LIN Bus Data Transfer

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	The code is a demonstration of how to transfer data through LIN Bus
BSP Version	NUC230_240 Series BSP CMSIS v3.01.001
Hardware	NuTiny-EVB-NUC240 v1.2

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 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

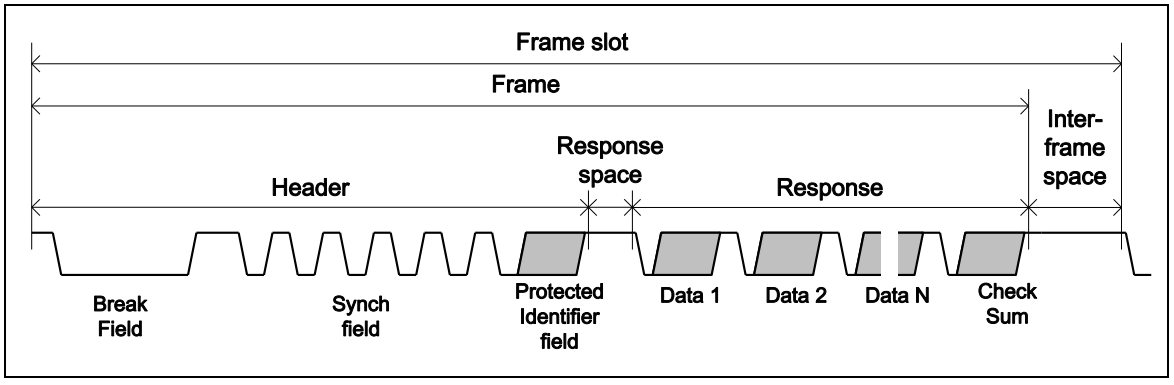
1 Function Description

1.1 Introduction

LIN (Local Interconnect Network) is a serial network protocol used for communication between components and supports a broadcast serial network comprising 16 nodes (one master and up to 15 slaves). This sample code demonstrates how to transmit/receive LIN message frame through LIN bus.

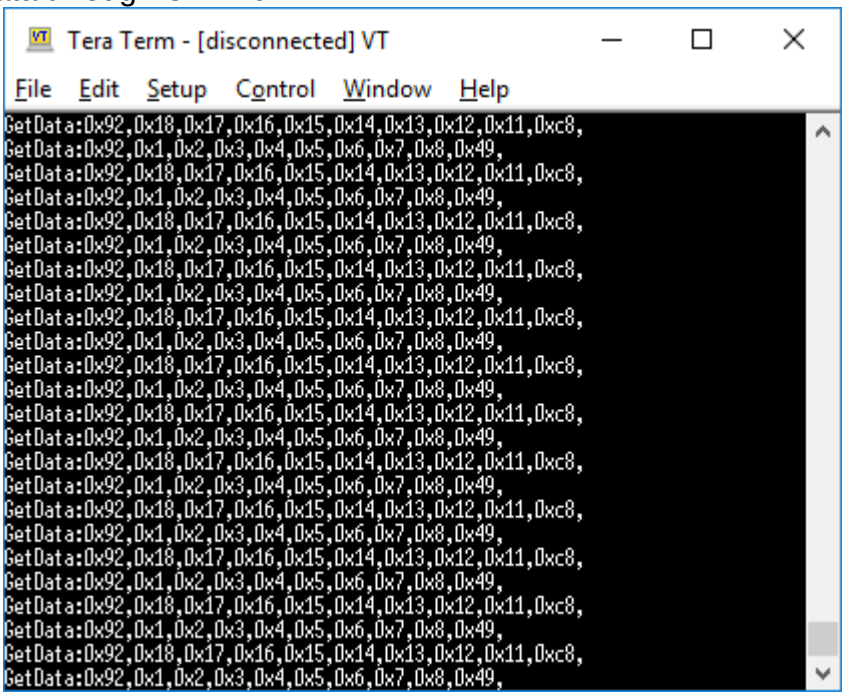
1.2 Principle

According to the LIN protocol, all information transmitted is packed as frames; a frame consists of a header (provided by the master task) and a response (provided by a slave task), followed by a response (provided by a slave task). The header (provided by the master task) consists of a break field and a sync field followed by a frame identifier (frame ID). The frame identifier uniquely defines the purpose of the frame. The slave task is appointed for providing the response associated with the frame ID. The response consists of a data field and a checksum field. The below figure is the structure of LIN Frame.



1.3 Demo Result

In this example, the UART1 is set as LIN function to transmit or to receive LIN frames that include 8-byte data. When communication is successful, the slave device will print message to show 8-byte data through UART0.



2 Code Description

2.1.1 UART Initial setting

```
void UART0_Init()
{
    /*---------------------------------------------------------------*/
    /* Init UART                                                     */
    /*---------------------------------------------------------------*/
    /* Reset UART0 module */
     SYS_ResetModule(UART0_RST);

    /* Configure UART0 and set UART0 Baudrate */
     UART_Open(UART0, 115200);
}

void UART1_Init()
{
    /*---------------------------------------------------------------*/
    /* Init UART                                                     */
    /*---------------------------------------------------------------*/
    /* Reset UART1 module */
     SYS_ResetModule(UART1_RST);

    /* Configure UART1 and set UART1 Baudrate */
     UART_Open(UART1, 9600);
}
```

2.1.2 Data Transmit through LIN Master

```
    /*---------------------------------------------------------------------*/
    /* MAIN function                                                       */
    /*---------------------------------------------------------------------*/
    int main(void)
    {
        …
        while (1)
        {
            i++;
            if (i & 1)
            {
```

```
                au8TestPattern[0] = 0x01;// 8 data byte
                au8TestPattern[1] = 0x02;
                au8TestPattern[2] = 0x03;
                au8TestPattern[3] = 0x04;
                au8TestPattern[4] = 0x05;
                au8TestPattern[5] = 0x06;
                au8TestPattern[6] = 0x07;
                au8TestPattern[7] = 0x08;
                au8TestPattern[8] = 0x00;// 1 byte checksum
                LIN_MasterTestUsingLinCtlReg(0x12, MODE_ENHANCED,au8TestPattern);
            }
            else
            {
                au8TestPattern[0] = 0x18;// 8 data byte
                au8TestPattern[1] = 0x17;
                au8TestPattern[2] = 0x16;
                au8TestPattern[3] = 0x15;
                au8TestPattern[4] = 0x14;
                au8TestPattern[5] = 0x13;
                au8TestPattern[6] = 0x12;
                au8TestPattern[7] = 0x11;
                au8TestPattern[8] = 0x00;// 1 byte checksum
                LIN_MasterTestUsingLinCtlReg(0x12, MODE_ENHANCED,au8TestPattern);
            }
            CLK_SysTickDelay(100000);
        }
        …
    }
    /*---------------------------------------------------------------------*/
    /* Master send header and response                                     */
    /*---------------------------------------------------------------------*/
    void LIN_MasterTestUsingLinCtlReg(uint32_t u32id, uint32_t u32ModeSel, uint8_t* p8data)
    {
        uint32_t i;
        if (u32ModeSel == MODE_CLASSIC)
        {
            /* Send break+sync+ID */
            LIN_SendHeaderUsingLinCtlReg(u32id, UART_LIN_CTL_LIN_HEAD_SEL_BREAK_SYNC_ID);
            /* Compute checksum without ID and fill checksum value to au8TestPattern[8] */
            p8data[8] = ComputeChksumValue(p8data, 8);
            UART_Write(UART1, p8data, 9);
```

```
        }
        else if (u32ModeSel == MODE_ENHANCED)
        {
            /* Send break+sync+ID and fill ID value to g_u8SendData[0]*/
            LIN_SendHeaderUsingLinCtlReg(u32id, UART_LIN_CTL_LIN_HEAD_SEL_BREAK_SYNC);

            /* Fill test pattern to g_u8SendData[1]~ g_u8SendData[8] */
            for (i = 0; i < 8; i++)
                g_u8SendData[g_i32pointer++] = p8data[i];

            /* Compute checksum value with ID and fill checksum value to g_u8SendData[9] */
            p8data[8] = ComputeChksumValue(g_u8SendData, 9) ;
            UART_Write(UART1, p8data, 9);
        }
    }
    /*---------------------------------------------------------------------*/
    /* Compute Parity Value                                                */
    /*---------------------------------------------------------------------*/
    uint8_t GetParityValue(uint32_t u32id)
    {
        uint32_t u32Res = 0, ID[6], p_Bit[2], mask = 0;

        for (mask = 0; mask < 6; mask++)
            ID[mask] = (u32id & (1 << mask)) >> mask;

        p_Bit[0] = (ID[0] + ID[1] + ID[2] + ID[4]) % 2;
        p_Bit[1] = (!((ID[1] + ID[3] + ID[4] + ID[5]) % 2));

        u32Res = u32id + (p_Bit[0] << 6) + (p_Bit[1] << 7);
        return u32Res;
    }
    /*---------------------------------------------------------------------*/
    /* Compute CheckSum Value , MODE_CLASSIC:(Not Include ID)              */
    /*                          MODE_ENHANCED:(Include ID)                 */
    /*---------------------------------------------------------------------*/
    uint32_t GetCheckSumValue(uint8_t *pu8Buf, uint32_t u32ModeSel)
    {
        uint32_t i, CheckSum = 0;

        for (i = u32ModeSel; i <= 9; i++)
```

```
        {
            CheckSum += pu8Buf[i];

            if (CheckSum >= 256)
                CheckSum -= 255;
        }
        return (255 - CheckSum);
    }
    /*---------------------------------------------------------------------*/
    /*  Send LIN Header Field                                              */
    /*  u32HeaderSel =  UART_LIN_CTL_LIN_HEAD_SEL_BREAK                    */
    /*                  UART_LIN_CTL_LIN_HEAD_SEL_BREAK_SYNC               */
    /*                  UART_LIN_CTL_LIN_HEAD_SEL_BREAK_SYNC_ID            */
    /*---------------------------------------------------------------------*/
    void LIN_SendHeaderUsingLinCtlReg(uint32_t u32id, uint32_t u32HeaderSel)
    {
        g_i32pointer = 0 ;

        /* Switch back to LIN Function */
        UART1->FUN_SEL = UART_FUNC_SEL_LIN;

        /* Set LIN 1. PID as 0x30 [UART_LIN_CTL_LIN_LIN_PID(0x30)]
                   2. Header select as includes "break field", "sync field" and "frame ID
                      field".[UART_LIN_CTL_LIN_HEAD_SEL_BREAK_SYNC_ID]
                   3. Break/Sync Delimiter Length as 1 bit time
                      [UART_LIN_CTL_LIN_BS_LEN(1)]
                   4. Break Field Length as 12 bit time [UART_LIN_CTL_LIN_BKFL(12)]
                   5. ID Parity Enable. Hardware will calculate and fill P0/P1
                      automatically  [UART_LIN_CTL_LIN_IDPEN_Msk]
        */
        if (u32HeaderSel == UART_LIN_CTL_LIN_HEAD_SEL_BREAK_SYNC_ID)
        {
            UART1->LIN_CTL = UART_LIN_CTL_LIN_LIN_PID(u32id) |
UART_LIN_CTL_LIN_HEAD_SEL_BREAK_SYNC_ID |
                             UART_LIN_CTL_LIN_BS_LEN(1) | UART_LIN_CTL_LIN_BKFL(12) |
UART_LIN_CTL_LIN_IDPEN_Msk;
            /* LIN TX Send Header Enable */
            UART1->LIN_CTL |= UART_LIN_CTL_LIN_SHD_Msk;

            /* Wait until break field, sync field and ID field transfer completed */
            while ((UART1->LIN_CTL & UART_LIN_CTL_LIN_SHD_Msk) ==
UART_LIN_CTL_LIN_SHD_Msk);
```

```
        }
        /* Set LIN 1. Header select as includes "break field" and "sync
                      field".[UART_LIN_CTL_LIN_HEAD_SEL_BREAK_SYNC]
                   2. Break/Sync Delimiter Length as 1 bit time
                      [UART_LIN_CTL_LIN_BS_LEN(1)]
                   3. Break Field Length as 12 bit time [UART_LIN_CTL_LIN_BKFL(12)]
        */
        else if (u32HeaderSel == UART_LIN_CTL_LIN_HEAD_SEL_BREAK_SYNC)
        {
            UART1->LIN_CTL = UART_LIN_CTL_LIN_HEAD_SEL_BREAK_SYNC |
UART_LIN_CTL_LIN_BS_LEN(1) | UART_LIN_CTL_LIN_BKFL(12);
            /* LIN TX Send Header Enable */
            UART1->LIN_CTL |= UART_LIN_CTL_LIN_SHD_Msk;

            /* Wait until break field and sync field transfer completed */
            while ((UART1->LIN_CTL & UART_LIN_CTL_LIN_SHD_Msk) ==
UART_LIN_CTL_LIN_SHD_Msk);

            /* Send ID field, g_u8SendData[0] is ID+parity field*/
            g_u8SendData[g_i32pointer++] = GetParityValue(u32id);   // ID+Parity Field
            UART_Write(UART1, g_u8SendData, 1);
        }
        /* Set LIN 1. Header select as includes "break field"
                      .[UART_LIN_CTL_LIN_HEAD_SEL_BREAK]
                   2. Break/Sync Delimiter Length as 1 bit time
                      [UART_LIN_CTL_LIN_BS_LEN(1)]
                   3. Break Field Length as 12 bit time [UART_LIN_CTL_LIN_BKFL(12)]
        */
        else if (u32HeaderSel == UART_LIN_CTL_LIN_HEAD_SEL_BREAK)
        {
            UART1->LIN_CTL = UART_LIN_CTL_LIN_HEAD_SEL_BREAK | UART_LIN_CTL_LIN_BS_LEN(1) |
UART_LIN_CTL_LIN_BKFL(12);
            /* LIN TX Send Header Enable */
            UART1->LIN_CTL |= UART_LIN_CTL_LIN_SHD_Msk;

            /* Wait until break field transfer completed */
            while ((UART1->LIN_CTL & UART_LIN_CTL_LIN_SHD_Msk) ==
UART_LIN_CTL_LIN_SHD_Msk);

            /* Send sync field and ID field*/
            g_u8SendData[g_i32pointer++] = 0x55 ;                  // SYNC Field
            g_u8SendData[g_i32pointer++] = GetParityValue(u32id);   // ID+Parity Field
            UART_Write(UART1, g_u8SendData, 2);
```

```
        }
    }
```

2.1.3 Data Receive through LIN Slave

```
/*---------------------------------------------------------------------*/
/* MAIN function                                                       */
/*---------------------------------------------------------------------*/
int main(void)
{
    …
    while (1)
    {
        if (!(ReceIndex < 10))
        {
            //reload default Baud rate for next Sync!
            UART1->LIN_CTL |= UART_LIN_CTL_LINS_DUM_EN_Msk;
            printf("\nGetData:");

            for (i = 0; i < 10; i++)
            {
                printf("0x%x,", g_u8ReceiveData[i]);
                g_u8ReceiveData[i] = 0;
            }
            ReceIndex = HeadFlag = 0;
        }
    }    …
}
/*---------------------------------------------------------------------*/
/* UART1 interrupt subroutine                                          */
/*---------------------------------------------------------------------*/
void UART1_IRQHandler(void)
{
    uint32_t u32IntSts = UART1->ISR;
    uint32_t u32LinSts = UART1->LIN_SR;
    uint8_t Temp;

    UART1->ISR = u32IntSts;

    if (u32IntSts & UART_ISR_LIN_IF_Msk)
    {
        if (u32LinSts & UART_LIN_SR_BIT_ERR_F_Msk)
```

```
            {
                UART1->LIN_SR |= UART_LIN_SR_BIT_ERR_F_Msk;
                printf("\n bit error\n");
            }

            if (u32LinSts & UART_LIN_SR_LINS_BKDET_F_Msk)
            {
                UART1->LIN_SR |= UART_LIN_SR_LINS_BKDET_F_Msk;
                printf("\n break detected \n");
            }

            if (u32LinSts & UART_LIN_SR_LINS_SYNC_F_Msk)
            {
                printf("sync\n");
            }

            if (u32LinSts & UART_LIN_SR_LINS_IDPERR_F_Msk)
            {
                UART1->LIN_SR |= UART_LIN_SR_LINS_IDPERR_F_Msk;
                printf("\n slave ID parity error\n");
            }

            if (u32LinSts & UART_LIN_SR_LINS_HERR_F_Msk)
            {
                UART1->LIN_SR |= UART_LIN_SR_LINS_HERR_F_Msk;
                printf("\n slave header error \n");
            }

            if (u32LinSts & UART_LIN_SR_LINS_HDET_F_Msk) //received header
            {
                UART1->LIN_SR |= UART_LIN_SR_LINS_HDET_F_Msk;
                ReceIndex = 0;
                HeadFlag = 1;
            }
        }

        if (u32IntSts & UART_ISR_RDA_IF_Msk)    //received data
        {
            Temp = UART1->DATA;
            if (HeadFlag == 1)
            {
```

```
                if (ReceIndex < 10)
                    g_u8ReceiveData[ReceIndex++] = Temp;
            }
        }
    }
```

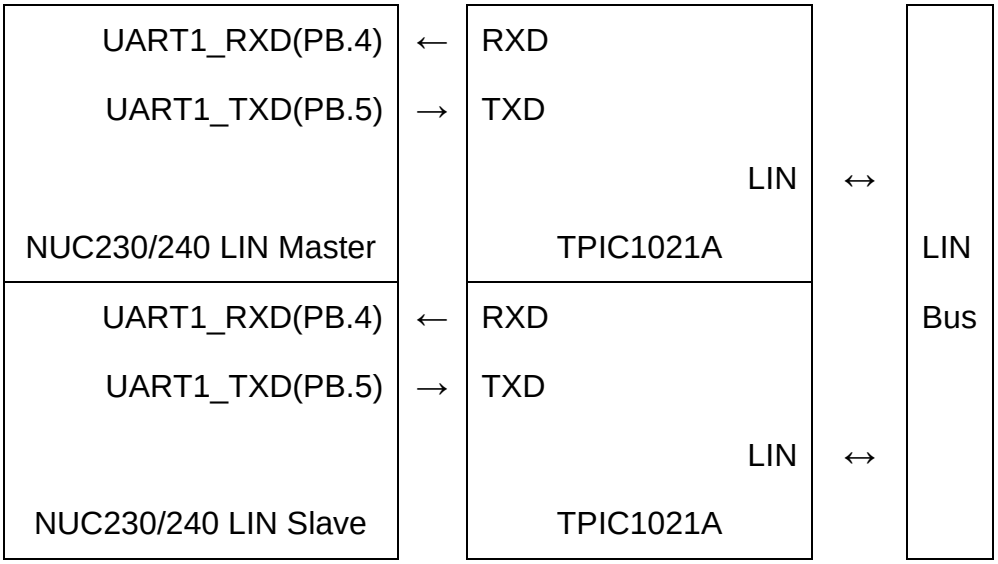
3 Software and Hardware Environment

Software Environment

- BSP version
  - NUC230_240 Series BSP CMSIS v3.01.001
- IDE version
  - Keil uVersion 5.26

Hardware Environment

- Circuit components
  - NuTiny-EVB-NUC240
- Diagram



4 Directory Information

- EC_NUC230_LIN_Bus_Data_Transfer_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 — Source file of example code

5 How to Execute Example Code

1. Browsing into sample code folder by Directory Information (section 4) and double click UART_LIN_Master.uvproj and UART_LIN_Slave.uvproj.
2. Enter Keil compile mode
   a. Build
   b. Download
   c. Start/Stop debug session
3. Enter debug mode
   a. Run

6 Revision History

Date	Revision	Description
Nov. 28, 2019	1.00	1. Initially issued.

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.