Application Note

LIN Basics and Implementation on MSPM0

Zoey Wei

ABSTRACT

This application note focuses on the implementation of basic function of LIN using MSPM0. Specifically, how MSPM0 works with LIN driver to implement protocol layer and physical layer functions is going to be presented, which can help develop the software project quickly. Also note the MSPM0 software level only supports simple LIN communication.

Table of Contents

Trademarks

1 Introduction

LIN (Local Interconnect Network) bus is a low-cost serial communication protocol based on UART/SCI (Universal Asynchronous Receiver-Transmitter/Serial Communication Interface). Due to the low cost, it is widely used in Auto area as the subline of CAN. As shown in Figure 1-1, LIN communication follows a single commander and multiple responders, and the MCU uses the UART interface combined with LIN transceiver to communicate between nodes.

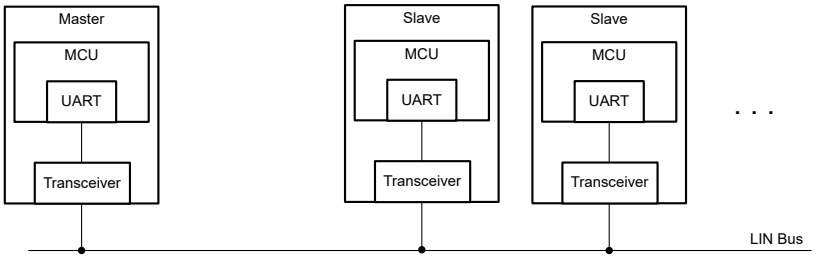

Figure 1-1. LIN Network

Similar to most network protocols, LIN is defined as a multi-layered system officially, which varying from the physical interface to the application, as shown in Figure 1-2. The node application layer transmits signals and messages to the lower layer, and encapsulates them into a frame format through the protocol layer, and transmits them to other nodes through the LIN bus.

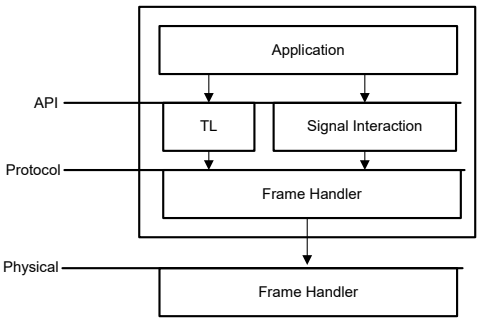

Figure 1-2. Composition of the LIN Node

This application note focuses on the implementation of basic functions of LIN using MSPM0. Specifically, how MSPM0 works with LIN driver to implement protocol layer and physical layer functions is presented, which can help develop the software project quickly.

2 How MSPM0 Support LIN Function

2.1 Clock

The LIN specification regulates the clock accuracy of the commander-responder node, which is reflected in the bit rate specification, as shown in Table 2-1.

Table 2-1. LIN Bit Rate Request

Bit Rate Tolerance	Name	ΔF / F_{Nom} [1]
Commander node (deviation from nominal bit rate)	$F_{TOL\_RES\_MASTER}$	<±0.5%
Responder node without making use of synchronization (deviation from nominal bit rate)	$F_{TOL\_RES\_SLAVE}$	<±1.5%
Deviation of responder node bit rate from the nominal bit rate before synchronization; relevant for nodes making use of synchronization and direct break detection.	F_{tol_unsync}	±14%
Deviation of responder node bit rate relative to the commander node bit rate after synchronization	F_{TOL_SYNC}	<±2%
For communication between any two nodes (for example, data stream from one responder to another responder), the bit rate must not differ by more than $F_{TOL\_SL\_to\_SL}$	$F_{TOL\_SL\_to\_SL}$	<±2%

(1) The specific bit rate used on a LIN bus is defined as the nominal bit rate F_{Nom}

Table 2-2 shows the clock specs of MSPM0.

Table 2-2. Clock Specs of MSPM0(T=25℃)

Series	Clock	Support External Crystal Oscillators
MSPM0G	Internal 4 - 32MHz oscillator with up to 0.7% accuracy (SYSOSC)[1]	External 4 - 48MHz crystal oscillator(HFXT)
	Internal 32kHz low-frequency oscillator (LFOSC) with ±3% accuracy	External 32kHz crystal oscillator(LFXT)
MSPM0L	Internal 4 - 32MHz oscillator with ±0.7% accuracy (SYSOSC)	Not support
	Internal 32kHz low-frequency oscillator with ±3% accuracy (LFOSC)	
MSPM0C	Internal 24MHz oscillator with up to ±1% accuracy (SYSOSC)	Support(only support for 20pins)
	Internal 32kHz low-frequency oscillator (LFOSC) with ±3% accuracy	

(1) MSPM0G series support PLL up to 80MHz

2.2 LIN Hardware

For supporting local interconnect network (LIN) protocol, the following hardware enhancements are implemented in the UART0 module:

- 16 bit up-counter (LINCNT) clocked by the UART clock.
- Interrupt capability on counter overflow (CPU_INT.IMASK.LINOVF).
- 16 bit capture register (LINC0) with two configurable modes
  - Capture of LINCNT value on RXD falling edge. Interrupt capability on capture.
  - Compare of LINCNT with interrupt capability on match.
- 16 bit capture register (LINC1) can be configured
  - Capture LINCNT value on RXD rising edge. Interrupt capability on capture.

Besides, MSPM0 also has large register to support LIN communication. For example, when served as commander, there is a LCRH.BRK register to enable UART.TXD to send continually low level. And for responder, LINCNT register can help get the time of the break field. For more detailed information, see *MSPM0 G-Series 80MHz Microcontrollers,* technical reference manual.

2.3 LIN Demo Code in SDK

To help develop LIN communication fast and conveniently, TI has provided demo codes of responder and commander in *SDK*. You can download MSPM0 all code example from *MSPM0-SDK Software Development Kit*. This demo code is configure UART as LIN commander or responder, and demonstrates basic transmit and receive of LIN 2.0 packets using enhanced checksum. Some key feature:

- Baud rate: 19200bps
- Choose interrupt or polling mode through predefined Symbols while transmitting data
- Timeout function while receiving data

2.3.1 LIN Commander

The main function of LIN commander demo code is to send different command in LIN protocol frame. As the two bottoms are pressed, PID 0x39 and 0x08 are sent desperately. One is to light LED of responder and the other one is to receive data from responder.

Figure 2-1 shows the flow process of LIN commander demo code.

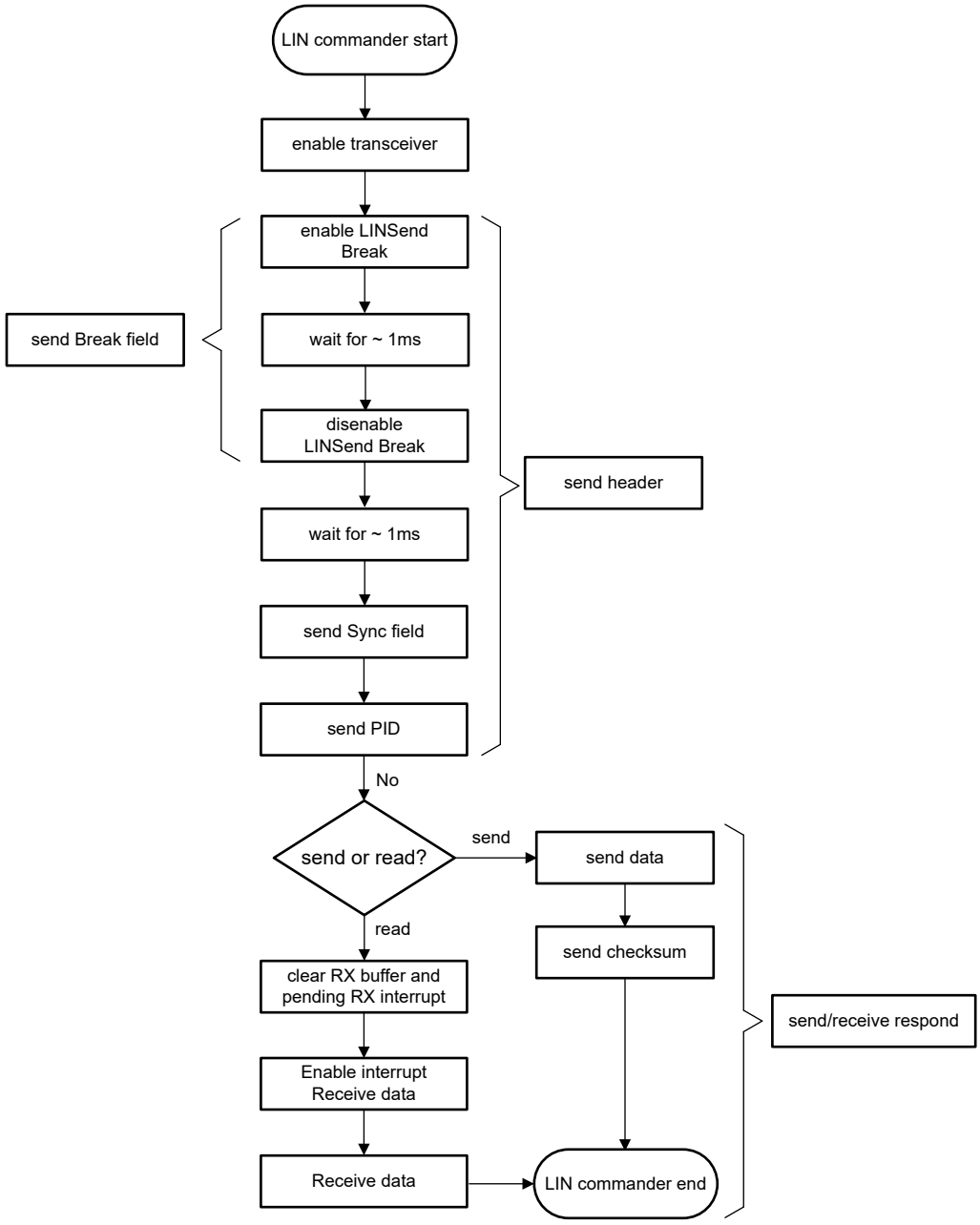

Figure 2-1. Flow Process of LIN Commander Demo Code

To initialize the hardware, TI System Configuration Tool (SysConfig) is used to generate configuration code of UART, such as UART clock, pin configuration and so on. In this demo code, we choose 19200 Baud Rate, following the LIN specs regulation.

Table 2-3 shows a summary for the LIN commander project, which includes the main used definitions and functions.

Table 2-3. LIN Commander Project Main Content

Name	Task	Description	Location
LIN_TABLE_INDEX_PID_xx	definition	The PID of each frame	lin_command.c
LIN_MESSAGE_NOT_FOUND	definition	0xFF. UART LIN value for the message not found	lin_config.h
LIN_SYNC_BYTE	definition	0x55. UART LIN value for the sync byte	lin_config.h
LIN_BREAK_LENGTH	definition	0x08. UART LIN break length to 1ms	lin_config.h
SYSCFG_DL_init()	function	Initialize the peripherals, generated by Sysconfig	ti_msp_dl_config.c
DL_UART_enableLINSendBreak(UART_Regs *uart)	function	Enable send break. when enabled, a low level is continually output on the TXD signal after completing transmission of the current character	dl_uart.c
DL_UART_disableLINSendBreak(UART_Regs *uart)	function	Disable send break	dl_uart.c
LIN_Commander_transmitMessage(UART_Regs *uart, uint8_t tableIndex, uint8_t *buffer, LIN_table_record_t *messageTable)	function	LIN transmits message	lin_config.c
LIN_processMessage_Rx()	function	Process as message received	Lin_commander.c

2.3.2 LIN Responder

The main function of LIN commander demo code is to receive command from commander and execute the corresponding instructions. In this demo code, responder does not clock sync up but uses self-clock and only checks whether the sync byte is correct. Also, there is no timeout error detection.

Figure 2-2 shows the flow process of LIN responder demo code.

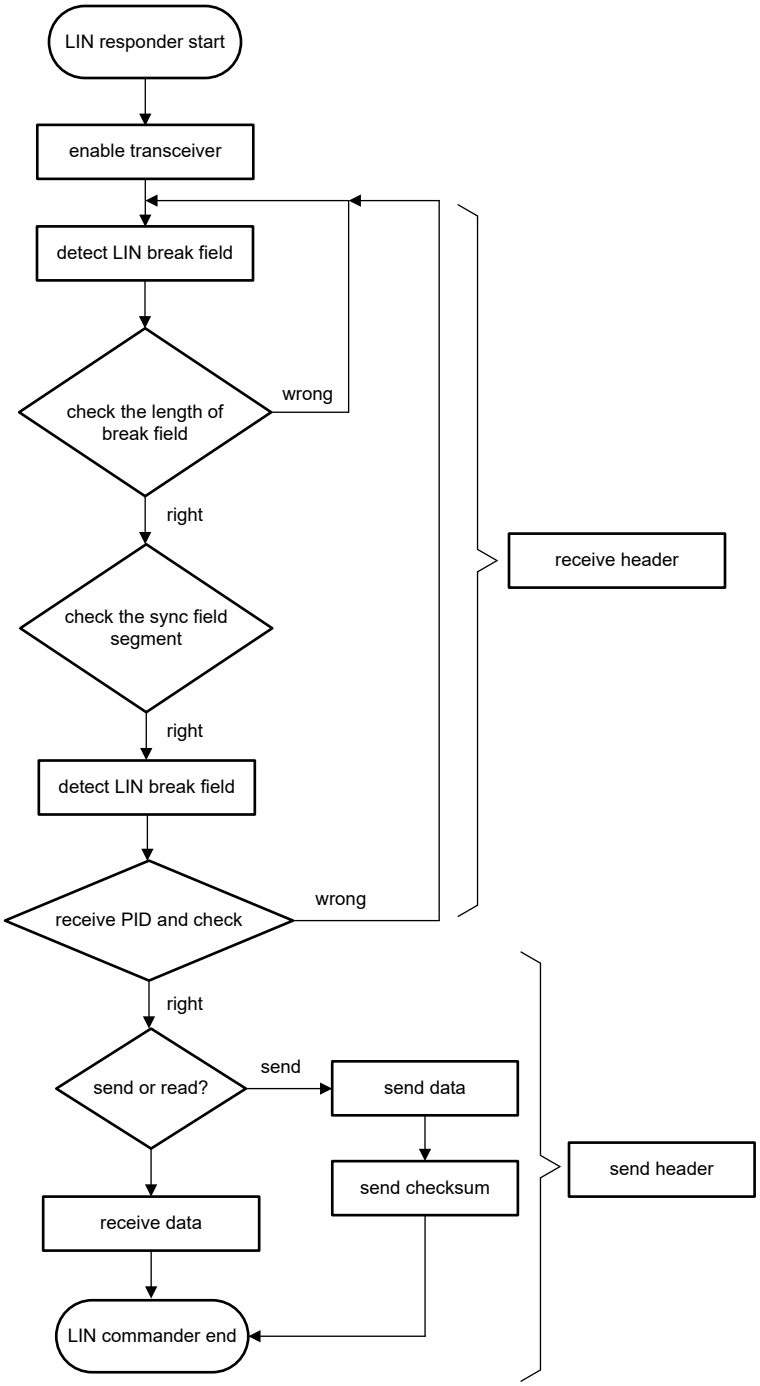

Figure 2-2. LIN Responder Demo Code Flow Process

In this demo code, a state machine is used to help receive command and give response, as shown in Figure 2-3. When an interrupt occurs, there are state flags to decide what is the next non-executed state and jump into.

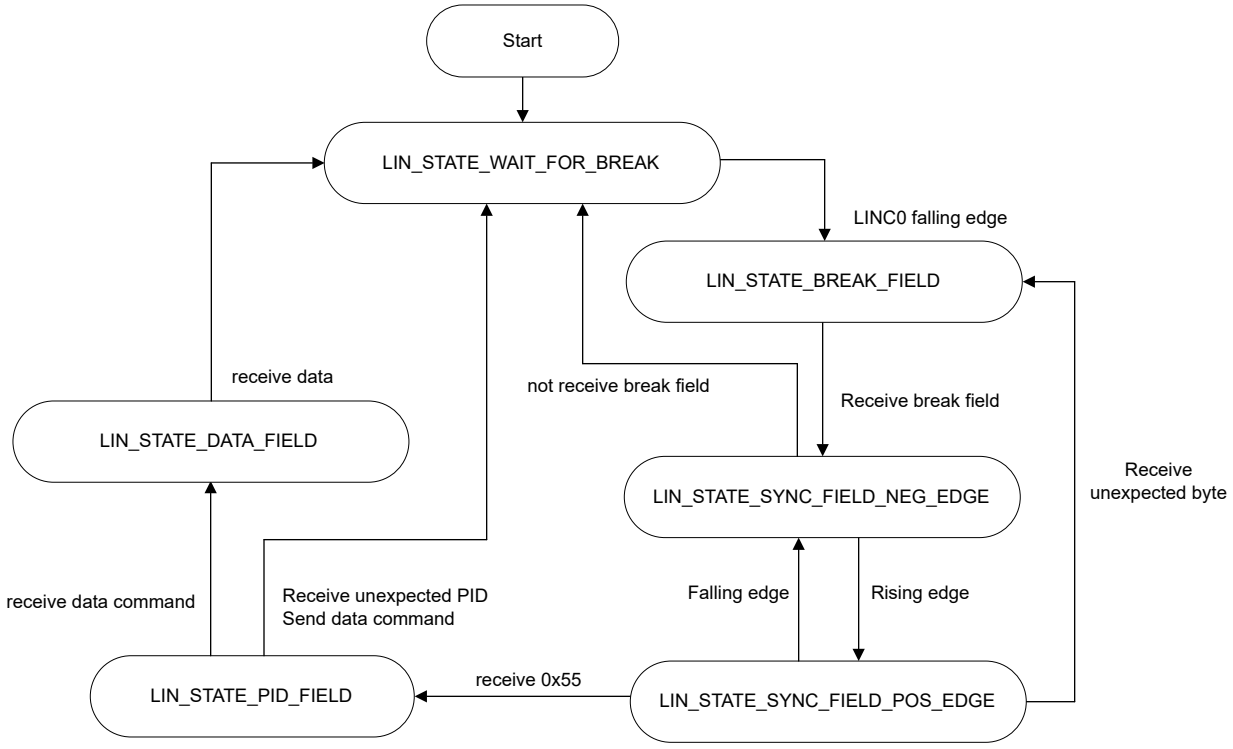

Figure 2-3. LIN Responder Demo Code State Machine

In this demo code, LIN hardware is initialized through Sysconfig. Besides pin configuration, interrupt and basic register needs to be set. LIN counter is used to detect and verify the break field. Also, Sync field is realized by the count and interrupt function of LINC0,LINC1. For detailed seeting, please refer to the .syscfg file in code example.

Table 2-4 is a summary for the LIN responder project, which includes the main used definitions and functions.

Table 2-4. LIN Responder Project Main Content

Name	Task	Description	Location
LIN_RESPONDER_SYNC_CYCLES	definition	5. The number of cycles in a sync validation	lin_config.h
LIN_MESSAGE_NOT_FOUND	definition	0xFF. UART LIN value for the message not found	lin_config.h
LIN_SYNC_BYTE	definition	0x55. UART LIN value for the sync byte	lin_config.h
LIN_RESPONSE_LAPSE	definition	Number of delay cycles between PID STOP bit and data transmission START bit	lin_config.h
SYSCFG_DL_init()	function	Initialize the peripherals, generated by Sysconfig	ti_msp_dl_config.c
DL_UART_Extend_getLINCounterValue(UART_Regs *uart)	function	Get the LIN counter value	dl_uart.c
DL_UART_Extend_getLINRisingEdgeCaptureValue(UART_Regs *uart)	function	Get the LINC1 counter value	dl_uart.c
DL_UART_Extend_getLINFallingEdgeCaptureValue(UART_Regs *uart)	function	Get the LINC0 counter value	dl_uart.c
DL_UART_Extend_receiveData(UART_Regs *uart)	function	Reads data from the RX FIFO	dl_uart.c
setLINResponderRXMessage(UART_Regs *uart, uint8_t data, volatile LIN_STATE *gStateMachine)	function	LIN give corresponding action	lin_responder.c

3 LIN Communication Realization

3.1 Hardware Connection

LIN transceiver is needed to cooperate with MCU, as the LIN transceiver can transform UART signal to LIN signal.

Here we take LP-MSPM0G3057 and TLIN2029EVM for example. The hardware connection is shown as Figure 3-1. The two of the connections form a communication unit, and can be connected to another unit like MCU with transceiver, LIN Analysis to PC.

1. Connect MSPM0G3507 UART pin to TLIN2029.
2. Connect MSPM0G3507 Power pin(3.3V and GND) to TLIN2029.
3. Connect MSPM0G3507 PB15 which served as enable pin to TLIN2029 ENABLE pin.
4. Use external 12V power supply to TLIN VBAT pin. If the other connected communication unit like LIN Analysis has already connect 12V, then this step can be ignored.
5. Connect VBAT,LIN and GND pin in TLIN2029 to responder or commander hardware.
6. Power supply MCU.

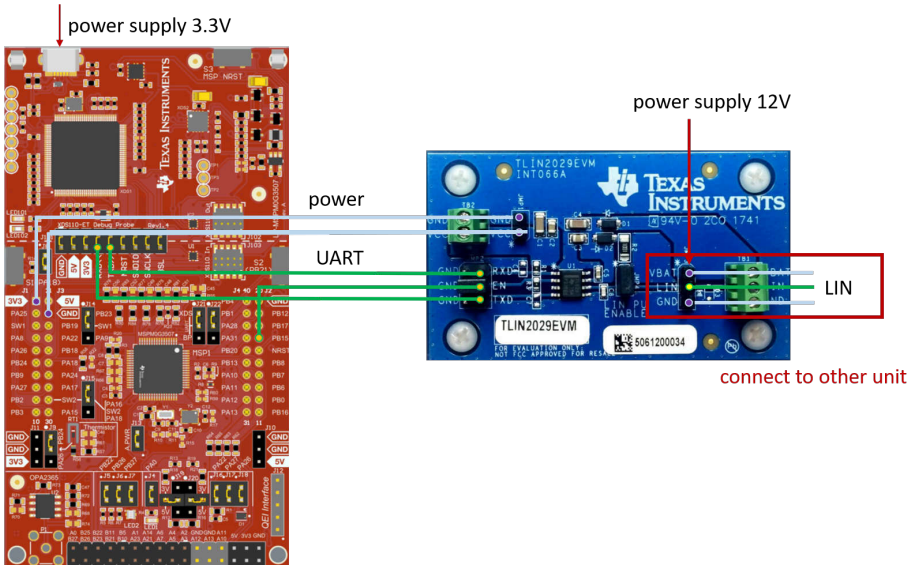

Figure 3-1. Hardware Connection Between MSPM0 and LIN Transceiver

3.2 Test Results

Next, test MSPM0 as commander and responder respectively, and CAN&LIN Analyzer can be used to communicate with MSPM0 through LIN.

3.2.1 Commander

In this case, MCU serves as commander, and PC with CAN/LIN analysis serves as responder. Baud rate is 19200.

When selecting the Button1 to make MCU send 0x39(PID), the results are shown in Figure 3-2. As shown from Figure 3-2, the host computer can successfully receive the data sent by the MCU, which can also be confirmed from the waveform diagram (Figure 3-3).

ID[PID]	DATA(Hex)	Check(Hex)	Check Type	Data Type	Timestamp	Channel
39 [39]	02 02 03 04 05 06 07 09	A0	Enhanced C…	Slave Read	0.000	LIN1
39 [39]	03 02 03 04 05 06 07 0A	9E	Enhanced C…	Slave Read	0.899	LIN1
39 [39]	04 02 03 04 05 06 07 0B	9C	Enhanced C…	Slave Read	2.100	LIN1
39 [39]	05 02 03 04 05 06 07 0C	9A	Enhanced C…	Slave Read	2.532	LIN1
39 [39]	06 02 03 04 05 06 07 0D	98	Enhanced C…	Slave Read	2.933	LIN1

Figure 3-2. MCU as Commander Transmitting Data Results

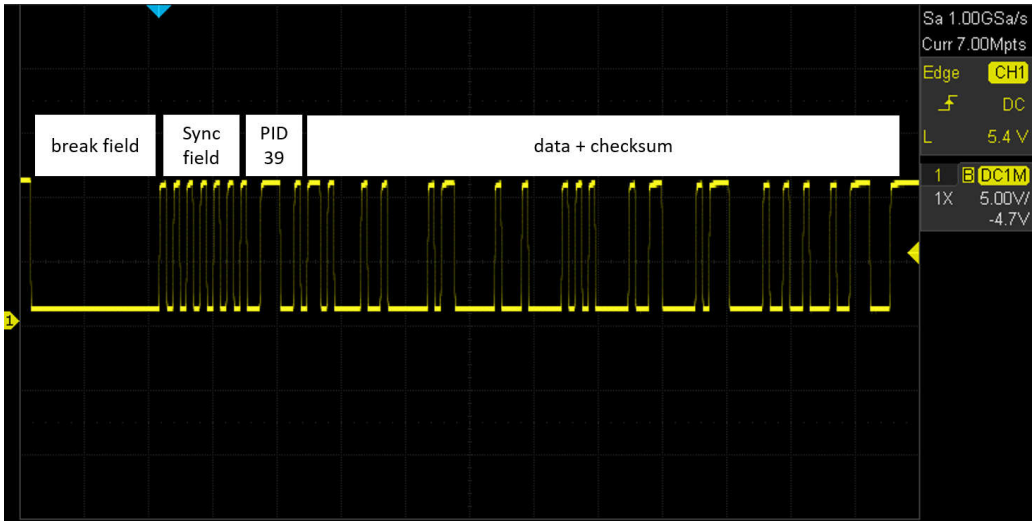

Figure 3-3. MCU as Commander Transmitting Data Waveform

When press the button2 to make MCU send 0x08(PID), the responder can send data to MCU. As shown from Figure 3-4, the responder transmit 0x11,0x22,0x33 and 0x44 and MCU succeeds to receive this data. But in this case, the check mode in PC is normal, so the checksum can not match the MCU and then the callback function can not work. However, the receive data can still be stored in the array.

ID[PID]	DATA(HEX)	Check(Hex)	Check Type	Data Type	Timestamp	Channel	Event
08 [08]	00 11 22 33 44	4D	Enhanced C…	Slave Write	0.000	LIN1	
08 [08]	00 11 22 33 44	4D	Enhanced C…	Slave Write	0.596	LIN1	
08 [08]	00 11 22 33 44	4D	Enhanced C…	Slave Write	1.156	LIN1	
08 [08]	00 11 22 33 44	4D	Enhanced C…	Slave Write	1.777	LIN1	

Set slave response data

Device Channel: UTA0503-[5300238F][LIN1]-Slave

Select	Data Type	Check Type	ID(Hex)	Data(Hex)	Response Mode
1	Slave Read	Standard Check	00		Loop Response
2	Slave Read	Standard Check	01		Loop Response
3	Slave Read	Standard Check	02		Loop Response
4	Slave Read	Standard Check	03		Loop Response
5	Slave Read	Standard Check	04		Loop Response
6	Slave Read	Standard Check	05		Loop Response
7	Slave Read	Standard Check	06		Loop Response
8	Slave Read	Standard Check	07		Loop Response
9	Slave Write	Enhanced Check	08	00 11 22 33 44	Loop Response
10	Slave Read	Standard Check	09		Loop Response
11	Slave Read	Standard Check	0A		Loop Response
12	Slave Read	Standard Check	0B		Loop Response
13	Slave Read	Standard Check	0C		Loop Response
14	Slave Read	Standard Check	0D		Loop Response
15	Slave Read	Standard Check	0E		Loop Response
16	Slave Read	Standard Check	0F		Loop Response
17	Slave Read	Standard Check	10		Loop Response

Select All | Invert Select | Set data

Figure 3-4. MCU as Commander Receiving Data Results

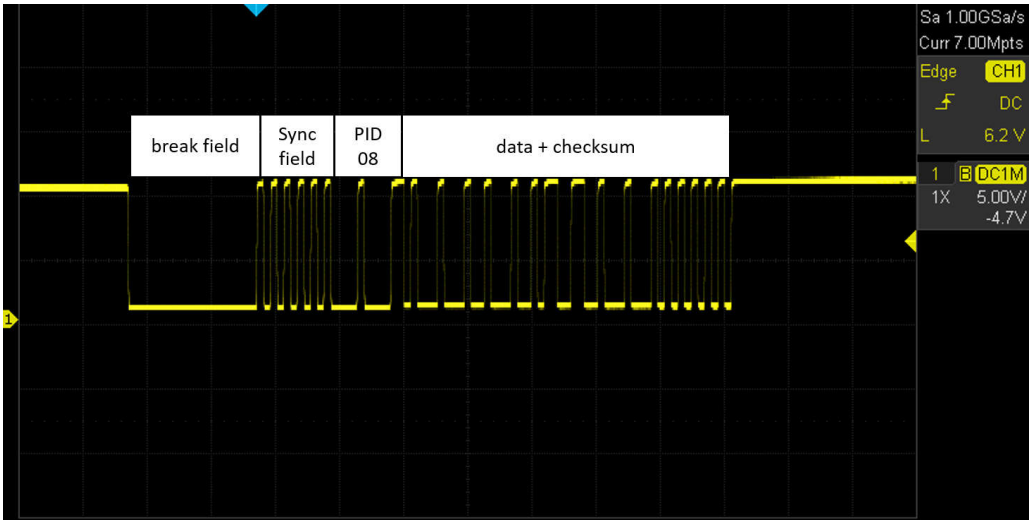

Figure 3-5. MCU as Commander Receiving Data Waveform

gCommanderRXBuffer	unsigned char[8]	[0x11 '\x11',0x22 '"',0x33 '3',0x44 'D',0x55 'U'...]...	0x20200038
[0]	unsigned char	0x11 '\x11' (Hex)	0x20200038
[1]	unsigned char	0x22 '"' (Hex)	0x20200039
[2]	unsigned char	0x33 '3' (Hex)	0x2020003A
[3]	unsigned char	0x44 'D' (Hex)	0x2020003B
[4]	unsigned char	0x55 'U' (Hex)	0x2020003C
[5]	unsigned char	0x00 '\x00' (Hex)	0x2020003D
[6]	unsigned char	0x00 '\x00' (Hex)	0x2020003E

Figure 3-6. gCommanderRXBuffer Results

3.2.2 Responder

In this case, MSPM0 serves as responder. The demo code realizes the function that if received 0x39/0xBA/ 0xFB, MCU can receive data from commander. And if received 0x08/0x49/0x0D PID, MCU can transmit data to the commander.

When commander send 0x3B(PID is 0xFB), the MCU is going to receive data from host. As shown in Figure 3-7, Figure 3-8, and Figure 3-9, the host computer successfully transmit the data which can be read from MCU RAM in debug mode.

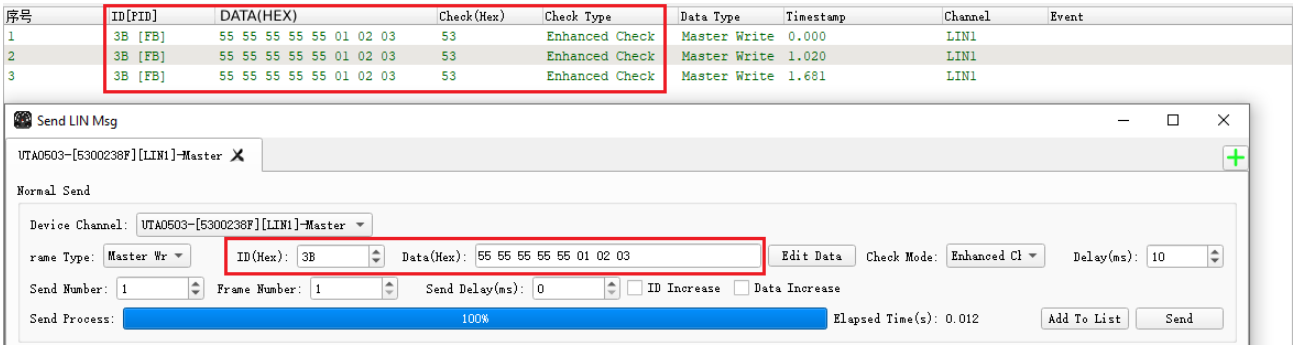

Figure 3-7. MCU as Responder Receiving Data Results

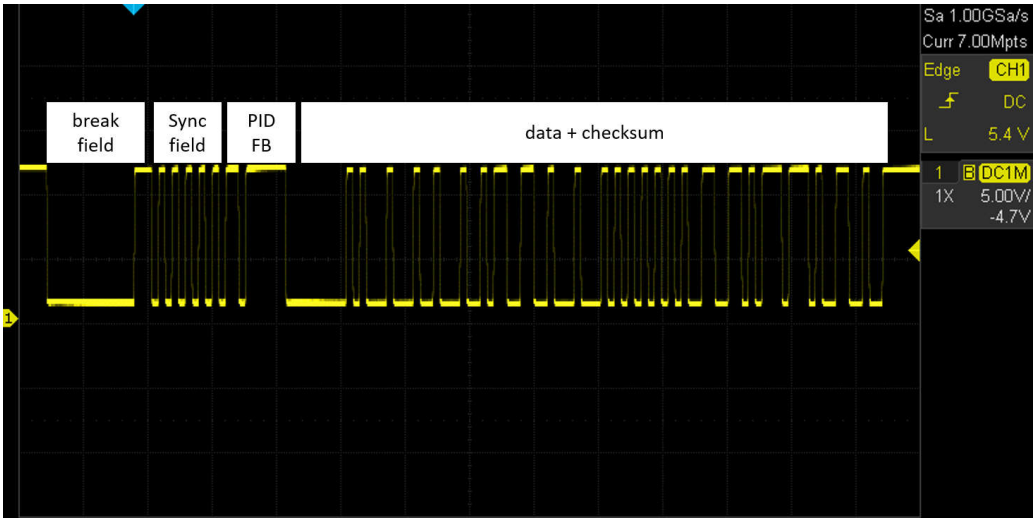

Figure 3-8. MCU as Responder Receiving Data Waveform

gResponderRXBuffer	unsigned char[8]	[0x55 'U',0x55 'U',0x55 'U',0x55 'U',0x55 'U'...] (...
[0]	unsigned char	0x55 'U' (Hex)
[1]	unsigned char	0x55 'U' (Hex)
[2]	unsigned char	0x55 'U' (Hex)
[3]	unsigned char	0x55 'U' (Hex)
[4]	unsigned char	0x55 'U' (Hex)
[5]	unsigned char	0x01 '\x01' (Hex)
[6]	unsigned char	0x02 '\x02' (Hex)
[7]	unsigned char	0x03 '\x03' (Hex)

Figure 3-9. gResponderRXBuffer Results

When commander send 0x08, the result is shown in Figure 3-10 and Figure 3-11. The host is set to read mode, and select Enhanced check mode. Then, the communication is successful, and the host computer successfully receives the data sent by the MCU, which can be confirmed through waveform. Finally the GPIO is toggled to show the end of communication.

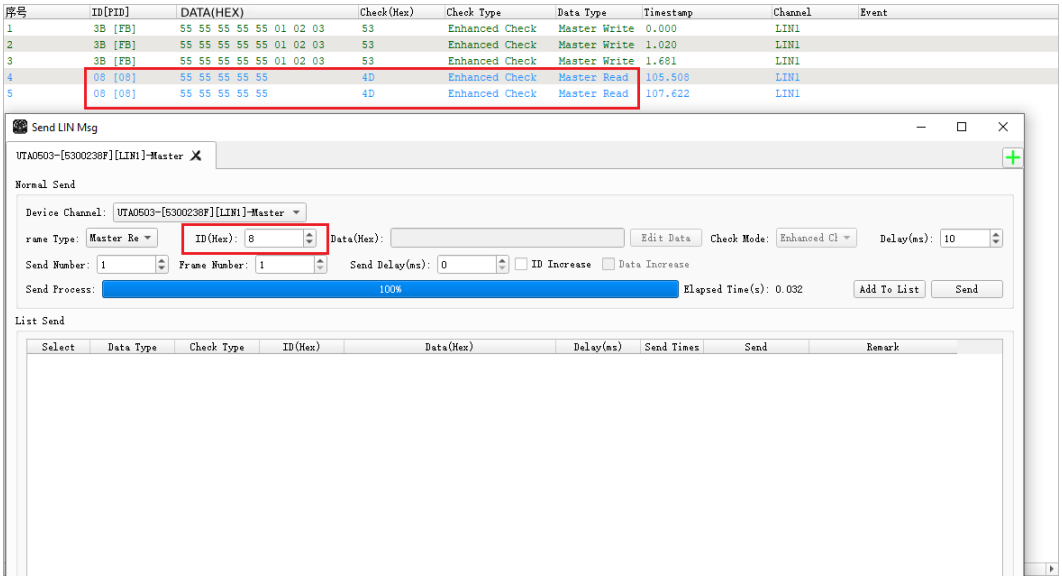

Figure 3-10. MCU as Responder Transmitting Data Results

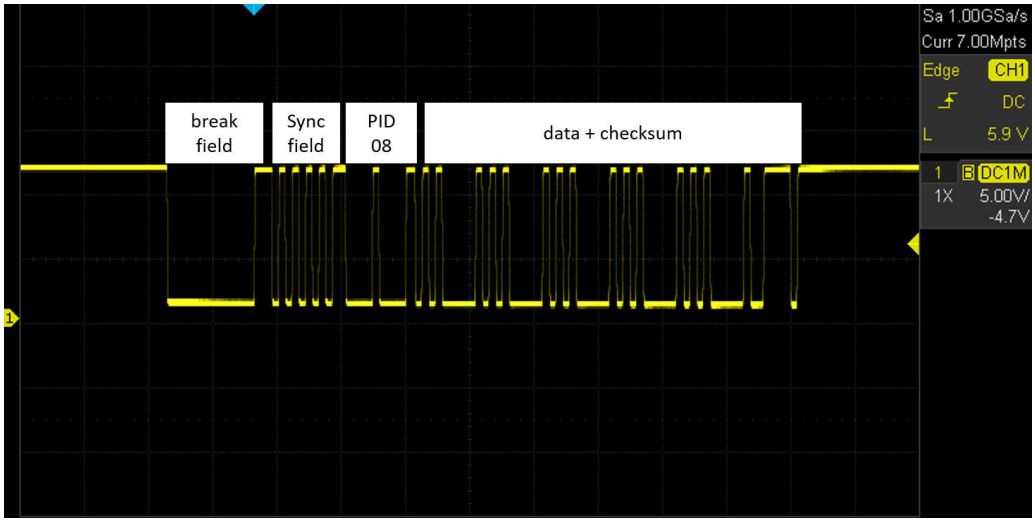

Figure 3-11. MCU as Responder Transmitting Data Waveform

4 Summary

In this document, a brief introduction to the LIN basic knowledge is covered and how MSPM0 support LIN communication with hardware and software. The document also provides a preliminary understanding of MSPM0 and LIN, and can help speed up the development progress.

5 References

- Texas Instruments, *MSPM0 C-series 24-MHz Microcontrollers,* technical reference manual.
- Texas Instruments, *MSPM0 L-series 24-MHz Microcontrollers,* technical reference manual.
- Texas Instruments, *MSPM0 G-series 24-MHz Microcontrollers,* technical reference manual.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265