Simulating LIN Bus Slave Code Using M2003 UART

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code implements LIN bus slave transmission and reception using UART on the M2003 series microcontroller.
BSP Version	M2003_Series_BSP_CMSIS_V3.00.001
Hardware	NuMaker-M2003FC 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 implements the LIN bus slave function using the M2003 series microcontroller (MCU). Two message frames are defined in the code. One is used for sending data, and the other is used for receiving data.

Messages on the LIN bus start with "at least 13-bit 0" followed by 0x55, packet ID, data, and checksum. The data field cannot have more than 9 bits of 0. The slave can transmit data only after receiving an ID that matches itself. The timing is shown in Figure 1-1 .

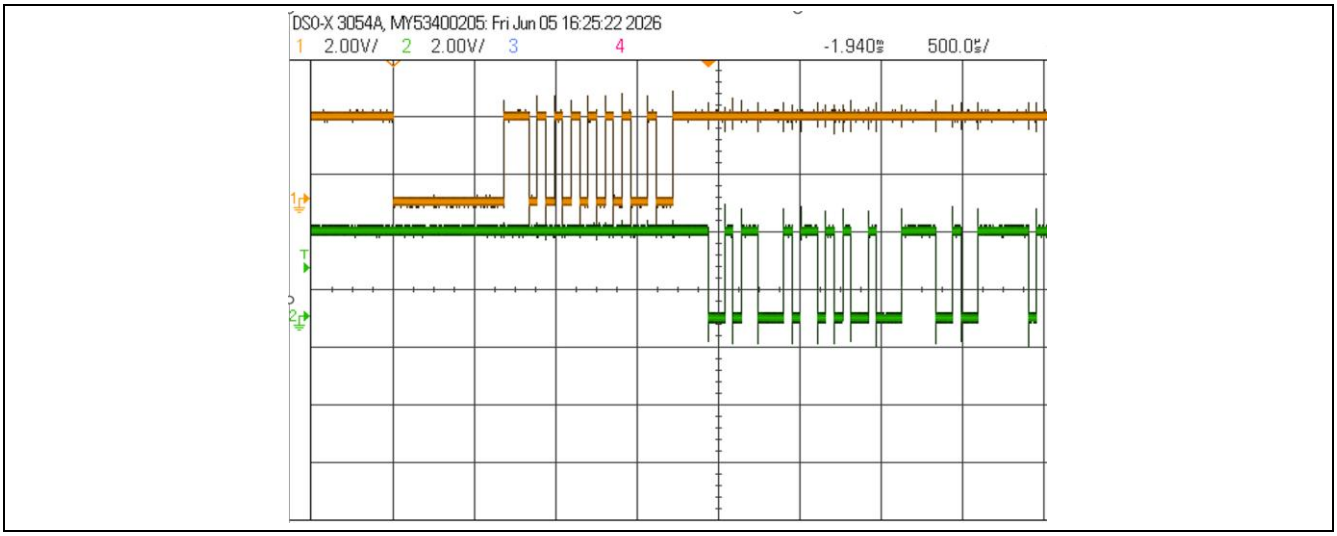


Figure 1-1 LIN Bus Data

1.1 Principle

To simplify the code for receiving and sending data from LIN bus slave devices, we use the break interrupt of UART to detect the frame headers on the bus. After a break interrupt is detected, data reception begins. The first byte should be 0x55, the second byte is ID, and the subsequent steps are receiving or sending data.

When operating as a LIN slave, when the M2003 transmits data, all data bytes are written to the UART transmit FIFO. The maximum amount of data sent is 8 bytes plus a check code. The M2003 UART includes a 16-byte FIFO.

The LIN slave can receive a maximum of 11 bytes of data: 0x55+ID+Data+Checksum. The received data is temporarily stored in the array StringRx[ ]. After all the data of the current message is received, it is transferred to the message structure array.

1.2 Demo Result

The LIN slave can only send and receive messages after receiving an ID that matches itself on the bus. Therefore, to test this code, it must be connected to the LIN bus. The M2003 UART0 will print out the message information for sending and receiving, as shown in Figure 1-2.

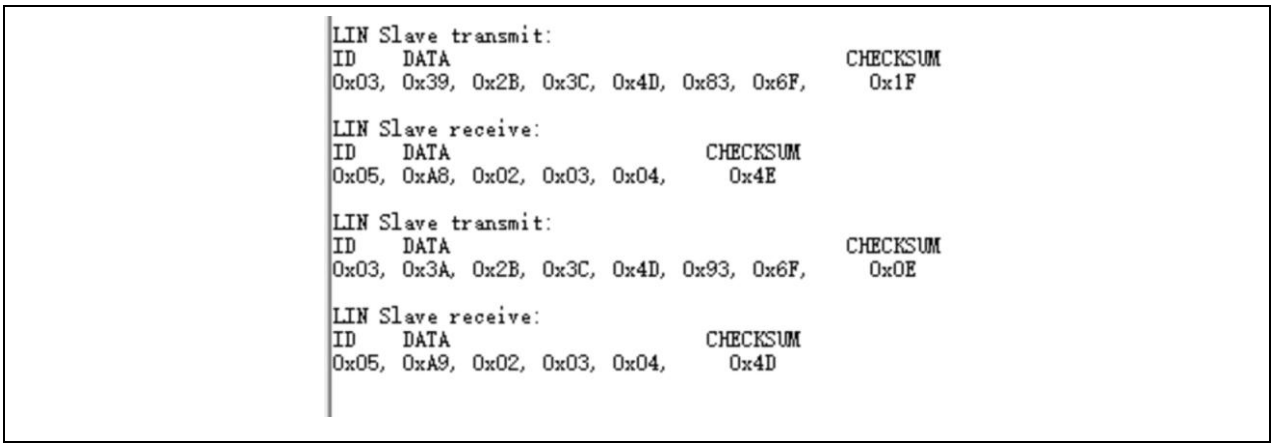


Figure 1-2 Printed Information

2. Code Description

LIN bus data structure

Only when the member TX_Frame_Cnt of the structure is not 0 and receiving a frame header containing a matching ID, the data corresponding to this ID will start sending to bus. Please update sent data only when TX_Frame_Cnt=0, which can ensure that the data is not being sent at the same time of update. This prevents data aliasing.

```
typedef struct
{
    uint8_t  PID;
    /* If frame is for reception, this will be written : 0x55 + PID + Data + CheckSum */
    uint8_t  Data[12];
    /* Data number. Only data number exclude PID 0x55 or others. */
    uint8_t  DataNum;
    /* It will decrease after transmission and stop transmission when it equals 0 */
    uint8_t  TX_Frame_Cnt;
    /* When it equals 0, Data[ ] won't be updated */
    uint8_t  Rx_Frame_Num;
    /* 0 = Classic checksum, 1 = Enhanced checksum */
    uint8_t  CheckSum_Type;
} Message_Slave_Type ;

/*---------------------------------------------*/
/* Define global variables and constants       */
/*---------------------------------------------*/
/* The checksum type must match the Master configuration for the same PID. */
Message_Slave_Type  Message_Slave_TxData =
{
    0x03, {0x1A, 0x2B, 0x3C, 0x4D, 0x5E, 0x6F}, 6, 0, 1, 0
};

/* Here random number is only for example. */
Message_Slave_Type  Message_Slave_RxData =
{
    0x05, { }, 4, 0, 1, 0
};

/* 0x55 + PID + Data + CheckSum = [0]~[10], StringRx[11] is no used. */
uint8_t StringRx[12];
```

Only two structures, one receiving and one transmitting, are defined in the code. If necessary, additional structures can be defined.

UART interrupt subroutines

This code uses the break interrupts of UART to detect the frame header. Once the frame header is detected, the data on the bus is received and saved to the array StringRx[]. The first byte received is 0x55. After receiving the second byte (the ID), if it matches the ID of a specific message, the transmission and reception of message's data will begin.

After the message is received, the data is copied from the temporary array StringRx[] to the message structure array only when Rx_Frame_Num is non-0. Other tasks should set Rx_Frame_Num to 0 first before reading data, which can prevent data update aliasing.

```
void UART1_IRQHandler(void)
{
    uint8_t temp8;
    uint32_t INT_Status, FIFO_Status;
    Message_Slave_Type static *ptr;

    uint8_t static RxCnt = 11, RxCnt_Max = 11;

    /* Read out interrupt status */
    INT_Status  = UART1->INTSTS;
    /* Read out FIFO status */
    FIFO_Status = UART1->FIFOSTS;

    /* Received a byte */
    if (INT_Status & UART_INTSTS_RDAIF_Msk)
    {
        /* Read out the received byte */
        temp8 = UART1->DAT;

        /* Receive Line interrupt */
        if (INT_Status & UART_INTSTS_RLSIF_Msk)
        {
            /* Break Interrupt */
            if (FIFO_Status & UART_FIFOSTS_BIF_Msk)
            {
                /* Frame head is received */
                if (temp8 == 0) RxCnt = 0;
            }
        }
        /* StringRx[0]=0x55, StringRx[1]=PID, */
        else StringRx[RxCnt++] = temp8;

        if (RxCnt == 2)
        {
            /*
             * PID received = the PID of the Slave response frame.
             * The Master must send this header before the Slave transmits.
             */
            if (temp8 == Message_Slave_TxData.PID)
            {
                /* This value will avoid processing received data */
                RxCnt_Max = 11;

                /* Permit transmission && (Datanum != 0) */
                if ((Message_Slave_TxData.TX_Frame_Cnt) && (Message_Slave_TxData.DataNum))
                {
                    for (temp8 = 0; temp8 <= Message_Slave_TxData.DataNum; ++temp8)
                    {
                        /* Transmit data */
                        UART1->DAT = Message_Slave_TxData.Data[temp8];
                    }
```

```
                    Message_Slave_TxData.TX_Frame_Cnt --;
                }
            }
            /* PID received = the PID of Message 4 */
            else if (temp8 == Message_Slave_RxData.PID)
            {
                ptr = &Message_Slave_RxData;
                /* Prepare to receive data */
                RxCnt_Max = Message_Slave_RxData.DataNum;

                if (RxCnt_Max > 8) RxCnt_Max = 8;
            }

        }
        /* CheckSum has been received into StringRx[RxCnt_Max+2] */
        else if (RxCnt == RxCnt_Max + 3)
        {
            /* Received data will be saved only when this value is non-zero */
            if (ptr->Rx_Frame_Num)
            {
                /* Avoid the value becoming 0 and discard the received data */
                if (LIN_Checksum_Verify(ptr->PID,
                                        &StringRx[2],
                                        ptr->DataNum,
                                        0))
                {
                    for (temp8 = 0; temp8 < RxCnt_Max + 3; ++temp8)
                    {
                        /* Copy data: 0x55 + PID + Data + CheckSum */
                        ptr->Data[temp8] = StringRx[temp8];
                    }

                    /* When read the data, Rx_Frame_Num=0 can avoid data overwriting. */
                    if (++ptr->Rx_Frame_Num == 0) ptr->Rx_Frame_Num = 1;
                }
            }
        }

        /* StringRx[11] will be discarded */
        if (RxCnt > 11) RxCnt = 11;
    }

    UART1->FIFOSTS |= UART_FIFOSTS_PEF_Msk
                    | UART_FIFOSTS_FEF_Msk
                    | UART_FIFOSTS_BIF_Msk;
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M2003_Series_BSP_CMSIS_V3.00.001
- IDE version
  - Keil uVision 5.43.1.0
  - VS Code 1.115.3

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M2003FC V1.1
  - - If no other LIN slave is available, NuMaker-M3334KI V2.0 can be used with the example code “EC_M3334_UART_LIN_Master_V1.00” for LIN bus communication.
  - After connecting the UART1 interfaces of the two boards, data can be sent and received to each other.

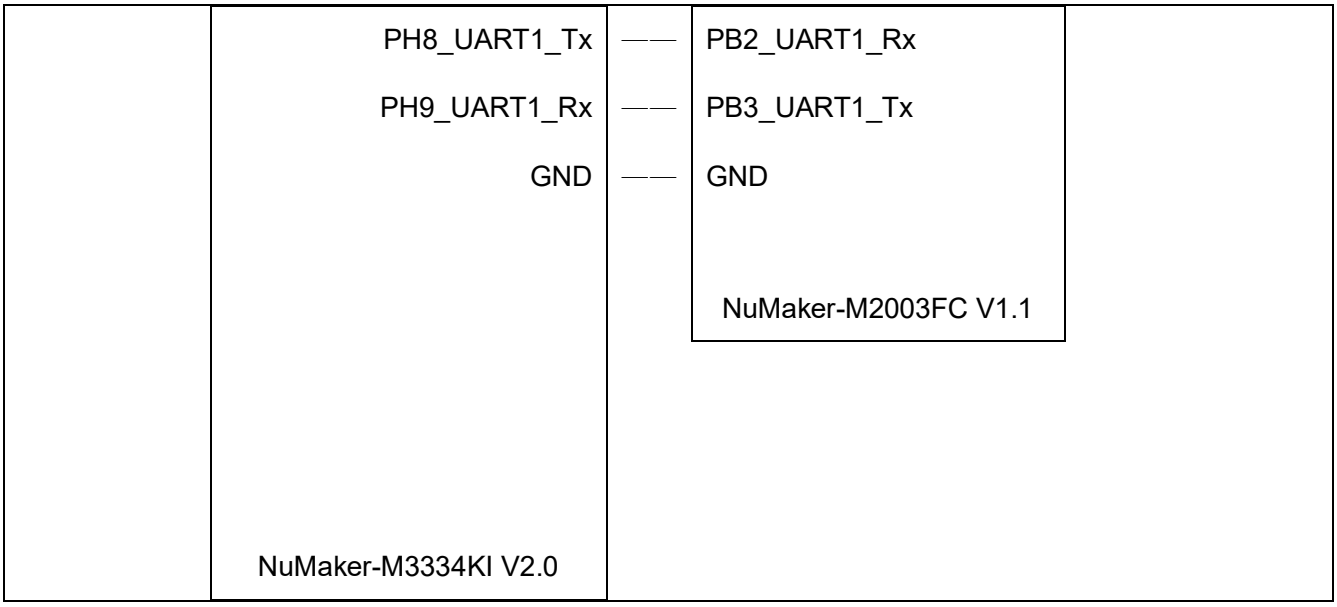


Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M2003_UART_LIN_Slave_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	
M2003_UART_LIN_Slave	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 *M2003_UART_LIN_Slave.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
2026.08.19	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.