AN12788

Emulating LIN Master/Slave with the FlexIO on i.MX RT Series MCU

Rev. 0 — March, 2020

Application Note

Contents

1 Introduction

This application note describes how to use FlexIO module to emulate LIN (Local Interconnect Network) bus based on i.MX RT series platform.

LIN is a widely used serial network protocol between components in vehicles. While the RT series MCU does not directly support LIN peripheral, the LPUART module of i.MX RT series can emulate the LIN bus. The FlexIO module can be a good workaround when LPUART is occupied.

FlexIO is an on-chip peripheral available on NXP i.MX RT series. It is a highly configurable module which is capable of emulating a wide range of communication protocols, such as UART, I2C, SPI, I2S, and so on. Users can also use FlexIO to emulate LIN bus.

This application creates a simple software demo based on the i.MX RT series platform for users to use FlexIO module emulating LIN Master&Slave with related configurations.

2 LIN overview

LIN is a low-cost serial communication protocol based on UART/SCI. LIN bus adopts the communication mode of one master and multiple slaves, and the transmission rate of LIN can reach up to 20 kbit/s. A LIN communication network can connect up to 16 nodes, 1 host node, and 1 to 15 slave nodes. The topology of LIN bus is linear which means all node devices are connected through single line. The following figure shows the LIN bus topology.

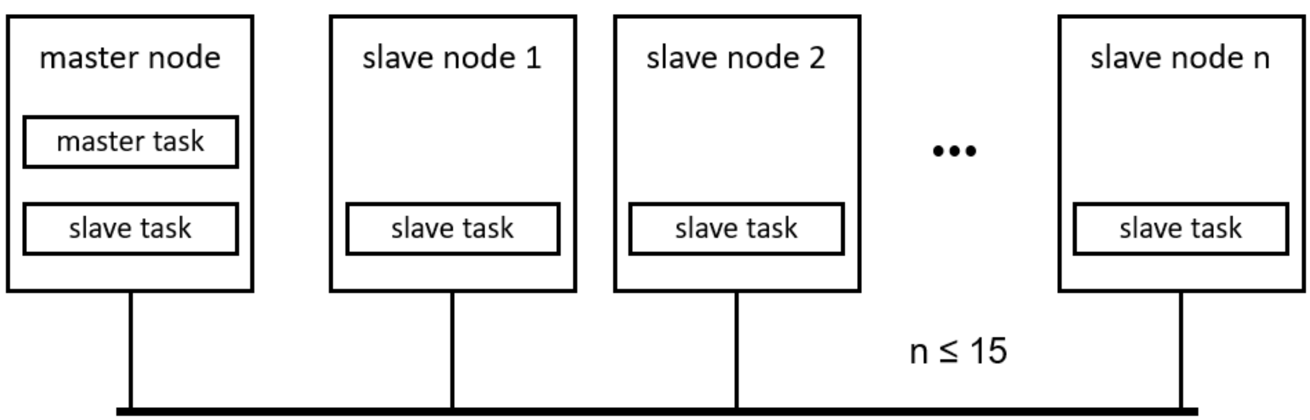

Figure 1. LIN bus topology

The host node contains the host task and the slave task, while the slave node contains only the slave task. The master task decides when and which frame shall be transferred on the bus. The slave tasks provide the data transported by each frame.

A frame of LIN consists of a header (provided by the master task) and a response (provided by a slave task). The host task sends the frame header, and the slave task determines whether to send or receive the response according to the frame header information. The following figure shows the data transmission process on the bus.

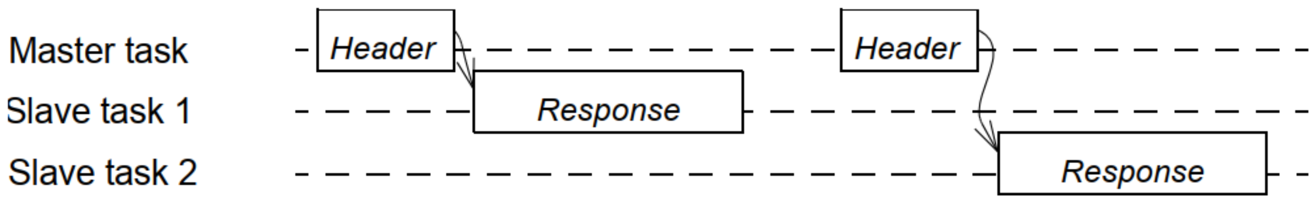

Figure 2. LIN bus Data transmission process

The header consists of a break field and sync field followed by a frame identifier. The frame identifier uniquely defines the purpose of the frame. The response consists of a data field and a checksum field. The following figure shows the structure of a complete data frame.

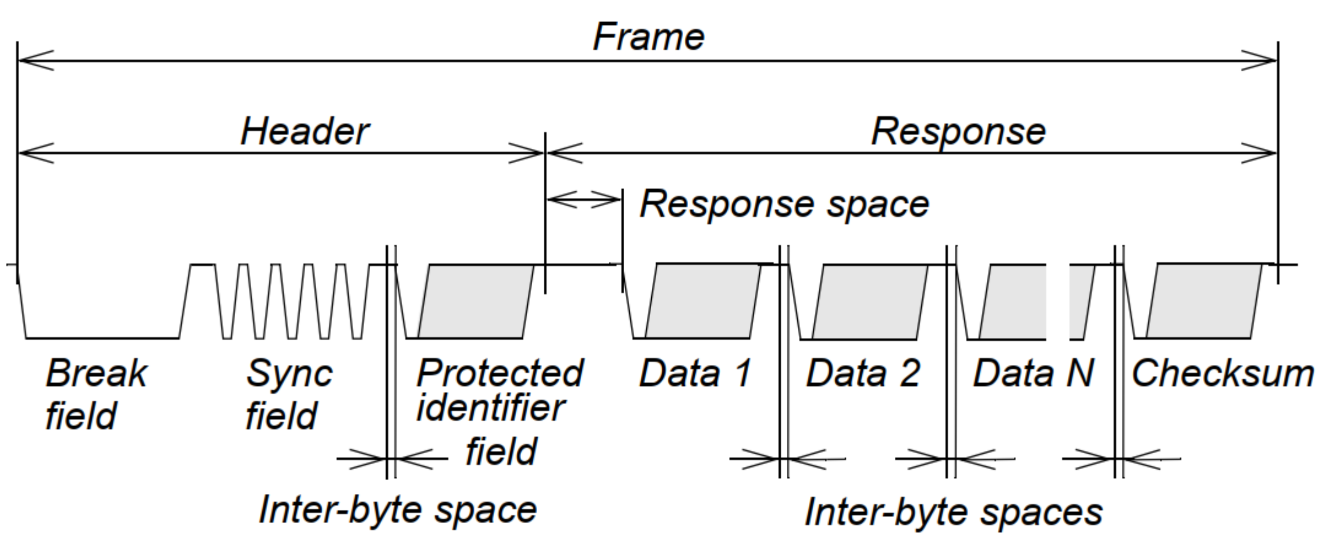

Figure 3. LIN frame structure

The break field is used to signal the beginning of a new frame and it shall be at least 13 nominal bit times of dominant value. The format of the rest of the data from the synchronous field to the checksum field is the same as that of the standard asynchronous serial port. For each byte of data, there is 1 low-level start bit(dominant bit), 1 high-level stop bit(recessive bit), and no checksum bit. A signal of LIN bus is transmitted with the LSB first and the MSB last. The sync field is a byte field with the data value 0x55.

3 Emulating LIN by using FlexIO

This chapter mainly introduces how to emulate LIN by using FlexIO module. LIN bus is a kind of low-cost serial communication system. So, the configuration of FlexIO used in emulating LIN is similar to that used in emulating UART.

3.1 LIN transmitter configuration

To emulate the LIN transmitter, users must use the following resources:

- One timer — configured as 8-bit baud counter mode to control the data shift.
- One shifter — controlled by Timer to shift data from SHIFTBUF.
- One pin — connect to the Shifter to output data.

Timer 0 is used to shift control the Shifter 0. The following figure shows the LIN transmitter block diagram.

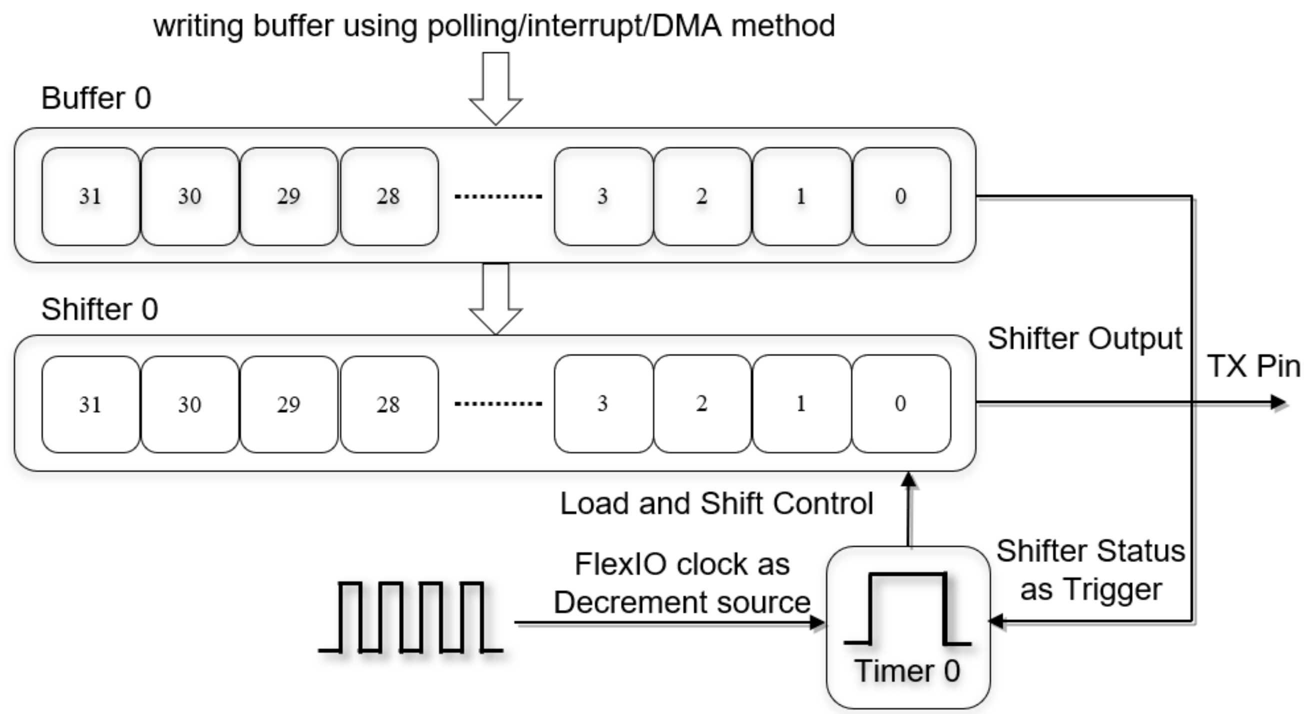

Figure 4. LIN transmitter block diagram

Limited by the protocol, the baud rate of this application demo is set as 19200. The following table shows the configuration for Timer 0.

Table 1. Configurations for Timer 0

Items	Configurations
Trigger Select	Shifter 0 status flag
Trigger Polarity	Active low
Trigger Source	Internal trigger
Pin Config	Output disable
Pin Select	N/A
Pin Polarity	N/A
Timer mode	Dual 8-bit counters baud mode
Timer Output	Timer output is logic one when enabled and is not affected by timer reset
Timer Decrement	Decrement counter on FlexIO clock, shift clock equals Timer output
Timer Reset	Timer never resets

Table continues on the next page...

Table 1. Configurations for Timer 0 (continued)

Timer Disable	Timer disabled on timer compare
Timer Enable	Timer enabled on trigger high
Timer Stop Bit	Enabled on timer disable
Timer Start Bit	Enable
Timer Compare	((bitCountPerChar[1] * 2 - 1) << 8) | (baudrate_divider[2]/ 2 - 1))

1. bitCountPerChar is the number of bits of each data.
2. baudrate_divider can be calculated from FlexIO clock divided by LIN baud rate.

In 8-bit Baud Counter Mode, the 16-bit counter is divided into two 8-bit counters. The lower 8-bits are used to configure the baud rate of the shift clock. The upper 8-bits are used to configure the number of shift clock edges in the transfer. When the lower 8-bits decrease to zero, the timer output is toggled and the lower 8-bits reload from the compare register. The upper 8-bits decrease when the lower 8-bits equal zero.

Table 2. Configurations for Shifter 0

Items	Configurations
Timer Select	Timer 0
Timer Polarity	Shift on positive edge of shift clock
Pin Config	Shifter pin output
Pin Select	FlexIO_D21
Pin Polarity	Active high
Shifter Mode	Transmit mode
Input Source	N/A
Shifter Stop Bit	Stop bit high
Shifter Start Bit	Start bit low

The shifter status flag is set when SHIFTBUF data has been transferred to the Shifter (SHIFTBUF is empty), and the status flag is cleared when the SHIFTBUF register is written. The shifter status flag of Shifter 0 is configured to be the trigger of the Timer 0. So, as soon as the SHIFTBUT is written, the status flag is cleared and Timer 0 is enabled. The FlexIO_D21 pin is configured as the data output pin. The Shifter 0 begins to shift out the data on the positive edge of the clock until the Timer 0 is disabled. Timer 0 is disabled when the timer counter counts down to 0.

3.2 LIN receiver configuration

To emulate the LIN receiver, use the following resources:

- One timer — configured as 8-bit baud counter mode to control the data shift.
- One shifter — controlled by timer to input data into SHIFTBUF.
- One pin — connect to the shifter to input data.

Timer 1 is used to shift control the Shifter 1. The following figure shows the LIN receiver block diagram.

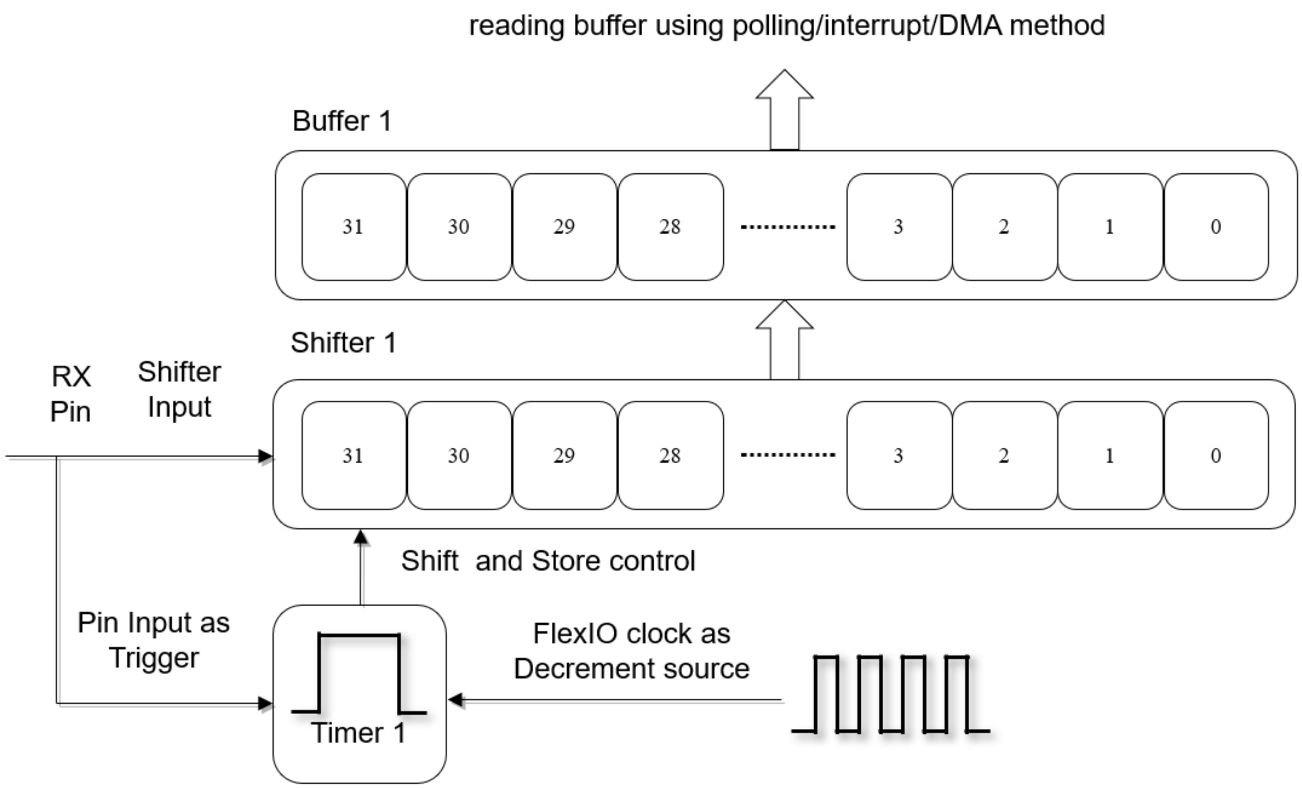

Figure 5. LIN receiver block diagram

Table 3. Configurations for Timer 1

Items	Configurations
Trigger Select	trigger from pin FlexIO_D26
Trigger Polarity	active high
Trigger Source	external trigger
Pin Config	output disable
Pin Select	N/A
Pin Polarity	N/A
Timer mode	dual 8-bit counters baud mode
Timer Output	Timer output is logic one when enabled and on timer reset
Timer Decrement	decrement counter on FlexIO clock, shift clock equals Timer output
Timer Reset	Timer reset on Timer Pin rising edge

Table continues on the next page...

Table 3. Configurations for Timer 1 (continued)

Timer Disable	Timer disabled on Timer compare
Timer Enable	Timer enabled on Pin rising edge
Timer Stop Bit	enabled on timer disable
Timer Start Bit	enable
Timer Compare	((bitCountPerChar * 2 - 1) << 8) | (baudrate_divider / 2 - 1))

The FlexIO_D26 pin's rising edge is configured to enable the Timer 1. The Shifter 1 begins to shift in the data on the negative edge of the clock until the timer is disabled. The timer is disabled when the timer counter counts down to 0. Table 4 shows the configuration for Shifter 1.

Table 4. Configurations for Shifter 1

Items	Configurations
Timer Select	Timer 1
Timer Polarity	Shift on negative edge of shift clock
Pin Config	Output disable
Pin Select	FlexIO_D26
Pin Polarity	Active high
Shifter Mode	Receive mode
Input Source	Input from pin
Shifter Stop Bit	Stop bit high
Shifter Start Bit	Start bit low

3.3 LIN task model

The configurations of transmitter and receiver of emulating LIN bus are the same as emulating UART, but these configurations cannot meet the requirement of LIN host task generating break field. This section introduces how to implement LIN protocol data transmission and reception.

3.3.1 Header of frame

The length of break field is at least 13 bits dominant value (low level). However, the standard format data includes a recessive bit (high level), low level exceeding 9 bits cannot be transmitted under the normal configuration. So when master task needs to initiate a data transfer, the configuration of FlexIO needs to be changed to implement the break field. Since the break field and the synchronous field are fixed values, and the total number of bits of these two fields is 24 bits, we can disable the start and stop bits of the first three bytes. The following figure shows the structure of these two fields.

	Break field														Synchronous field									
Bus Data	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1
End Conversion Data	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0	1	0
Send Data	0x00								0xA0								0xAA							

Figure 6. Structure of break and synchronous field

When master node send the Header, users can change the setting of Timer 0 and Shifter 0:

- Timer Stop Bit: Disable
- Timer Start Bit: Disable
- Shifter Stop Bit: Disable
- Shifter Stop Bit: Disable

And set the send size as 3 bytes, send data as {0x00, 0xA0, 0xAA}.

A slave node shall use a break detection threshold of 11 dominant local slave bit times. Get rid of the first bit as the start bit. Change the upper 8 bits of the Timer 1, which configure the number of bits in each word, to a 10-bit per word.

- Timer Compare[15:8]: 0x13

3.3.2 Master node state machine

In this application, to simulate the send and response status, we set a state machine to manage the multiple running states. This state machine contains the application layer and the driver layer. This application sets two enumeration variables to represent the two layers. The following figure shows the state machine of LIN master node. Blue boxes represent application layer state, green boxes represent driver layer state.

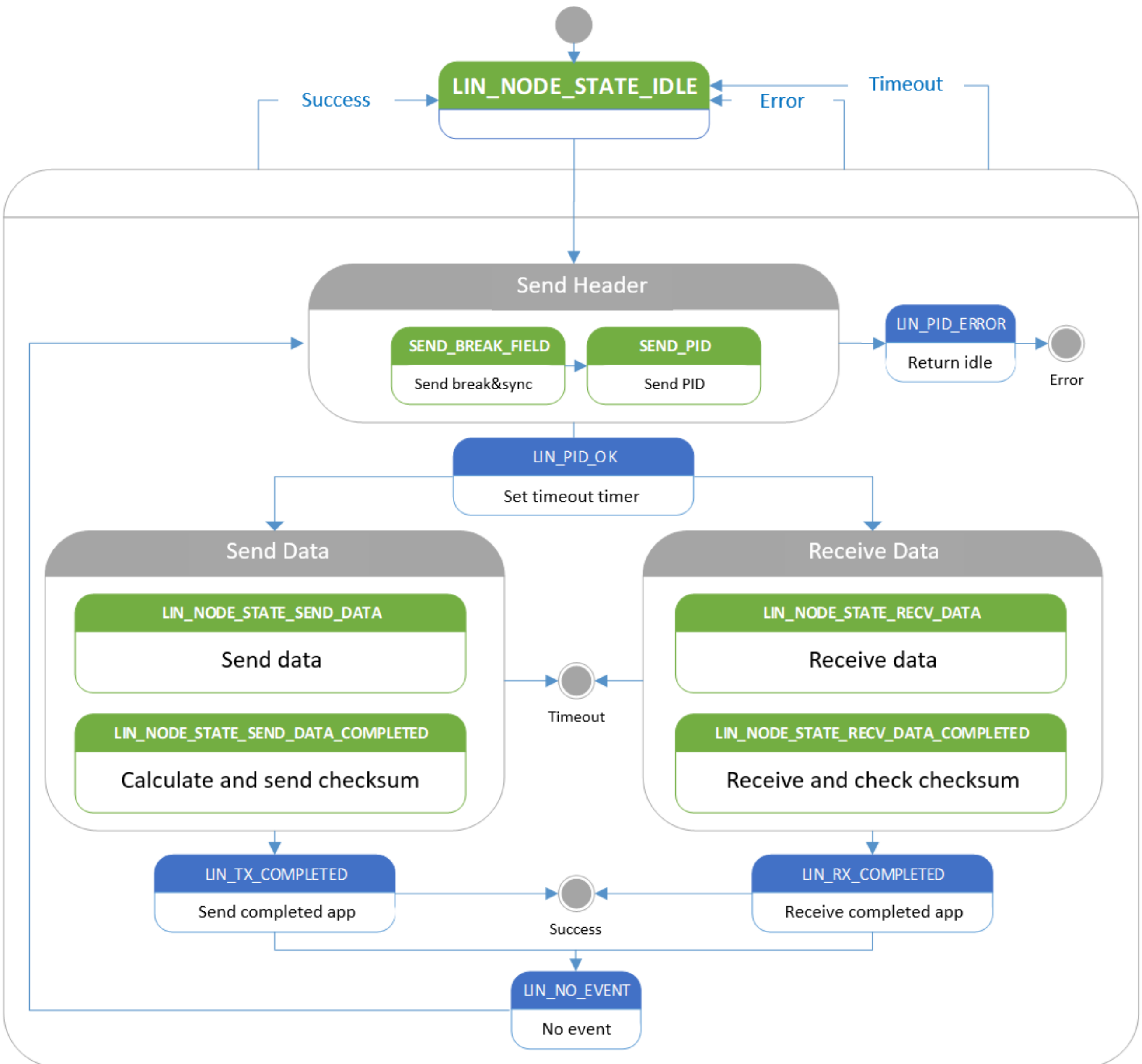

Figure 7. Master node state machine

3.3.3 Slave node state machine

Same as the master node, the state machine of slave node also contains the application layer and the driver layer. The following figure shows the state machine of LIN slave node.

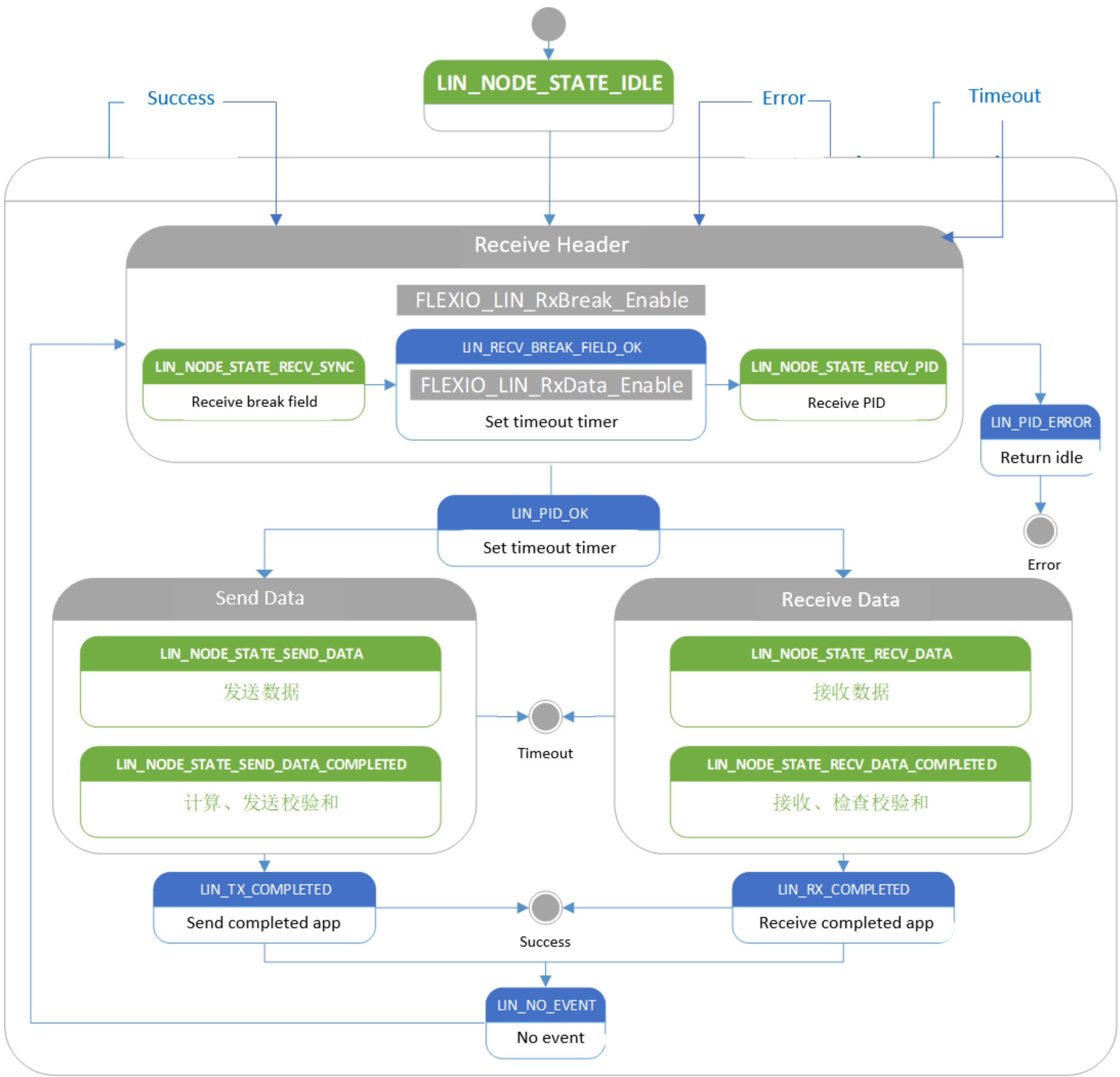

Figure 8. Slave node state machine

In this application, the slave node keeps monitoring the break field from the LIN bus. We use the *FIEXIO_LIN_RxBreak_Enable* to change the setting for receiving break field and *FLEXIO_LIN_RxData_Enable* to set configuration back to normal for receiving other fields.

4 Run the example

4.1 Development platform

This document describes the example application based on the i.MX RT1010-EVK board shown in the following figure. Users can also easily enable this application on other i.MX RT series EVK board.

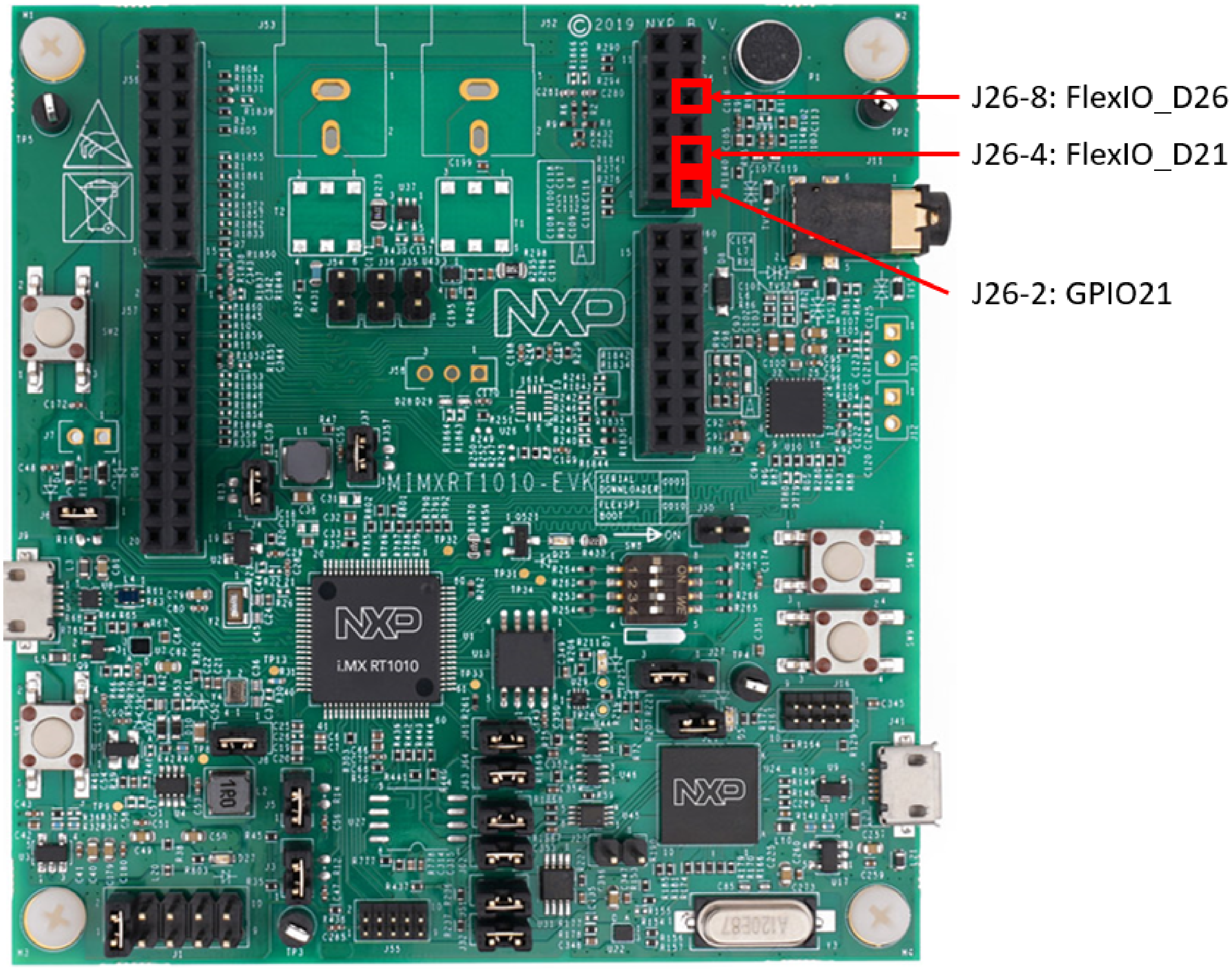

Figure 9. i.MX RT1010-EVK Board

In this application, FlexIO using FlexIO_D21 as the transmitting pin and FlexIO_D26 as the receiving pin.

This demo uses two boards, one board as master and one board as slave. In order to comply with the physical layer standard of Lin bus, a LIN transceiver is needed. This demo uses two TJA1020 modules to implement the single-wire LIN. The transceiver uses GPIO21 to enable.

The connection between LIN nodes and transceivers is as follows:

Table 5. LIN nodes and transceiver connections

Master Node		Transceiver		Transceiver		Slave Node
J26-4(LIN_TX)	↔	RX LIN	↔	LIN TX	↔	J26-8(LIN_RX)
J26-8(LIN_RX)	↔	TX INH		INH RX	↔	J26-4(LIN_TX)
J26-2(Trans_EN)	↔	SLP 12 V		12 V SLP	↔	J26-2(Trans_EN)
J60-14(GND)	↔	GND GND	↔	GND GND	↔	J60-14(GND)

4.2 Run the demo

Users can download the software from www.nxp.com. Find the IAR project *flexio_LIN*. This demo contains the master node and slave node codes which are separated by a macro. In *flexio_LIN_driver.h* file, set the macro FLEXIO_LIN_MODE as FLEXIO_LIN_SLAVE_MODE and FLEXIO_LIN_MASTER_NODE, and separately download to two boards, connect these boards and transceiver modules as above and run the two boards.

The following figure shows the waveform that an oscilloscope captures from the LIN bus.

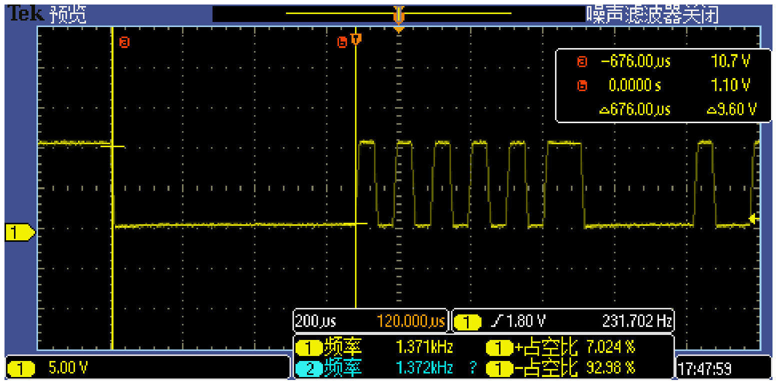

Figure 10. Header of LIN captured by oscilloscope

The following figure shows the information printed by debug port when the master node controls the slave node’s LED brightness. Users can type in the characters ‘L’,’M’, or ‘H’ in the master node’s debug console to change the remote slave node’s LED brightness.

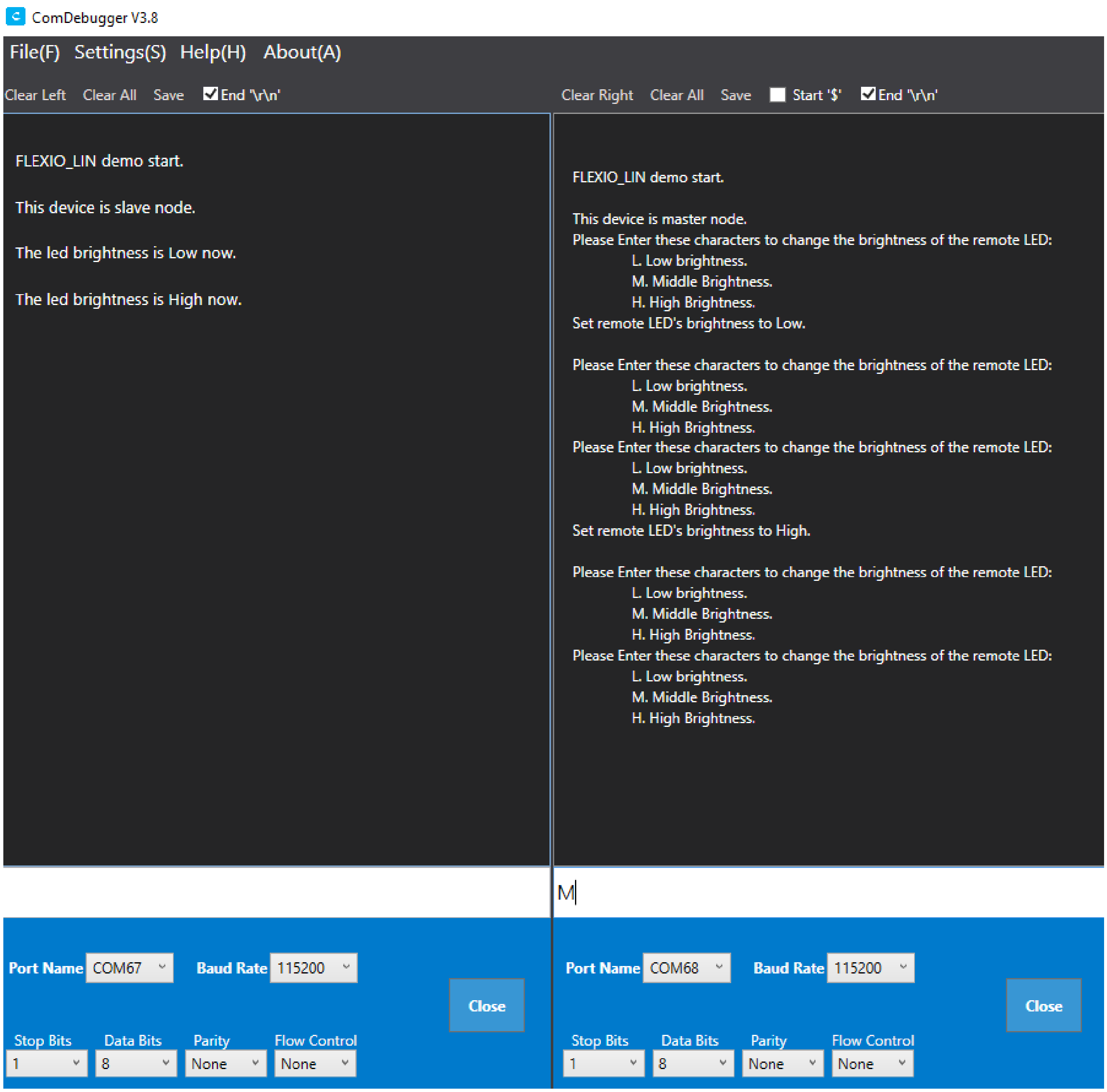

Figure 11. Control information printed by debug console

5 References

1. i.MX RT1010 Processor Reference Manual (document I.MXRT1010RM)
2. MCUXpresso SDK: Software Development Kit for NXP MCUs (https://mcuxpresso.nxp.com/en/welcome).

6 Revision history

Table 6. Revision history

Revision number	Date	Substantive changes
0	03/2020	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: March, 2020

Document identifier: AN12788