Simulating LIN Bus Master Code Using M3334 UART

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code implements LIN bus master transmission and reception using UART1 on the M3334 series microcontroller.
BSP Version	M3334_Series_BSP_CMSIS_V3.00.002
Hardware	NuMaker-M3334KI V2.0

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

The LIN bus is a UART/SCI-based low-cost interface with a single master and multiple slave structure. The maximum rate is 20 kbps and 19,200 bps or 9,600 bps is commonly used. Its frame structure starts at 13 bits of 0 followed by multiple bytes of standard UART communication: 0x55 + ID + data + checksum. The numbers of data bytes are agreed upon by both parties, up to 8 bytes. 13 bits 0 + 0x55 + ID is the frame header and can only be sent by the master. The slave can only send data after receiving a frame header matching its own ID. Therefore, the master must periodically send frame headers so that the slave can send data promptly.

This example code implements LIN host functionality using UART. The code defines a list of six messages sent in a loop at 20 ms intervals timed by a Timer. After a frame header is sent, if the host should send data next, the host will continue to send data. If the host should receive data next, it starts receiving data. The bus timing is shown in Figure 1-1.

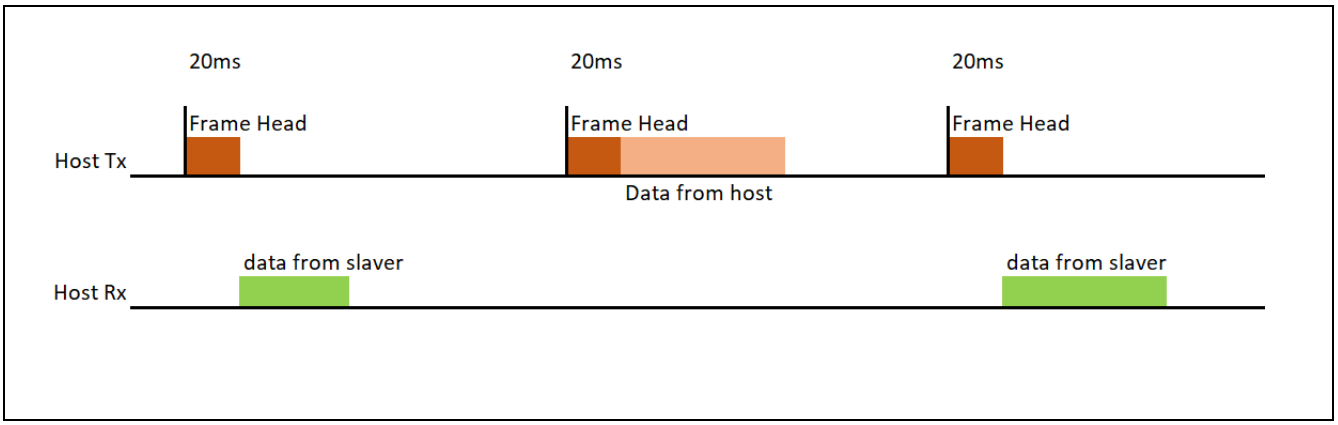


Figure 1-1 Bus Timing

1.1 Principle

The frame header structure is shown in Figure 1-2.

- The frame ID values range, from 0 to 0x3F, totaling 64, are placed in bit 0 ~ bit 5 of the ID segment.
- The ID segment bit 6 = the XOR values of bit 0, bit 1, bit 2, and bit 4. Bit 7 = the inverted XOR values of bit 1, bit 3, bit 4, and bit 5.
- The maximum baud rate is 20 kbps, and in this example, it is taken as 19,200 bps.

During the 20 ms interrupt of the Timer, the baud rate is lowered to 9/13 and one byte 0 is sent, which produces 13-bit 0 of the normal baud rate on the bus. After 13-bit 0 is sent completely, the 0x55 and ID is sent in the UART interrupt. At that point, the frame header is ending transmission.

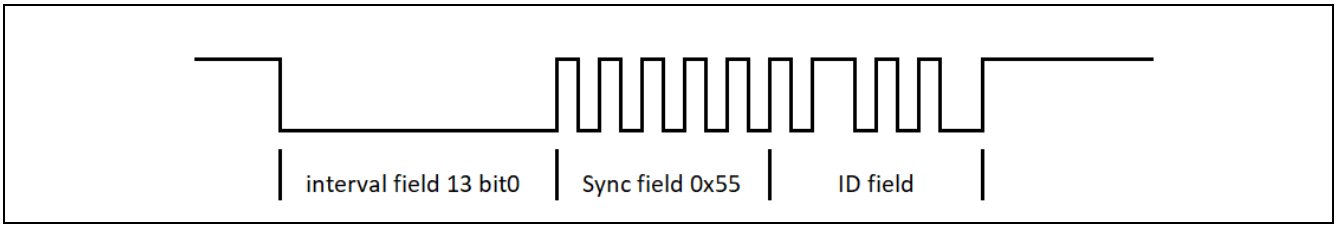


Figure 1-2 LIN Bus Frame Header

After the 0x55 and ID are sent in the UART interrupt, if the current frame is a host send frame, host will continue sending data. If the current frame is a host receive frame, the code will configure the receive length and prepare for reception.

1.2 Demo Result

The code defines six messages Message_Buffer [6]: four for sending and two for receiving. To facilitate checking the execution effect of this code, the main function periodically initiates sending messages with ID equal to 0x05 and prints information as shown in Figure 1-3.

```
=== UART1 emulate LIN Master ===

LIN Master transmit:
ID    DATA                          CHECKSUM
0x05, 0x02, 0x02, 0x03, 0x04,       0xF4

LIN Master transmit:
ID    DATA                          CHECKSUM
0x05, 0x03, 0x02, 0x03, 0x04,       0xF3

LIN Master transmit:
ID    DATA                          CHECKSUM
0x05, 0x04, 0x02, 0x03, 0x04,       0xF2

LIN Master transmit:
ID    DATA                          CHECKSUM
0x05, 0x05, 0x02, 0x03, 0x04,       0xF1
```

Figure 1-3 Printed Information 1

If a LIN slave is attached, the print interface can also print out the received message information, as shown in Figure 1-4.

```
LIN Master receive:
ID    DATA                                      CHECKSUM
0x03, 0x2B, 0x2B, 0x3C, 0x4D, 0xA2, 0x6F,       0x0E

LIN Master transmit:
ID    DATA                            CHECKSUM
0x05, 0x36, 0x02, 0x03, 0x04,         0xC0

LIN Master transmit:
ID    DATA                            CHECKSUM
0x05, 0x37, 0x02, 0x03, 0x04,         0xBF

LIN Master receive:
ID    DATA                                      CHECKSUM
0x03, 0x2C, 0x2B, 0x3C, 0x4D, 0xB2, 0x6F,       0xFC
```

Figure 1-4 Printed Information 2

2. Code Description

Timer1 20 ms interrupt subroutines

In interrupt handler, the baud rate is reduced to 9/13 of its original value, and a byte 0x00 is transmitted first. Then the bit6 and bit7 of the ID is recalculated, and the UART transmit interrupt is enabled. The UART receive interrupt is disabled to terminate reception of the previous frame.

```
void TMR1_IRQHandler(void)
{
    int32_t temp32, temp32_1;

    Timer1_Period_Dec --;

    /*
     * Generate LIN Break by transmitting UART 0x00 at a reduced
     * baud rate. UART 0x00 has 9 LOW bits (start + 8 data bits).
     * Scaling the baud rate makes these approximately 13 LIN bits.
     *
     * At 19.2 kbps:
     *     13 LIN bits = 677.08 us
     *
     * Verify the actual waveform because the UART baud-rate divider
     * may introduce a small timing error.
     */
    UART_Open(UART1, LIN_BREAK_BAUD_RATE);
    UART1->DAT = 0x00;
    Frame_Head = TRUE;

    if (++ptr_Msg > Message_End) ptr_Msg = Message_Buffer;

    /* Clear TIMER1 time-out interrupt flag */
    TIMER_ClearIntFlag(TIMER1);
    /* bit6 = bit1 ^ bit4 */
    temp32   = (ptr_Msg->PID << 5) ^ (ptr_Msg->PID << 2);
    /* bit2 = bit0 ^ bit2 */
    temp32_1 = (ptr_Msg->PID << 6) ^ (ptr_Msg->PID << 4);
    /* bit6 = bit0 ^ bit1 ^ bit2 ^ bit4 */
    temp32_1 = (temp32 ^ temp32_1) & 0x40;

    temp32 <<= 1;
    temp32  ^= ptr_Msg->PID << 4;
    temp32  ^= ptr_Msg->PID << 2;
    temp32  ^= 0x80;

    /* bit7 = ~(bit1 ^ bit3 ^ bit4 ^ bit5) */
    temp32 &= 0x80;
    /* Calculate bit6,7 of PID */
    ptr_Msg->PID = (ptr_Msg->PID & 0x3F) | temp32 | temp32_1;

    if (ptr_Msg->DataNum > 8) ptr_Msg->DataNum = 8;

    /* Last frame stop receiving */
    UART_DisableInt(UART1, UART_INTEN_RXTOIEN_Msk);
```

```
    /* 13bit zero finishing will occur interrupt */
    UART_EnableInt(UART1, UART_INTEN_TXENDIEN_Msk);
}
```

Frame structure

The member Tx_Frame_Cnt is only used to send data frames. Each time the corresponding ID data frame is sent, it is deducted by 1. When it is reduced to 0, the data frame corresponding ID will no longer be sent, but the frame header will still be sent. Therefore, when this value equals 0, the data in the frame can be updated without causing data aliasing error.

Rx_Frame_Num is only used to receive data frames. Each time a complete data frame corresponding to the ID is received, it is added by 1. When this value equals 0, the received data is not updated to the frame list structure. In this case, the data in the frame list can be read out without aliasing.

```
typedef struct
{
    /* 1: This frame will transmit data.  0: This frame is for reception */
    uint8_t  Flag_Master_Tx;
    /* Message ID */
    uint8_t  PID;
    /* Data for transmitting */
    uint8_t  TxData[9];
    /* data number, Max =8 */
    uint8_t  DataNum;
    /* received (data + Checksum) */
    uint8_t  RxData[12];
    /* when Tx_Frame_Cnt != 0, data TxData[] will be transmitted */
    uint8_t  Tx_Frame_Cnt;
    /* If Rx_Frame_Num=0, RxData[] won't be updated. */
    uint8_t  Rx_Frame_Num;
    /* If this equals 0 checksum is classic */
    uint8_t  CheckSum_Type;
} Message_Type;
```

UART interrupt subroutines

At the beginning of interrupt subroutines, non-zero Frame_Head indicates that Timer1 has just sent 13 bits of 0 previously. The baud rate is restored to 19200. Then the frame head data 0x55 and the frame ID is sent.

Next, it is judged that the current frame is whether sending frame or receiving frame by the master based on Flag_Master_Tx in the frame structure data. If the master sends data, all the data can be written into the sending cache of UART TxD at once because the M3334 UART has 16 levels of cache while the LIN bus has a maximum 8 bytes of data following a checksum byte. If the master receives, it is necessary to wait until the 0x55+ID is sent completely before starting the reception. The purpose is that when single line is used, it does not receive the 0x55+ID sent by itself.

```
void  UART1_IRQHandler(void)
{
    int32_t static TxCnt, RxCnt, CntMax;
```

```
    uint8_t static RxStr_Buf[11];
    uint8_t temp8u;

    /* If frame header has just been transmitted out. */
    if (Frame_Head == TRUE)
    {
        /* Transmit frame head */
        UART_Open(UART1, LIN_BAUD_RATE);
        UART1->DAT = 0x55;
        UART1->DAT = ptr_Msg->PID;

        /* Disable UART Tx interrupt */
        UART_DisableInt(UART1, UART_INTEN_TXENDIEN_Msk);
        Frame_Head = FALSE;

        CntMax = ptr_Msg->DataNum;

        /* Master transmits data subsequently */
        if (ptr_Msg->Flag_Master_Tx)
        {
            /* When CPU updates the data, please set Tx_Frame_Cnt = 0 */
            if (ptr_Msg->Tx_Frame_Cnt)
            {
                /* Decrease every transmission */
                ptr_Msg->Tx_Frame_Cnt --;
                TxCnt = 0;

                while (TxCnt <= CntMax)
                {
                    /* Transmits data and CheckSum */
                    UART1->DAT = ptr_Msg->TxData[TxCnt++];
                }
            }
        }
        /* Master receive data subsequently. */
        else
        {
            /* Wait for 0x55+PID is transmitted completely */
            while ((UART1->FIFOSTS & UART_FIFOSTS_TXEMPTYF_Msk) == 0);

            RxCnt = 0;
            /* Clear Rx FIFO for acceptance */
            UART1->FIFO |= UART_FIFO_RXRST_Msk;

            /* Waiting for Rx FIFO clear completion */
            while ((UART1->FIFOSTS & UART_FIFOSTS_RXIDLE_Msk) == 0) __NOP();

            /* Enable TxD Timer over interrupt */
            UART_EnableInt(UART1, UART_INTEN_RXTOIEN_Msk);
        }

    }
    /* Rx data process */
    else
    {
        /* RxD Timer over interrupt */
```

```
        while (UART1->INTSTS & UART_INTSTS_RXTOINT_Msk)
        {
            while ((UART1->FIFOSTS & UART_FIFOSTS_RXEMPTY_Msk) == 0)
            {
                /* Read out received data */
                temp8u = UART1->DAT;

                /* Copy data and CheckSum */
                if (RxCnt <= CntMax) RxStr_Buf[RxCnt++] = temp8u;
                /* After receiving this frame Data field, disable UART1 TO interrupt */
                else UART_DisableInt(UART1, UART_INTEN_RXTOIEN_Msk);
            }

            /* Last byte, checksum has been received */
            if (RxCnt == CntMax + 1)
            {
                /*
                 * Accept the response only when the LIN checksum valid.
                 */
                if (LIN_Checksum_Verify(ptr_Msg->PID,
                                        RxStr_Buf,
                                        CntMax,
                                        ptr_Msg->CheckSum_Type))
                {
                    /* If Rx_Frame_Num=0, data updates here are prohibited */
                    if (ptr_Msg->Rx_Frame_Num)
                    {
                        for (RxCnt = 0; RxCnt <= CntMax; ++RxCnt)
                        {
                            /* Copy data and CheckSum */
                            ptr_Msg->RxData[RxCnt] = RxStr_Buf[RxCnt];
                        }

                        if (++ptr_Msg->Rx_Frame_Num == 0) ptr_Msg->Rx_Frame_Num = 1;
                    }
                }
                else
                {
                    /* Invalid LIN checksum: discard the response. */
                    LIN_Rx_Checksum_Error++;
                }
            }
        }
    }
}
```

When the number of received data is equal to the length of the frame data by one more byte, it indicates that the checksum byte has been received. The received data can be transferred from RxStr_Buf [ ] to the corresponding array in the frame list. No data will be transferred if the member Rx_Frame_Num=0. When reading data from the frame list, the corresponding Rx_Frame_Num can be set to 0 at first to prevent data aliasing.

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M3334_Series_BSP_CMSIS_V3.00.002
- IDE version
  - Keil uVision 5.43.1.0
  - VS Code 1.115.3

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M3334KI V2.0
  - If no other LIN slave is available, NuMaker-M2003FCV1.1 can be used with the example code “EC_M2003_UART_LIN_Slave_V1.00” for LIN bus communication.
- Hardware Connection

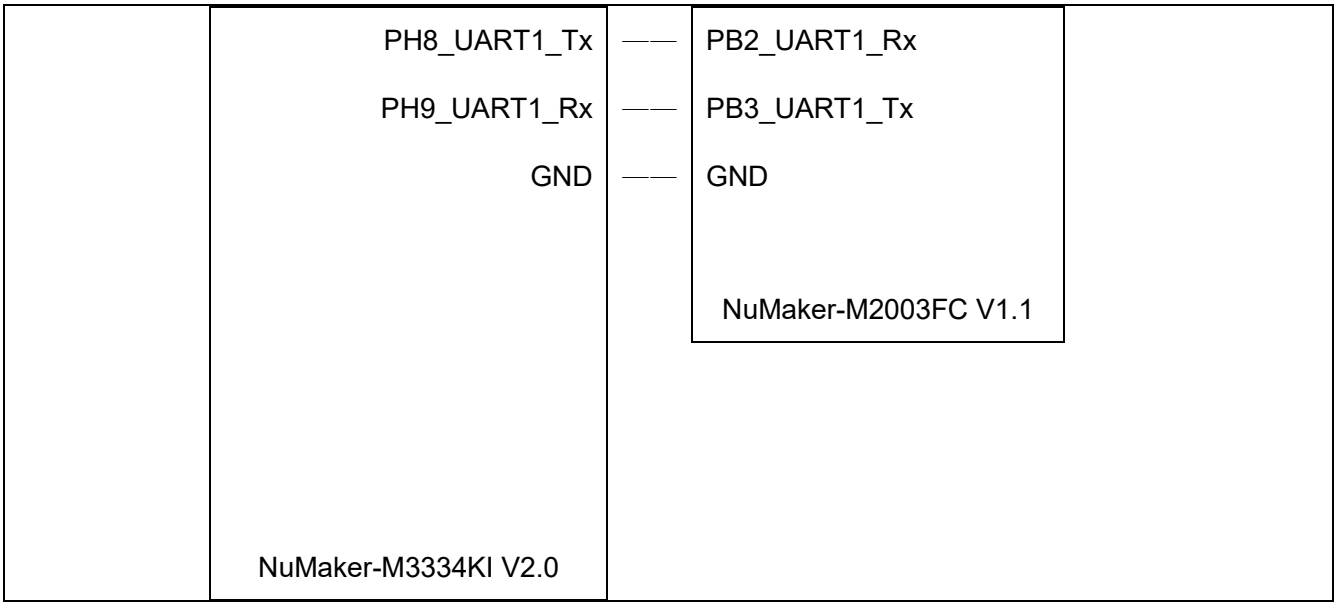


Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M3334_UART_LIN_Master_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	
M3334_UART_LIN_Master	Source file of example code

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 *M3334_UART_LIN_Master.uvprojx* project file.
2. Build
3. Download
4. Enter debug mode.
   - Run

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

6. Revision History

Date	Revision	Description
2026.09.19	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.