        Operation mode = basic mode
        Request source = LPUART0 RX (peripheral to memory)
        transfer done interrupt = enable
        Transfer count = RXBUFSIZE
        Transfer width = 8 bits (one byte)
        Source address = LPUART0->DAT
        Source address increment size = 0 (fixed)
        Destination address = g_u8RecData
        Destination address increment size = 8 bits (one byte)
        Transfer type = Single transfer
        Total transfer length = RXBUFSIZE * 8 bits
---------------------------------------------------------------------------------------------*/
    /* Open Channel 0 */
    LPPDMA_Open(LPPDMA0,1 << LPUART_RX_LPPDMA_CH);

    /* Transfer count is RXBUFSIZE, transfer width is 8 bits (one byte) */
    LPPDMA_SetTransferCnt(LPPDMA0,LPUART_RX_LPPDMA_CH, LPPDMA_WIDTH_8, RXBUFSIZE);

    /* Set address */
    LPPDMA_SetTransferAddr(LPPDMA0,
             LPUART_RX_LPPDMA_CH,
             (uint32_t)&(LPUART0->DAT),/* Source address */
             LPPDMA_SAR_FIX,           /* Source address fixed */
             (uint32_t)g_u8RecData,    /* Destination address */
             LPPDMA_DAR_INC);          /* Destination address incremented, size = 1byte */

    /* Request source is LPUART0 RX to memory */
```

```
    LPPDMA_SetTransferMode(LPPDMA0,LPUART_RX_LPPDMA_CH, LPPDMA_LPUART0_RX, FALSE, 0);

    /* Transfer type is Single transfer */
    LPPDMA_SetBurstType(LPPDMA0,LPUART_RX_LPPDMA_CH, LPPDMA_REQ_SINGLE, 1);

    /* Enable transfer done interrupt */
    LPPDMA_EnableInt(LPPDMA0, LPUART_RX_LPPDMA_CH, LPPDMA_INT_TRANS_DONE);

    /* Enable LPPDMA0 IRQ */
    NVIC_EnableIRQ(LPPDMA0_IRQn);
}
```

LPUART initialization

Set the baud rate of LPUART to 9600 bps, enable auto-operation mode, RX trigger LPPDMA, bus idle time-out wake-up and RX time-out interrupt. When the first data starts to be received, the time-out counter starts counting. When new data starts to be received, the counter will be reset. Once the counter times out, the MCU will wake up and generate an interrupt. The time-out time is set in the register TOUT [7:0] (TOIC). The time corresponding to the TOIC value is related to the baud rate. For example, TOIC = 40, baud rate = 9600 bps. 40 represents the time required to transmit 40 bits. Calculated with a baud rate of 9600 bps, it is 4.17 ms. If the data format = 1 start bit + 8 data bits + 1 stop bit (no parity bit), TOIC = 40 is equivalent to 4 bytes. To avoid receiver time-out interrupt generation immediately during one character is being received, TOIC value should be set between 40 and 255.

```
void LPUART0_Init(void)
{
    /* Reset Low Power UART0 */
    SYS_ResetModule(LPUART0_RST);

    /* Configure Low Power UART0 and set Low Power UART0 Baudrate */
    LPUART_Open(LPUART0, 9600);

    /* Keep LPUART clock at power-down mode NPD3/NDP4 */
    LPSCC->CLKKEEP0 |= LPSCC_CLKKEEP0_LPUART0KEEP_Msk;

    /* Set Automatic Operation Enable */
    LPUART0->AUTOCTL |= LPUART_AUTOCTL_AOEN_Msk;

    /* Enable LPUART PDMA RX */
    LPUART0->INTEN |= LPUART_INTEN_RXPDMAEN_Msk;

    /* Enable bus idle time-out(Mode 1) and set time-out counter */
    LPUART0->TOUT =  255 | LPUART_TOUT_BITOMEN_Msk;

    /* Enable time-out counter */
    LPUART0->INTEN |= LPUART_INTEN_TOCNTEN_Msk;

    /* Enable Bus Idle Time-out Wake-up */
    LPUART0->AUTOCTL |= LPUART_AUTOCTL_WKAOTOEN_Msk;

    /* Enable RX Time-out interrupt */
    LPUART_EnableInt(LPUART0, LPUART_INTEN_RXTOIEN_Msk);

/* Enable LPUART0 IRQ */
```

```
    NVIC_EnableIRQ(LPUART0_IRQn);
}
```

Record the number of transmitted data

After the MCU wakes up, the number of transmitted data is calculated in interrupt function. There are two situations here:

- LPPDMA has not finished transmitting. The number of transmissions = the set number of LPPDMA − the remaining number of LPPDMA.

```
void LPUART0_IRQHandler(void)
{
    if (LPUART_GET_INT_FLAG(LPUART0, LPUART_INTSTS_HWTOINT_Msk))
    {
        /* Record the number of data */
        g_u32cnt = RXBUFSIZE - LPPDMA_GET_TRANS_CNT(LPPDMA0, LPUART_RX_LPPDMA_CH) - 1;
    }

/* Clear interrupt flag */
    LPUART0->INTSTS = LPUART0->INTSTS;
}
```

- When LPPDMA transmission is completed, the number of transmissions = the set number of LPPDMA.

Note: The LPPDMA transmission data length is set as 1024 bytes. If the transmission data exceeds 1024 bytes, data loss will occur. You can modify the value of RXBUFSIZE in the macro definition to change the maximum data size that can be received.

```
void LPPDMA0_IRQHandler(void)
{
  volatile uint32_t u32regISR;

  u32regISR = LPPDMA0->INTSTS;

  if ((u32regISR & LPPDMA_INTSTS_TDIF_Msk) == LPPDMA_INTSTS_TDIF_Msk)/* transfer done */
  {
    /* Record the number of data */
    g_u32cnt = RXBUFSIZE;

    printf("Warning:The received data is greater than or equal to the buffer size!!!\n");
  }

/* Clear interrupt flag */
  LPPDMA0->TDSTS = LPPDMA0->TDSTS;
}
```

Stop transmission after wake-up

Mainly to stop LPPDMA and disable LPUART from triggering LPPDMA and stop the LPUART bus idle timeout counter.

Ensure that the LPPDMA counter and the LPUART bus idle time-out counter will count again after the next power-down.

```
void Transfer_Stop(void)
{
    /* Disable LPUART PDMA RX and time-out counter */
    LPUART0->INTEN &= ~(LPUART_INTEN_RXPDMAEN_Msk | LPUART_INTEN_TOCNTEN_Msk);

    /* Stop LPPDMA */
    LPPDMA0->LPDSCT[1].CTL &= ~LPPDMA_DSCT_CTL_OPMODE_Msk;
}
```

Reconfigure transmission

It is to prepare for normal data reception and wake-up after the next power-down. LPPDMA does not need to reconfigure all registers, but mainly needs to configure the number of transmissions and operation mode. LPUART needs to be reset and reconfigured.

```
void Transfer_Recfg(void)
{
    ClearBuf((uint32_t)g_u8RecData, RXBUFSIZE, 0xFF);

    g_u32cnt = 0;

    /* Restart LPPDMA channel */
    /* Transfer count is RXBUFSIZE, transfer width is 8 bits(one byte) */
    LPPDMA_SetTransferCnt(LPPDMA0, LPUART_RX_LPPDMA_CH, LPPDMA_WIDTH_8, RXBUFSIZE);

    /* Request source is LPUART RX to memory */
    LPPDMA_SetTransferMode(LPPDMA0, LPUART_RX_LPPDMA_CH, LPPDMA_LPUART0_RX, FALSE, 0);

    /* Restart LPUART */
    LPUART0_Init();
}
```

System initialization

Note that in NPD4 mode, HCLK1 cannot exceed 4 MHz, so MIRC is chosen as the clock source for HCLK1.

```
void SYS_Init(void)
{
  /* Unlock protected registers */
  SYS_UnlockReg();

  /* Enable External XTAL (4~24 MHz) */
  CLK_EnableXtalRC(CLK_PWRCTL_HIRCEN_Msk);

  /* Switch HCLK clock source to HIRC */
  CLK->CLKSEL0 = (CLK->CLKSEL0 & ~CLK_CLKSEL0_HCLK0SEL_Msk) |
                 CLK_CLKSEL0_HCLKSEL_HIRC48M ;

  /* Switch HCLK1 clock source to MIRC */
  CLK->CLKSEL0 = (CLK->CLKSEL0 & ~CLK_CLKSEL0_HCLK1SEL_Msk) |
                 CLK_CLKSEL0_HCLK1SEL_MIRC ;

  /* Select UART0 clock source is HIRC and UART module clock divider as 1 */
  CLK_SetModuleClock(UART0_MODULE, CLK_CLKSEL4_UART0SEL_HIRC, CLK_CLKDIV0_UART0(1));

  /* Select LPUART0 clock source is MIRC and LPUART module clock divider as 1*/
```

```
    CLK_SetModuleClock(LPUART0_MODULE, LPSCC_CLKSEL0_LPUART0SEL_MIRC,
                       LPSCC_CLKDIV0_LPUART0(1));

    /* Enable UART0 peripheral clock */
    CLK_EnableModuleClock(UART0_MODULE);

    /* Enable LPUART0 peripheral clock */
    CLK_EnableModuleClock(LPUART0_MODULE);

    /* Enable LPSRAM clock */
    CLK_EnableModuleClock(LPSRAM_MODULE);

    /* LPPDMA Clock Enable */
    CLK_EnableModuleClock(LPPDMA0_MODULE);

    /* Set GPIO mode for unused pins */
    GPIO_Init();

    /* Set GPB multi-function pins for UART0 RXD and TXD */
    Uart0DefaultMPF();

    /* Set PA multi-function pins for LPUART0 TXD and RXD */
    SYS->GPA_MFP0 = (SYS->GPA_MFP0 & ~(SYS_GPA_MFP0_PA0MFP_Msk | SYS_GPA_MFP0_PA1MFP_Msk)) |
                    (SYS_GPA_MFP0_PA0MFP_LPUART0_RXD | SYS_GPA_MFP0_PA1MFP_LPUART0_TXD);

    /* Lock protected registers */
    SYS_LockReg();
}
```

Main function

```
int main(void)
{
    uint32_t i, count = 0;

    /* Init System, peripheral clock and multi-function I/O */
    SYS_Init();

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

    /*---------------------------------------------------------------------------------------------*/
    /* LPUART Receives Variable Length Data in Power-down Mode Test                                */
    /* 1. LPUART uses LPPDMA Channel-0 to transfer RX data to g_u8RecData                          */
    /* 2. LPUART RX pin (PA.0) is connected to UART Tool                                           */
    /* 3. System enters power-down mode and UART RX receives data in power-down mode               */
    /* 4. When RX transfer done or timed out and wake-up system                                    */
    /* 5. Print the received data through uart0                                                    */
    /* 6. Repeat steps 3~5                                                                         */
    /*---------------------------------------------------------------------------------------------*/
    printf("+-----------------------------------------------------------------------+\n");
    printf("|     M2L31 LPUART receive under Power-Down mode Sample Code            |\n");
    printf("+-----------------------------------------------------------------------+\n\n");
    printf("  >> Please connect PA.0 to PC by UART tool. << \n");
    printf("     Press any key to start test\n\n");

    getchar();

    /* Clear buffer */
    ClearBuf((uint32_t)g_u8RecData, RXBUFSIZE , 0xFF);
```

```
    /* LPPDMA CH-0 basic Mode RX to receive data */
    LPPDMA_RX_Init();

    /* Init LPUART0 as Auto-operation Mode for test */
    LPUART0_Init();

    while (1)
    {
        /* Reconfigure transmission */
       Transfer_Recfg();

        count++;

        /* Enter power down mode */
        PowerDown_Enter();

        printf("\nWakeup %d\n", count);

        /* Wait for the number of data to be recorded */
        while (g_u32cnt == 0);

       Transfer_Stop();

        /* Add data processing code */
        printf("Received Data(%d):\n", g_u32cnt);

        for (i = 0; i < g_u32cnt; i++)
            printf("%c", g_u8RecData[i]);

        printf("\n\n");
    }
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M2L31_Series_BSP_CMSIS_V3.00.000
- IDE version
  - Keil uVersion 5.38.0.0

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M2L31KI V2.1
- Pin Connect
  - Connect the UART0 TX (PB.13) pin to PC via UART to USB Tool.
  - Connect the UART0 RX (PB.12) pin to PC via UART to USB Tool.
  - Connect the LPUART RX (PA.0) pin to PC via UART to USB Tool.

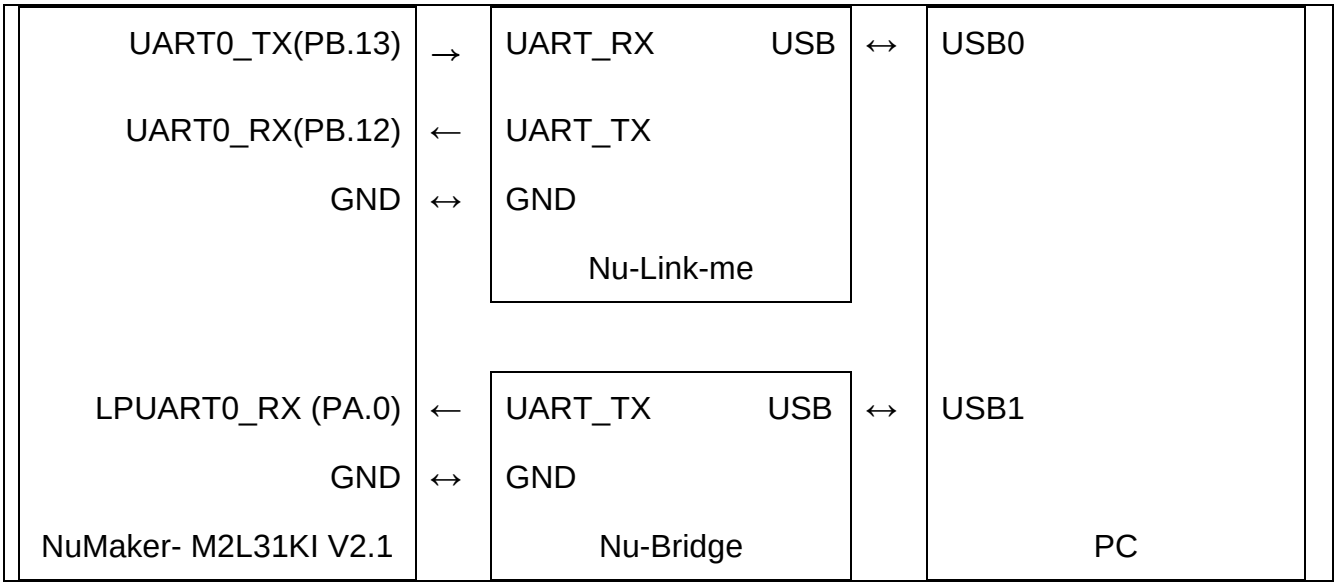

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M2L31_LPUART_RX_in_PD_V1.01	
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	
Project	Source file of example code

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 *M2L31_LPUART_RX_in_PD.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2025.03.07	1.01	Remove disable LVR code to prevent power on and off issues.
2024.04.19	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.