

48 V Motor Control Reference Design Based on LIN Communication

PRODUCT OVERVIEW

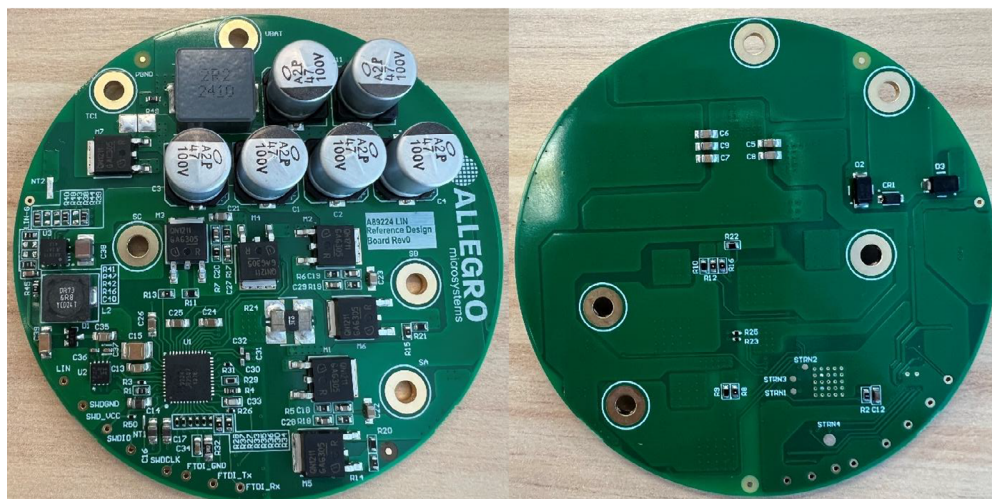
Introduction

This reference design aims to provide an efficient, highly integrated, and cost-effective solution for local interconnect network (LIN) based motor control systems in automotive applications. The document details how to build a LIN-based motor control system using the Allegro A89224 high-performance 48 V SoC in conjunction with the APM81815 DC-DC voltage conversion chip.

The document covers the complete system architecture of the LIN reference design, including functional descriptions

of each module on the demonstration board, hardware schematics and bill of materials, PCB layout, example code, debugging procedures, and waveform analysis. Additionally, it introduces how to use the Allegro graphical user interface (GUI) motor control tuning tool for parameter adjustment, enabling fast startup and precise control.

The document also describes how to perform key operations such as motor start/stop, direction control, speed setting, and sleep/wake-up via the LIN bus interface tool, helping users quickly complete functional verification, system debugging, and prototype design.



**Figure 1: A89224 48 V Motor Control Reference Design
Demo Board Based on LIN Communication**

Table of Contents

Product Overview.....	1	Firmware.....	11
Introduction	1	LIN Reference Design Example Code Main.c	11
Key Features.....	2	LIN Peripheral Module Initialization	16
Recommended Reading	2	LIN Interrupt Service Routine	18
Hardware	3	Tuning Steps	23
System Architecture Overview	3	Test Waveform	30
Functional Description of LIN Reference Design Demo Board....	4	Test Condition	30
Hardware Schematic and Bill-of-Materials	5	Waveform	30
		Revision History	39

Notwithstanding anything to the contrary in this document, any application-related information, schematics, support, or otherwise that may be provided by Allegro, the customer agrees that it retains sole responsibility to validate the suitability of Allegro's products and any support information provided herein for customers application.

Key Features

- **Input Voltage:** DC 12 V to 48 V, compatible with various automotive power supply systems.
- **Maximum Output Power:** Up to 800 W.
- **Maximum Continuous (RMS) Current:** Up to 15 A.
- **Motor Control Method:** Uses sensorless single-shunt field-oriented control (FOC) technology for high-efficiency and precision motor control.
- **Startup and Protection Mechanisms:** Supports AC alignment, IPD, and windmilling startup, with integrated stall protection to enhance system robustness and reliability.
- **LIN Communication:** The A89224 features an integrated LIN communication data link layer with hardware-based automatic baud rate detection and frame synchronization, simplifying system configuration and reducing software overhead.
- **Fault Diagnosis and Protection:** Incorporates comprehensive fault detection and protection mechanisms, including power-on reset, overcurrent, overtemperature, undervoltage, etc.
- **Debugging and Expansion Interfaces:** Supports universal asynchronous receiver-transmitter (UART) and serial wire debug (SWD) interfaces for convenient firmware programming and debugging.
- **Graphical Tuning Tool:** Fully compatible with the Allegro GUI motor control tuning tool, enabling rapid and intuitive parameter adjustment for optimized performance.

Recommended Reading

A review of the related reference documents is recommended prior to reading this document. This preliminary step will facilitate a better understanding of the technical content and application background presented. All these documents can be found in the Allegro customer portal (<https://registration.allegromicro.com/#/parts/A89224>).

- A89224 datasheet
- A89224 hardware manuals
- A89211_12_24_Motor control quick starter guide
- A892xx_MotorControlAPI
- A892xx_Application Note_LIN Example

HARDWARE

System Architecture Overview

This reference design is compatible with DC power supply systems ranging from 12 V to 48 V. The main controller is the Allegro A89224 high-performance system-on-chip (SoC). This highly integrated device features an embedded ARM Cortex-M4 microcontroller (MCU), a three-phase gate driver for motor control, and a precision differential current sensing amplifier. All these features enable sensorless single-shunt FOC motor control technology, delivering efficient and accurate motor operation with minimal external components. Its integration significantly reduces system complexity and footprint, making it ideal for compact automotive applications.

The A89224 integrates a back electromotive force (BEMF) detection module, which provides robust support for windmilling startup functionality. The GPIO0 to GPIO7 pins can be flexibly configured as LIN communication interfaces for both transmission and reception. The WAKE pin is connected to the INH pin of the LIN transceiver, enabling wake-up control via the LIN bus. If no message frames are transmitted on the LIN bus within the maximum predefined idle timeout period, then the A89224 enters

sleep mode and all internal LDOs are automatically shut down, achieving true low-power consumption operation.

In addition, the GPIO0 to GPIO7 pins of the A89224 can also be flexibly configured as UART interfaces for data transmission and reception.

Through the UART interface, users can establish an interaction with the Allegro graphical user interface (GUI) tool, enabling real-time tuning and optimization of motor control parameters. This feature provides an efficient and intuitive method for system debugging and performance calibration, significantly enhancing development productivity and the overall debugging experience.

The A89224 also supports programming and debugging via the SWD interface using a J-Link debugger. This interface provides a reliable and efficient method for downloading firmware and performing real-time debugging.

In terms of power management, this reference design uses the Allegro APM81815 DC-DC conversion chip to supply a stable 12 V DC output. It powers all internal LDO modules of the A89224 as well as the LIN transceiver, ensuring reliable and consistent system operation under various operating conditions.

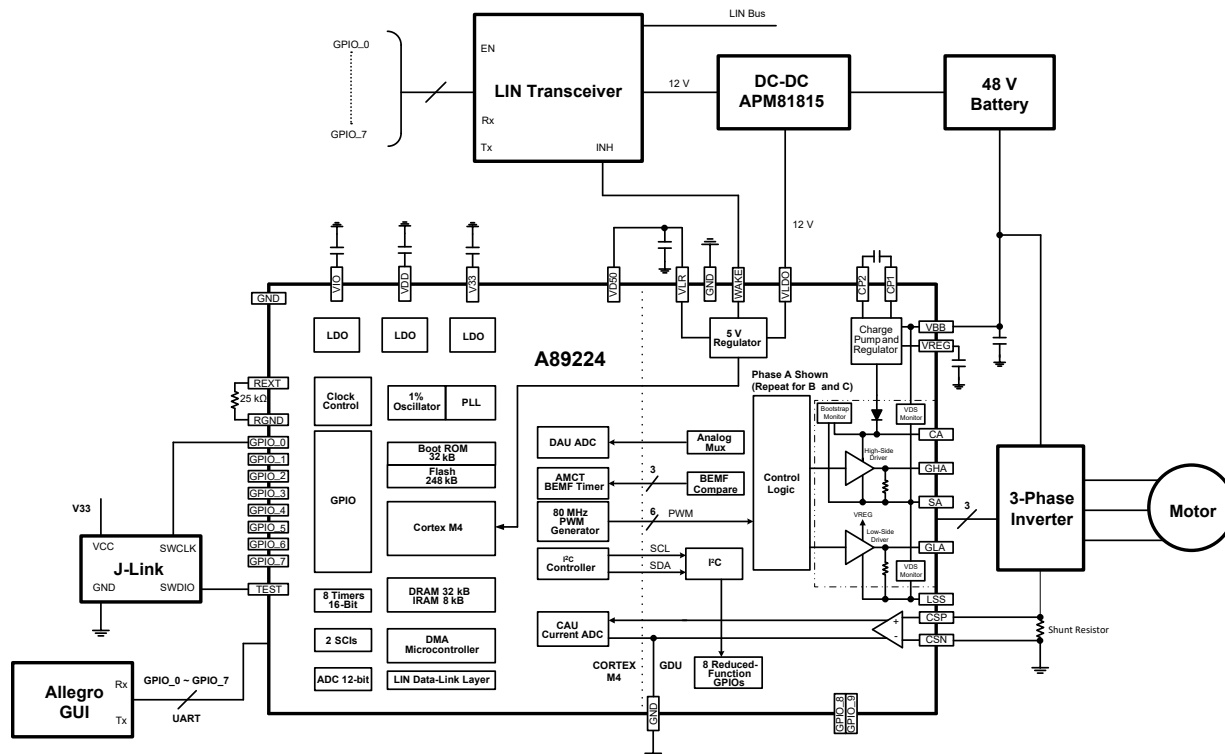


Figure 2: LIN Reference Design System Architecture Diagram

Functional Description of LIN Reference Design Demo Board

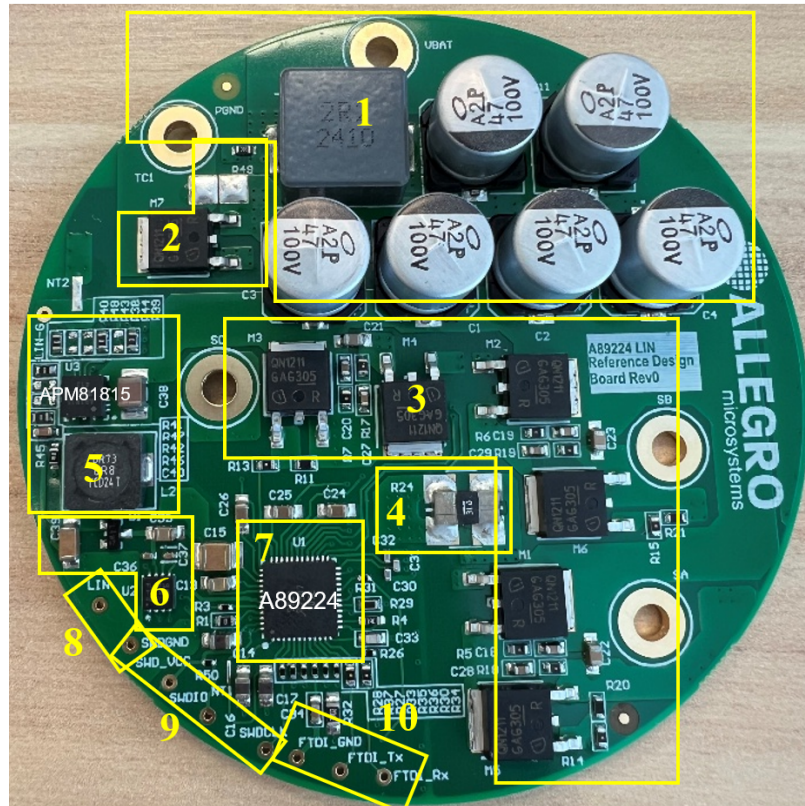


Figure 3: Hardware Sections

Table 1: Description of Hardware Sections

Section	Description
1	48 V DC bus CLC filter circuit
2	Reverse-polarity protection circuit
3	Three-phase inverter bridge circuit, connects to motor windings via SA, SB, and SC terminals
4	Shunt resistor for sensing bus current
5	Allegro APM81815 DC-DC 48 V to 12 V conversion circuit
6	LIN transceiver circuit
7	Allegro A89224 SoC (MCU + gate driver)
8	LIN communication interface
9	J-Link SWD interface
10	UART communication interface

48 V Motor Control Reference Design Based on LIN Communication User Guide

Hardware Schematic and Bill-of-Materials

Table 2: Description of the Hardware Schematic

Schematic Drawing Number	Description
Schematic 1	A89224 interconnections. The overall circuit architecture includes DC bus filter circuit, reverse-polarity protection module, three-phase inverter module, DC-DC voltage conversion module, and LIN transceiver module.
Schematic 2	Three-phase inverter circuit.
Schematic 3	A89224 GPIO pin configuration, including SWD interface circuit, UART interface circuit, LIN communication interface circuit, etc.
Schematic 4	LIN transceiver circuit.
Schematic 5	APM81815 DC-DC voltage conversion circuit.
Schematic 6	Reverse polarity protection circuit.

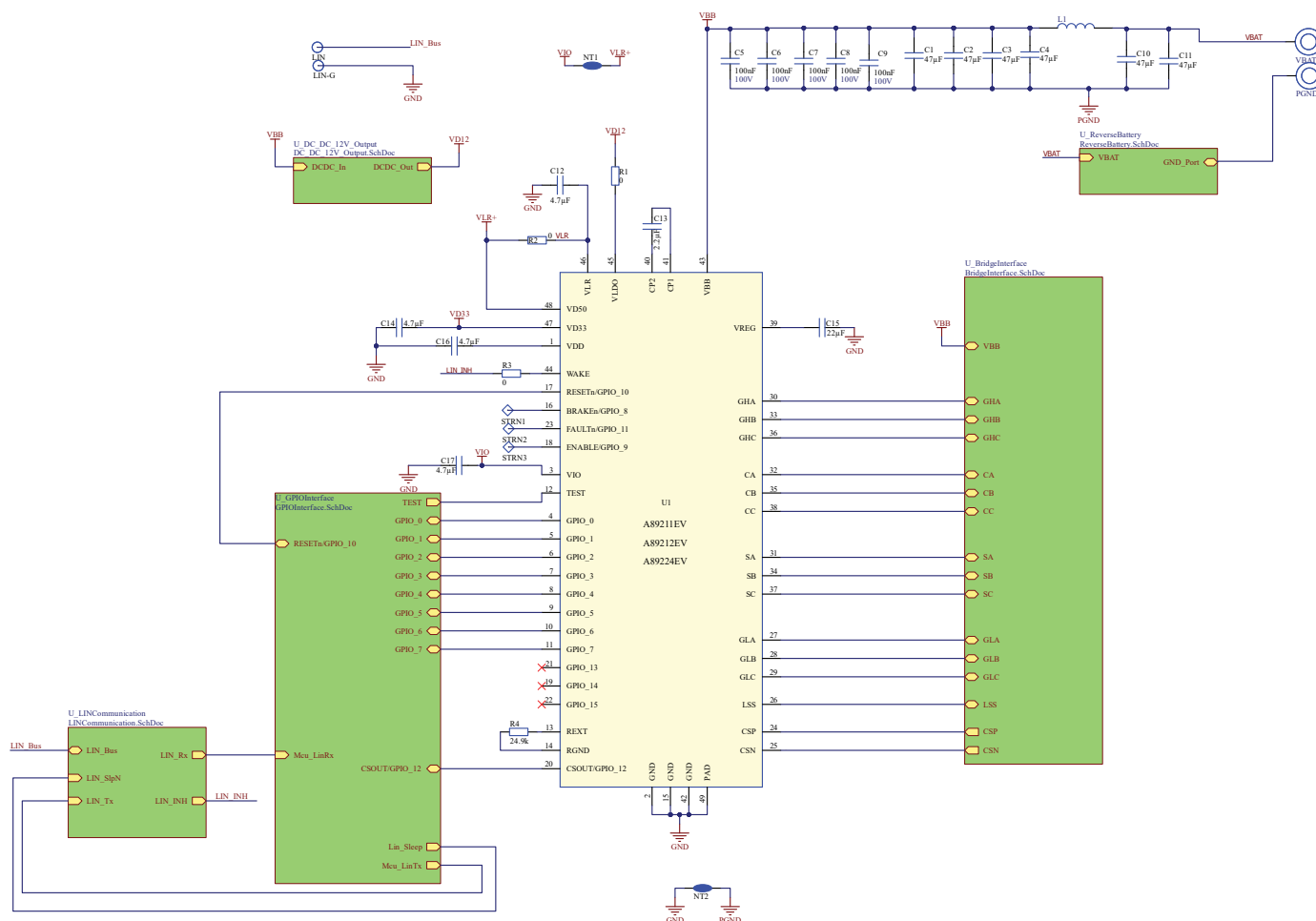


Figure 4: Schematic 1

48 V Motor Control Reference Design Based on LIN Communication User Guide



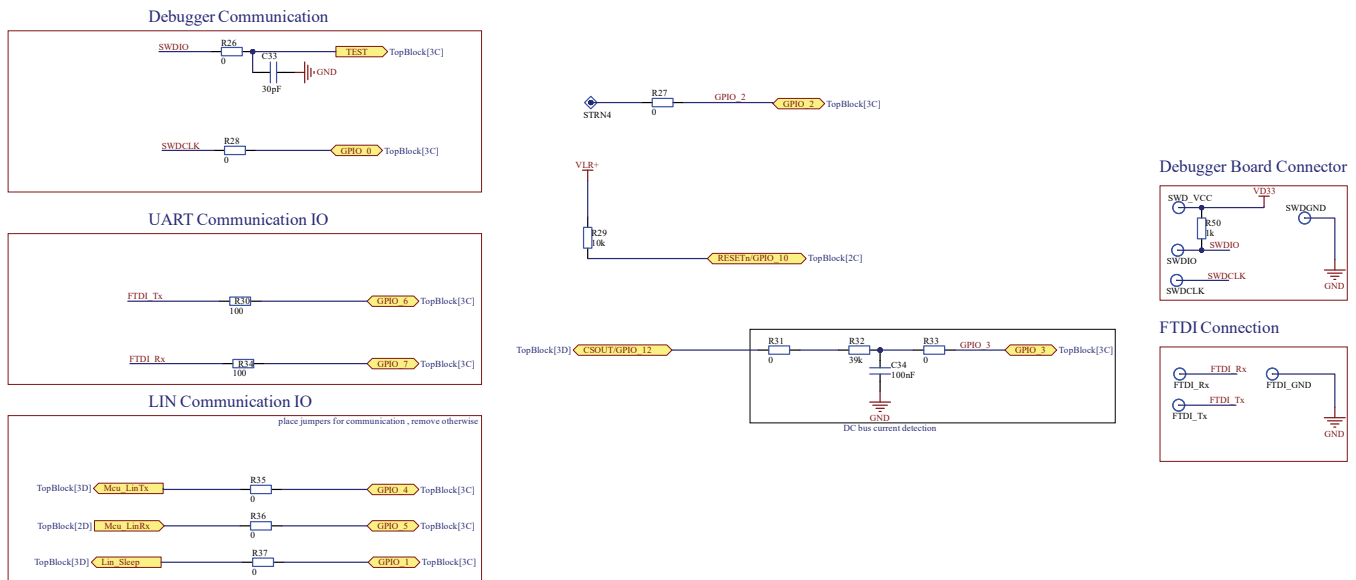


Figure 6: Schematic 3

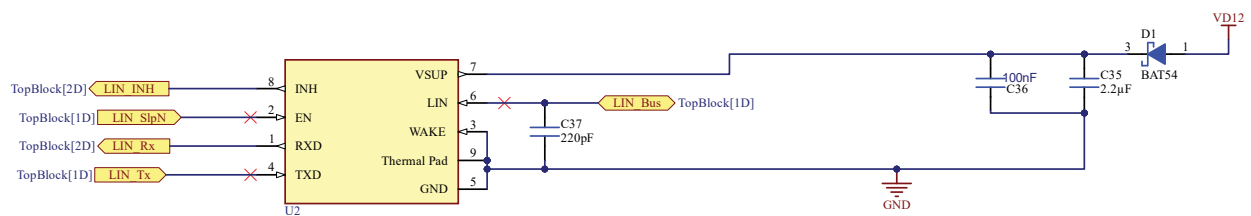


Figure 7: Schematic 4

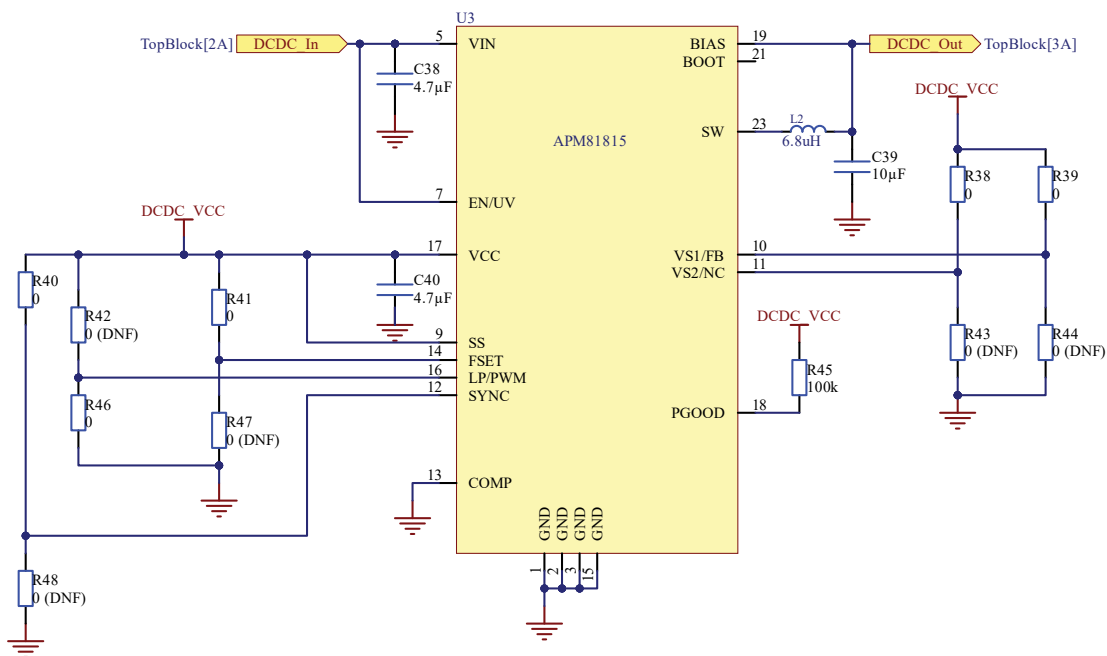


Figure 8: Schematic 5

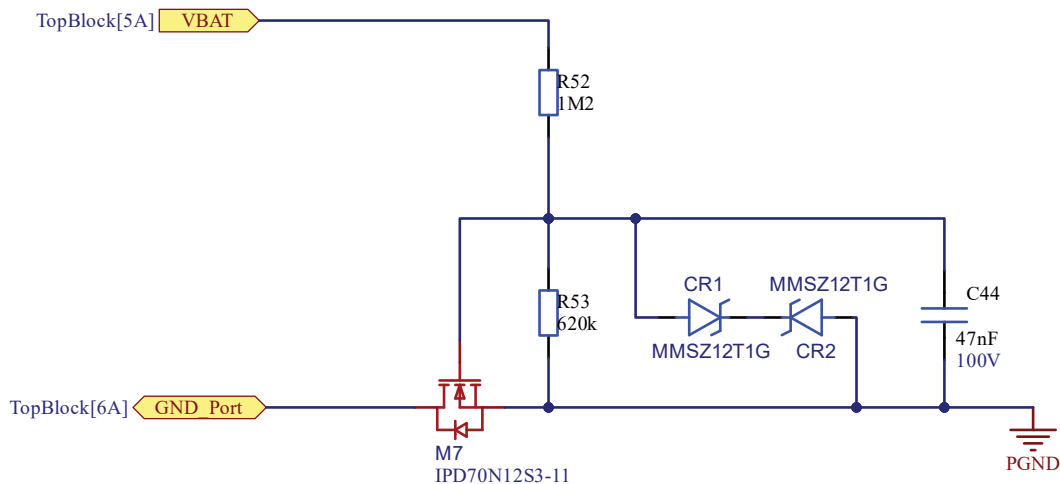


Figure 9: Schematic 6

Table 3: BOM of LIN Reference Design Demo Board

Designator	Description	Quantity	Part Number	Manufacturer
C1, C2, C3, C4, C10, C11	Polarized, 100 V, 20%, 47 μ F, Surface Mount, 4141	6	UUX2A470MNL1GS	Nichicon
C5, C6, C7, C8, C9, C21, C22, C23	Ceramic Capacitor, 100 V, 10%, X7R, 100 nF, 0805	8	08051C104KAT2A	Kyocera AVX
C12	Ceramic Capacitor, 6.3 V, 10%, X7R, 4.7 μ F, 0603	1	GCJ188C70J475KE02D	Murata
C13, C35	Ceramic Capacitor, 25 V, 10%, X7R, 2.2 μ F, 0805	2	GCM21BR71E225KA73L	Murata
C14, C16, C17	Ceramic Capacitor, 6.3 V, 10%, X7R, 4.7 μ F, 0805	3	C0805C475K9RACTU	KEMET
C15	Ceramic Capacitor, 25 V, 10%, X7R, 22 μ F, 1210	1	GRM32ER71E226KE15L	Murata
C18, C19, C20, C27, C28, C29	Ceramic Capacitor, 200 V, 10%, X7R, 4.7 nF, 0603	6	C0603C472K2RACAUTO	KEMET
C24, C25, C26	Ceramic Capacitor, 50 V, 10%, X7R, 330 nF, 0805	3	GRM219R71H334KA88D	Murata
C30, C32	Ceramic Capacitor, 35 V, 2%, C0G, 0.01 μ F, 0402	2	GRM1555CYA103GE01D	Murata
C31	Ceramic Capacitor, 50 V, 2%, C0G, 0.0047 μ F, 0402	1	GRM1555C1H472GE01D	Murata
C33	Ceramic Capacitor, 250 V, 5%, C0G, 0.00003 μ F, 0603	1	VJ0603D300JXPAJ	Vishay
C34	Ceramic Capacitor, 50 V, 10%, X7R, 0.1 μ F, 0603	1	C0603R104K5RACTU	KEMET
C36	Ceramic Capacitor, 50 V, 10%, X7R, 100 nF, 0402	1	GRM155R71H104KE14J	Murata
C37	Ceramic Capacitor, 100 V, 10%, X7R, 0.00022 μ F, 0402	1	GCM155R72A221KA37D	Murata
C38	Ceramic Capacitor, 100 V, 10%, X7R, 4.7 μ F, 1210	1	12101C475K4Z2A	Kyocera AVX
C39	Ceramic Capacitor, 25 V, 10%, X7R, 10 μ F, 1206	1	GRM31CR71E106KA12L	Murata
C40	Ceramic Capacitor, 16 V, 10%, X5R, 4.7 μ F, 0603	1	GRT188R61C475KE13D	Murata
C44	Ceramic Capacitor, 100 V, 10%, X7R, 47 nF, 0805	1	08051C473KAT2A	Kyocera AVX
CR1, CR2	Zener Diode, 12 V, 500 mW, SOD-123	1	MMSZ4699-E3-08	Vishay
D1	Diode: Schottky, 30 V, 0.2 A, SOT-23-3	1	BAT54,215	Nexperia
FTDI_GND, FTDI_Rx, FTDI_Tx, LIN, LIN-G, SWD_VCC, SWDCLK, SWDGND, SWDIO	PAD, TipNBarrel, 1.5 mm Pad, 0.75 mm Hole	9	NC	–
L1	Inductor, 2.2 μ H, 20%, SMD, 5349	1	SRP1265A-2R2M	Bourns
L2	Inductor, 6.8 μ H, 20 %, 3.12 Asat, 43.5 m Ω DCR, SMD	1	DR73-6R8-R	Eaton Electronics
M1, M2, M3, M4, M5, M6, M7	MOSFET, 70 A I(D), 120 V, 0.0111 Ω , N-Channel, TO-252-3	7	IPD70N12S311ATMA1	Infineon
NT1	PCB : Net-Tie, 0.4 mm	1	NC	–
PGND, SA, SB, SC, VBAT	Plated through mounting hole, size M3	5	NC	–
R1, R2, R38, R39, R40, R41, R42, R43, R44, R46, R47, R48	Resistor, 0 Ω , 1%, 0.1 W, 0603	12	0603WAF0000T5E	UNI-ROYAL
R3, R26, R27, R28, R31, R33, R35, R36, R37	Resistor, 0 Ω , 1%, 0.0625 W, 0402	9	RMCF0402ZT0R00	SEI Stackpole
R4	Resistor, 24.9 k Ω , 0.1%, 0.1 W, 0603	1	RT0603BRE0724K9L	YAGEO
R5, R6, R7, R17, R18, R19	Resistor, 2.2 Ω , 1%, 0.25 W, 0603	6	ESR03EZPF2R20	ROHM
R8, R12, R13, R14, R15, R16, R30, R34	Resistor, 100 Ω , 1%, 0.25 W, 0603	8	ESR03EZPF1000	ROHM
R9, R10, R11, R20, R21, R22	Resistor, 100 k Ω , 1%, 0.25 W, 0603	6	ERJ-PA3F1003V	Panasonic
R23, R25	Resistor, 4.99 Ω , 0.1%, 0.0625 W, 0402	2	CPF0402B4R99E1	TE
R24	Resistor, 0.001 Ω , 1%, 3 W, 2512	1	WSLP25121L000FEA	Vishay

Continued on next page...

Table 3: BOM of LIN Reference Design Demo Board

Designator	Description	Quantity	Part Number	Manufacturer
R29	Resistor, 10 k Ω , 1%, 0.1 W, 0603	1	ERJ-3EKF1002V	Panasonic
R32	Resistor, 39 k Ω , 1%, 0.125 W, 0805	1	0805W8F3902T5E	UNI-ROYAL
R45	Resistor, 100 k Ω , 1%, 0.1 W, 0603	1	RMCF0603FT100K	SEI Stackpole
R49	Resistor, 20 k Ω , 1%, 0.1 W, 0603	1	RMCF0603FT20K0	SEI Stackpole
R50	Resistor, 1 k Ω , 1%, 0.1 W, 0402	1	ERJ-2RKF1001X	Panasonic
R52	Resistor, 1.2 M Ω , 1%, 0.1 W, 0603	1	0603WAF1204T5E	UNI-ROYAL
R53	Resistor, 620 k Ω , 1%, 0.1 W, 0603	1	0603WAF6203T5E	UNI-ROYAL
STRN1, STRN2, STRN3	Test Point, SMT Pad, 1 mm	3	NC	–
STRN4	Test Point, SMT Pad, 2 mm	1	NC	
TC1	Track Cut, 3 mm	1	NC	
U1	A89224EV, QFN-48	1	–	Allegro
U2	60 V abs max LIN Transceiver, TLIN2021ADRBRQ1, PDSO8	1	TLIN2021ADRBRQ1	TI
U3	APM81815 in MIS package	1	–	Allegro

FIRMWARE

LIN Reference Design Example Code Main.c

The LIN reference design example code allows users to control a motor either through the Allegro GUI or via LIN communication for direct command-based control.

When selecting motor control via the Allegro GUI, UART communication function must be enabled. Refer to the example code in Figure 10.

```

80 void System_init(void);
81 static __attribute__((always_inline)) inline
82 void SetupInterruptPriority(void);
83
84 #define UART_COMMUNICATION
85
86 __asm(".global __ARM_use_no_argv\n\t");
87 int main(int argc, char **argv)
88 {

```

Figure 10: Enable UART Communication Function

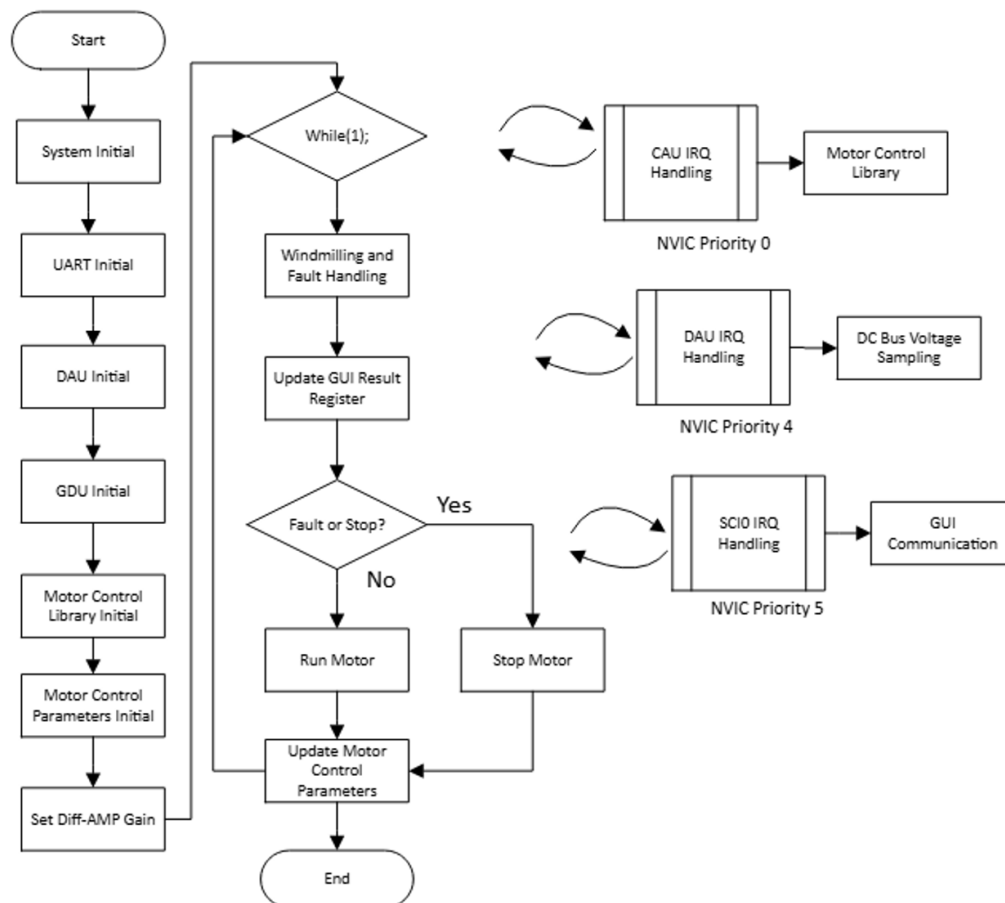


Figure 11: Flowchart of Allegro GUI Motor Control Main.c

```

87 int main(int argc, char **argv)
88 {
89     System_init(); System Initial
90
91     #ifdef LIN_COMMUNICATION
92     LinCommunication_Initialize();
93     #endif
94
95     #ifdef UART_COMMUNICATION
96     UartCommunication_Initialize(6,7,eUartCommunication_UartProduct_A89211_12_24); UART Initial
97     #endif
98
99     DauHandling_PeripheralInitialize(); DAU Initial
100    GduInterface_Initialise(); GDU Initial
101    MotorControlInterface_Initialize(); Motor Control Library Initial
102    UpdMotControlSettings_Initialize(); Motor Control Parameters Initial
103
104    #ifdef LIN_COMMUNICATION
105    GduInterface_SetSenseAmpGain( MotorControlSettings_SAG );
106    #endif
107
108    #ifdef UART_COMMUNICATION
109    GduInterface_SetSenseAmpGain(0); Set diff-AMP Gain
110    #endif
111
112    while(1)
113    {
114        #ifdef LIN_COMMUNICATION
115        FaultHandling_Run_LinComm();
116        if( FaultHandling_GetStatus() | (!LinCommunication_IsMotorRunCommand()))
117        {
118            MotorControlInterface_Stop();
119        }
120        else
121        {
122            MotorControlInterface_Run();
123        }
124
125        UpdMotControlSettings_LinCom( LinCommunication_aul6Registers, &LinCommunication_
126
127        if (LinCommunication_IsSleepCommand())
128        {
129            GduInterface_GoToSleep();
130        }
131        #endif
132
133        #ifdef UART_COMMUNICATION
134        /* During windmil GduInterface must be avoided as bemf comparator is constant.
135        if(MotorControlInterface_GetStateMachine() !=eMotCont_MCSM_WindMil)
136        {
137            if((MotorControlInterface_GetStateMachine() == eMotCont_MCSM_StandBy)&&(Uar
138            {
139                GduInterface_SetSenseAmpGain(UartCommunication_aul6Registers[3]); Windmilling handling
140            }

```

Figure 12: Code Description of Allegro GUI Motor Control Main.c

Continued on next page...

```

141     else
142     {
143         FaultHandling_Run_UartComm(UartCommunication_aul6Registers[39]); Fault Handling
144     }
145 }
146 else
147 {
148     ;
149 }
150     UpdateGuiResultRegisters();
151     if(FaultHandling_GetStatus() | (!UartCommunication_IsMotorRunCommanded()))
152     {
153         MotorControlInterface_Stop(); Stop Motor
154     }
155     else
156     {
157         MotorControlInterface_Run(); Run Motor
158     }
159     UpdMotControlSettings_UartCom(UartCommunication_aul6Registers,&UartCommunication_bNewData);
160     #endif Update Motor Control Parameters
161 }
162 return 0;
163 }

```

Figure 12: Code Description of Allegro GUI Motor Control Main.c (continued)

When selecting LIN-based motor control, LIN communication function must be enabled. Refer to the example code in Figure 13.

```

80 void System_init(void);
81 static __attribute__((always_inline)) inline void U;
82 void SetupInterruptPriority(void);
83
84 #define LIN_COMMUNICATION
85
86 __asm(".global __ARM_use_no_argv\n\t");
87 int main(int argc, char **argv)
88 {

```

Figure 13: Enable LIN Communication Function

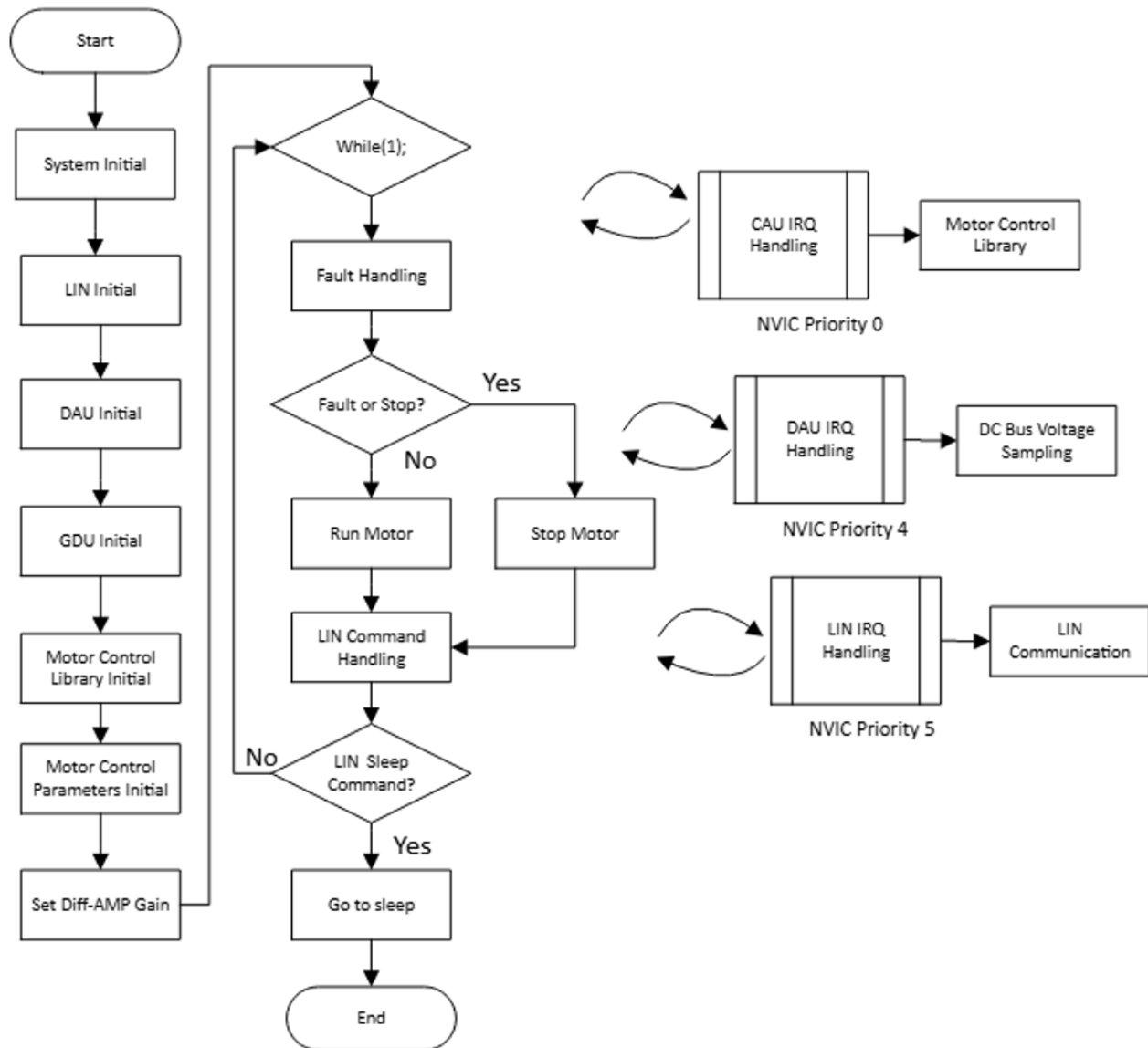


Figure 14: Flowchart of LIN-based Motor Control Main.c

```

87 int main(int argc, char **argv)
88 {
89     System_init(); System Initial
90
91     #ifdef LIN_COMMUNICATION
92     LinCommunication_Initialize(); LIN module Initial
93     #endif
94
95     #ifdef UART_COMMUNICATION
96     UartCommunication_Initialize(6,7,eUartCommunication_UartP;
97     #endif
98
99     DauHandling_PeripheralInitialize(); DAU Initial
100    GduInterface_Initialize(); GDU Initial
101    MotorControlInterface_Initialize(); Motor Control Library Initial
102    UpdMotControlSettings_Initialize(); Motor Control Parameters Initial
103
104    #ifdef LIN_COMMUNICATION
105    GduInterface_SetSenseAmpGain( MotorControlSettings_SAG ); Set diff-AMP Gain
106    #endif
107
108    #ifdef UART_COMMUNICATION
109    GduInterface_SetSenseAmpGain(0);
110    #endif
111
112    while(1)
113    {
114        #ifdef LIN_COMMUNICATION
115        FaultHandling_Run_LinComm(); Fault Handling
116        if( FaultHandling_GetStatus() | (!LinCommunication_IsMotorRunCommand()))
117        {
118            MotorControlInterface_Stop(); Stop Motor
119        }
120        else
121        {
122            MotorControlInterface_Run(); Run Motor
123        }
124
125        UpdMotControlSettings_LinCom( LinCommunication_aui6Registers, &LinCommunication_bNewData ); LIN Command Handling
126
127        if (LinCommunication_IsSleepCommand())
128        {
129            GduInterface_GoToSleep(); GDU Go-To-Sleep
130        }
131        #endif

```

Figure 15: Code Description of LIN-based Motor Control Main.c

LIN Peripheral Module Initialization

The GPIO0 to GPIO7 pins of the A89224 can be configured as LIN communication interfaces for both receive (Rx) and transmit (Tx) functions. In this reference design:

- GPIO5 is configured as the LIN Rx pin.
- GPIO4 is configured as the LIN Tx pin.
- GPIO1 is used as the Enable pin for the LIN transceiver.

In addition, the example code enables multiple LIN-related interruption service routines (ISRs), including:

- Header Received ISR: Triggered upon reception of a LIN frame header.
- Response Received ISR: Triggered when a frame response is received.
- Response Transmitted ISR: Triggered after a frame response is successfully sent.
- Go-to-Sleep ISR: Triggered when the LIN bus remains idle for a predefined duration, the A89224 device enters sleep mode.

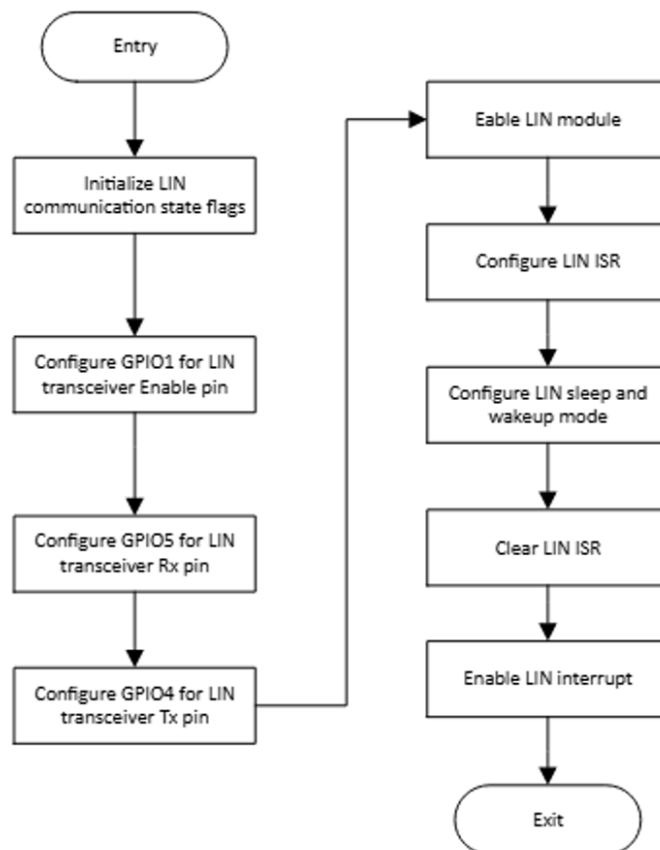


Figure 16: Flowchart of LIN Peripheral Module Initialization

```

137 void LinCommunication_Initialize()
138 {
139     // Initialize LIN communication state flags
140     bFaultClearRequest = False;
141     bMotorRunRequest = False;
142     LinCommunication_bNewData = False;
143     LinCommunication_bSleepCommand = False;
144
145     // GPIO1 used for LIN EN (Enable) pin
146     Gpio_SetPinActive( eGPIO_INSTANCE_ID1 );
147     Gpio_SetMode( eGPIO_INSTANCE_ID1, eGPIO_MODE_OUTPUT_PUSH_PULL );
148     Gpio_Set( eGPIO_INSTANCE_ID1 );
149
150     // Configure LIN system control register
151     Scu_SetLinRegister( 5 );
152
153     // GPIO5 configured as LIN RX input
154     Gpio_SetPinActive( eGPIO_INSTANCE_ID5 );
155     Gpio_SetMode( eGPIO_INSTANCE_ID5, eGPIO_MODE_DIGITAL_INPUT_PULLUP );
156
157     // GPIO4 configured as LIN TX output
158     Gpio_SetPinActive( eGPIO_INSTANCE_ID4 );
159     Gpio_SetMode( eGPIO_INSTANCE_ID4, eGPIO_MODE_OUTPUT_PUSH_PULL );
160     Gpio_SetSource( eGPIO_INSTANCE_ID4, eGPIO_ASRC8 );
161
162     // Configure LIN interrupt
163     NVIC_DisableIRQ( (IrqType_t)eIRQNUM_LIN );    Disable LIN ISR
164     NVIC_ClearPendingIRQ( (IrqType_t)eIRQNUM_LIN );    Clear LIN ISR
165
166     // Enable LIN module before any operation
167     Lin_SetEnable( True );    Enable LIN module
168
169     // Configure LIN IRQ handler and enable required interrupt flags
170     Lin_SetIrqHandler( HandleLinIrq );
171     Lin_SetIrqEnableStatusRegister( eLIN_IRQENABLE_ALL_FLAGS, False );
172     Lin_SetIrqEnableStatusRegister( eLIN_IRQENABLE_HEADER_RECEIVED, True );    Enable header received ISR
173     Lin_SetIrqEnableStatusRegister( eLIN_IRQENABLE_RESPONSE_RECEIVED, True );    Enable response received ISR
174     Lin_SetIrqEnableStatusRegister( eLIN_IRQENABLE_RESPONSE_SENT, True );    Enable response transmitted ISR
175     Lin_SetIrqEnableStatusRegister( eLIN_IRQENABLE_GOTOSLEEP, True );    Enable LIN go-to-sleep ISR
176
177     // Configure LIN bus idle timeout
178     Lin_SetBusIdleTimeOut( 6 );    LIN bus idle time-out 10 seconds
179     Lin_SetMasterInt( True );    Enable all the above enabled ISR
180     Lin_SetWakeUpRequest( True );    Wake-up request
181
182     // Re-enable interrupt
183     NVIC_ClearPendingIRQ( (IrqType_t)eIRQNUM_LIN );    Clear LIN ISR
184     NVIC_EnableIRQ( (IrqType_t)eIRQNUM_LIN );    Enable LIN ISR
185 }

```

Figure 17: Code Description of LIN Peripheral Module Initialization

LIN Interrupt Service Routine

Upon successful reception of the LIN frame header transmitted by the master, the A89224 device decodes the ID field within the header. Based on the predefinition of the received ID, it generates the corresponding frame response to execute the designated task.

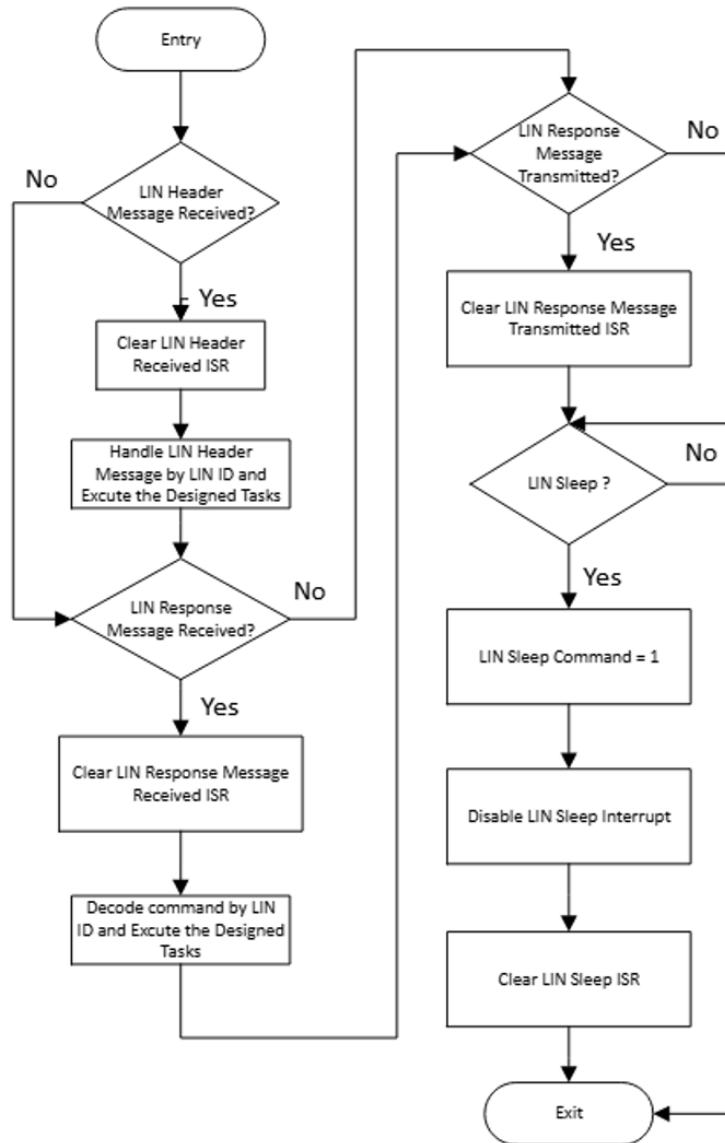


Figure 18: Flowchart of LIN ISR

```

187 /**
188  * @brief LIN interrupt service routine.
189  *      Handles header received, response received, response sent, and sleep mode
190  */
191 void HandleLinIrq( void )
192 {
193     // --- Header Received Interrupt ---
194     if( Lin_GetStatusRegFlag( eLIN_STATUS_HEADER_RECEIVED ) == True )    LIN header message received ISR
195     {
196         Lin_SetStatusRegFlag( eLIN_STATUS_HEADER_RECEIVED, True );    Clear header received ISR // Clear status fl.
197     }
198     // Handle LIN header by ID
199     switch ( Lin_GetMessageIdentifier() )
200     {
201     case
202         LIN_ID_STATUS_SPEED:    Transmit "Speed" payload
203     {
204         uint8 au8MsgTransmit[LIN_LENGTH_STATUS_SPEED] = {0,};
205         LinCommunication_RequestDecode( au8MsgTransmit, LIN_ID_STATUS_SPEED );    Prepare response payload
206         Lin_SetMessageFrame( au8MsgTransmit, LIN_LENGTH_STATUS_SPEED );    Write speed payload to LIN data register
207         Lin_SetNumResponseBytes( LIN_LENGTH_STATUS_SPEED );    Length of the payload
208         Lin_SetRxTxControl( eLIN_TX_RESPONSE );    Set as "transmit mode"
209         break;
210     }
211
212     case
213         LIN_ID_STATUS_FAULT:    Transmit "Fault" status payload
214     {
215         uint8 au8MsgTransmit[LIN_LENGTH_STATUS_FAULT] = {0,};
216         LinCommunication_RequestDecode( au8MsgTransmit, LIN_ID_STATUS_FAULT );
217         Lin_SetMessageFrame( au8MsgTransmit, LIN_LENGTH_STATUS_FAULT );
218         Lin_SetNumResponseBytes( LIN_LENGTH_STATUS_FAULT );
219         Lin_SetRxTxControl( eLIN_TX_RESPONSE );
220         break;
221     }
222
223     case
224         LIN_ID_CMD_RUN_STOP:    Receive "Run/Stop" command from master
225     {
226         Lin_SetNumResponseBytes( LIN_LENGTH_CMD_RUN_STOP );    Length of received data
227         Lin_SetRxTxControlFast( eLIN_RX_RESPONSE );    Note: Set as "receiver mode"
228         break;
229     }
230
231     case
232         LIN_ID_CMD_DIRECTION:    Receive "Direction" command from master
233     {
234         Lin_SetNumResponseBytes( LIN_LENGTH_CMD_DIRECTION );
235         Lin_SetRxTxControlFast( eLIN_RX_RESPONSE );    Note: this function MUST
236         break;
237     }
238
239     case
240         LIN_ID_CMD_SPEED:    Receive "Speed" command from master
241     {
242         Lin_SetNumResponseBytes( LIN_LENGTH_CMD_SPEED );
243         Lin_SetRxTxControlFast( eLIN_RX_RESPONSE );    Note: th
244         break;
245     }
246
247     case
248         LIN_ID_CMD_FAULT_CLEAR:    Receive "FaultClear" command from master
249     {
250         Lin_SetNumResponseBytes( LIN_LENGTH_CMD_FAULT_CLEAR );
251         Lin_SetRxTxControlFast( eLIN_RX_RESPONSE );    Note: th
252         break;
253     }
254
255     case
256         LIN_ID_DIAG_MASTER_REQ:    Receive diagnostic command from master
257     {
258         // Receive diagnostic request payload
259         Lin_SetNumResponseBytes( 8 );
260         Lin_SetRxTxControlFast( eLIN_RX_RESPONSE );    Note: th
261         break;
262     }
263 }
264
265

```

Figure 19: Code Description of LIN ISR

Continued on next page...

```

266         case
267             LIN_ID_DIAG_SLAVE_RESP:    Transmit diagnostic command
268         {
269             // Transmit diagnostic response to master
270             Lin_SetMessageFrame( kau8Reply, 8 );
271             Lin_SetNumResponseBytes( 8 );
272             Lin_SetRxTxControl( eLIN_TX_RESPONSE );
273             break;
274         }
275
276         default:
277             break;
278     }
279
280     return;
281 }

```

```

282
283 // --- Response Received Interrupt ---
284 if( Lin_GetStatusRegFlag( eLIN_STATUS_RESPONSE_RECEIVED ) == True ) LIN response message received ISR
285 {
286     Lin_SetStatusRegFlag( eLIN_STATUS_RESPONSE_RECEIVED, True ) // Clear flag Clear LIN response message received ISR
287
288     // Decode command based on LIN ID
289     switch ( Lin_GetMessageIdentifier() )
290     {
291
292     case
293         LIN_ID_CMD_RUN_STOP:    Receive "Run/Stop" command from master
294     {
295         uint8 au8MsgReceived[LIN_LENGTH_CMD_RUN_STOP] = { 0, };
296         Lin_GetMessageFrame( au8MsgReceived ); Read out received LIN data Decode the received
297         LinCommunication_CommandDecode( au8MsgReceived, LIN_ID_CMD_RUN_STOP ); data and store the
298         break; command for further
299         use
300     }
301
302     case
303         LIN_ID_CMD_DIRECTION:    Receive "Direction" command from master
304     {
305         uint8 au8MsgReceived[LIN_LENGTH_CMD_DIRECTION] = { 0, };
306         Lin_GetMessageFrame( au8MsgReceived );
307         LinCommunication_CommandDecode( au8MsgReceived, LIN_ID_CMD_DIRECTION );
308         break;
309     }
310
311     case
312         LIN_ID_CMD_SPEED:    Receive "Speed" command from master
313     {
314         uint8 au8MsgReceived[LIN_LENGTH_CMD_SPEED] = { 0, };
315         Lin_GetMessageFrame( au8MsgReceived );
316         LinCommunication_CommandDecode( au8MsgReceived, LIN_ID_CMD_SPEED );
317         break;
318     }
319
320     case
321         LIN_ID_CMD_FAULT_CLEAR:    Receive "FaultClear" command from master
322     {
323         uint8 au8MsgReceived[LIN_LENGTH_CMD_FAULT_CLEAR] = { 0, };
324         Lin_GetMessageFrame( au8MsgReceived );
325         LinCommunication_CommandDecode( au8MsgReceived, LIN_ID_CMD_FAULT_CLEAR );
326         break;
327     }
328
329     default:
330         // Unknown command, do nothing
331         break;
332     }
333     return;
334 }

```

Figure 19: Code Description of LIN ISR (continued)

Continued on next page...

```

334 // --- Response Sent Interrupt ---
335
336 if ( Lin_GetStatusRegFlag( eLIN_STATUS_RESPONSE_SENT ) == True ) LIN response message transmitted ISR
337 {
338     Lin_SetStatusRegFlag( eLIN_STATUS_RESPONSE_SENT, True ); Clear LIN response message transmitted ISR
339     return;
340 }
341
342 // --- Sleep Mode Interrupt ---
343
344 if ( Lin_GetStatusRegFlag( eLIN_STATUS_SLEEPMODE ) == True ) LIN sleep mode ISR
345 {
346     LinCommunication_bSleepCommand = True; Set sleep command
347     Lin_SetIrqEnableStatusRegister( eLIN_IRQENABLE_GOTOSLEEP, False ); Disable LIN sleep mode ISR
348     Lin_SetStatusRegFlag( eLIN_STATUS_SLEEPMODE, True ); // Clear flag Clear LIN sleep mode ISR
349     return;
350 }
351 }

```



```

356 void LinCommunication_CommandDecode(uint8 au8Msg[], uint8 u8CommandType) Decode the received
357 { data and store the
358     boolean bChanged = False; command for further
359                                     use
360     switch ( u8CommandType )
361     {
362         case
363             LIN_ID_CMD_RUN_STOP:
364         {
365             uint16 ul6NewVal = (uint16)au8Msg[0];
366             if ( LinCommunication_aul6Registers[REGIDX_RUN_STOP_CMD] != ul6NewVal )
367             {
368                 LinCommunication_aul6Registers[REGIDX_RUN_STOP_CMD] = ul6NewVal;
369                 bChanged = True;
370             }
371             break;
372         }
373
374         case
375             LIN_ID_CMD_DIRECTION:
376         {
377             uint16 ul6NewVal = (uint16)au8Msg[0];
378             if ( LinCommunication_aul6Registers[REGIDX_DIRECTION_CMD] != ul6NewVal )
379             {
380                 LinCommunication_aul6Registers[REGIDX_DIRECTION_CMD] = ul6NewVal;
381                 bChanged = True;
382             }
383             break;
384         }

```



```

385
386         case
387             LIN_ID_CMD_SPEED:
388         {
389             uint16 ul6NewVal = ((uint16)au8Msg[0] << 8) | au8Msg[1];
390             if ( LinCommunication_aul6Registers[REGIDX_SPEED_CMD] != ul6NewVal )
391             {
392                 LinCommunication_aul6Registers[REGIDX_SPEED_CMD] = ul6NewVal;
393                 bChanged = True;
394             }
395             break;
396         }
397
398         case
399             LIN_ID_CMD_FAULT_CLEAR:
400         {
401             uint16 ul6NewVal = (uint16)au8Msg[0];
402             if ( LinCommunication_aul6Registers[REGIDX_FAULT_CMD] != ul6NewVal )
403             {
404                 LinCommunication_aul6Registers[REGIDX_FAULT_CMD] = ul6NewVal;
405                 bChanged = True;
406             }
407             break;
408         }

```

Figure 19: Code Description of LIN ISR (continued)

Continued on next page...

```

409         default:
410             // Unknown command, do nothing
411             break;
412     }
413
414     if (bChanged)
415     {
416         LinCommunication_bNewData = True;
417     }
418 }
419
424 void LinCommunication_RequestDecode(uint8* pu8Buf, uint8 u8ReqType) Prepare response payload
425 {
426     if ( u8ReqType == LIN_ID_STATUS_SPEED ) Speed payload
427     {
428         uint16 ul6Speed = MotorControlInterface_GetMotorSpeed();
429         pu8Buf[0] = (uint8) ((ul6Speed & 0xFF00) >> 8);
430         pu8Buf[1] = (uint8) ( ul6Speed & 0x00FF);
431         for( int i=2; i <= 7; i++ ) { pu8Buf[i] = 0; }
432         return;
433     }
434     else if( u8ReqType == LIN_ID_STATUS_FAULT ) Fault status payload
435     {
436         uint16 ul6FaultStatus = FaultHandling_GetStatus();
437         pu8Buf[0] = (uint8) ( ( ul6FaultStatus >> 15) & 0x01 ); // POR
438         pu8Buf[1] = (uint8) ( ( ul6FaultStatus >> 14) & 0x01 ); // SE
439         pu8Buf[2] = (uint8) ( ( ul6FaultStatus >> 12) & 0x01 ); // MF
440         pu8Buf[3] = (uint8) ( ( ul6FaultStatus >> 11) & 0x01 ); // OT
441         pu8Buf[4] = (uint8) ( ( ul6FaultStatus >> 7) & 0x01 ); // BU
442         pu8Buf[5] = (uint8) ( ( ul6FaultStatus >> 6) & 0x01 ); // VDS
443         pu8Buf[6] = (uint8) ( ( ul6FaultStatus >> 4) & 0x01 ); // VLR
444         pu8Buf[7] = (uint8) ( 0 );
445         return;
446     }
447 }

```

Figure 19: Code Description of LIN ISR (continued)

TUNING STEPS

1. Connect the three-phase motor to the demo board using SA, SB, and SC connectors.



**Figure 20: LIN Reference Design Demo Board
SA, SB, and SC Connectors**

2. Before powering on the demo board, ensure that the output of the DC power supply is turned off. Connect the positive terminal of the power supply to VBAT, and the negative terminal to PGND. Select an appropriate input voltage between 12 V to 48 V.



Figure 21: Power Supply Connectors on Demo Board

3. Connect the SWD interface of the J-Link debugger to the corresponding SWDGND, SWD_VCC, SWDIO, and SWDCLK pins on the demo board. Then, connect the J-Link to the computer via USB to enable firmware programming and online debugging.



Figure 22: SWD Interface on Demo Board

4. After verifying all connections, switch on the DC power supply for the demo board.
5. Next, program and download the LIN reference design example code to the A89224 chip. Launch the Keil V5.28 development environment. The versions of various Keil components are as follows:

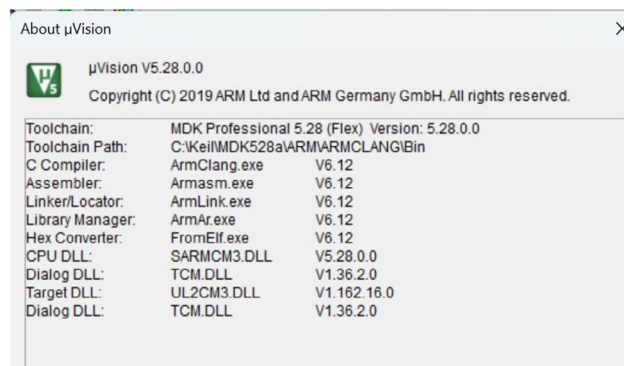


Figure 23: Version Information

6. Load the LIN reference design example code in the Keil. Open the project file named "FieldOrientedControl.uvprojx", located in the folder A89211_12_24_MotorControl_FOC_v1.1.0_Lin&UartInterface.

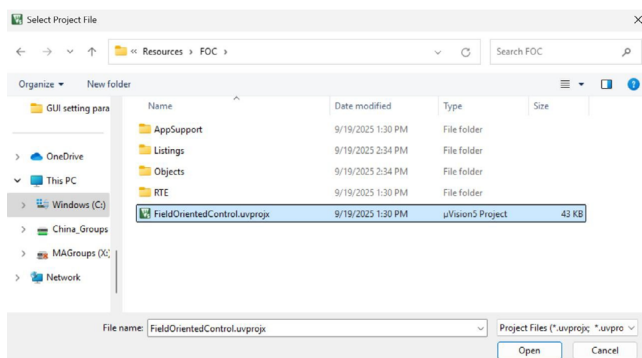


Figure 24: Load LIN Reference Design Example Code

7. Before using the Allegro GUI to tune motor control parameters, UART communication function must be enabled. Modify the configuration according to the following example code.

```

72  /*
73  */
74
75  #include "MotorControlSdkHeaders.h"
76  #include "MotorControlHeaders.h"
77  #include "AppSupport.h"
78
79
80  void System_init(void);
81  static __attribute__((always_inline))
82  void SetupInterruptPriority(void);
83
84  #define UART_COMMUNICATION
85
86  __asm(".global __ARM_use_no_argv\n\t");
87  int main(int argc, char **argv)
88  {
89      System_init();
  
```

Figure 25: Enable UART Communication Function

8. After completing the code modifications, click the “Build” button in the Keil toolbar to compile the project.

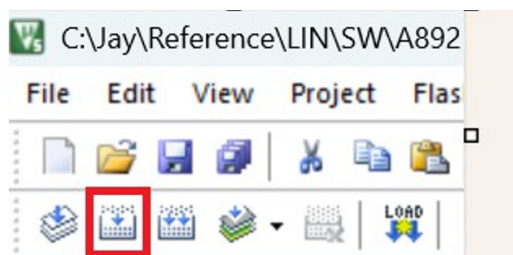


Figure 26: “Build” button in the Keil toolbar

9. Once the example code is successfully compiled, the following message will be displayed in the Keil:

```

Build Output
FromELF: creating hex file...
".\Objects\Debug02.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:01:57
  
```

Figure 27: Successful Compilation Message

10. Next, click the “Download” button in the Keil toolbar to download the example code into the A89224.

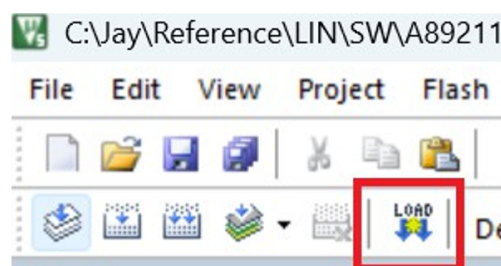


Figure 28: “Download” button in the Keil toolbar

11. The following message will be displayed once the code is successfully downloaded.

```

Build Output
Full Chip Erase Done.
No Algorithm found for: 20000000H - 20000007H
Partial Programming Done (areas with no algorithms skipped!)
Partial Verify OK (areas with no algorithms skipped!)
Flash Load finished at 16:32:27
  
```

Figure 29: Successfully Download Message

12. Turn off the output of the DC power supply and wait until the bus capacitors are fully discharged. Then, connect the Rx, Tx, and GND pins of the FTDI USB-to-UART debugging tool to the corresponding FTDI_Rx, FTDI_Tx, and FTDI_GND interfaces on the demo board. Finally, connect the UART tool to the computer via USB.



Figure 30: UART Interface on the Demo Board

13. Launch the Allegro GUI to tune the motor control parameters. The interaction interface is shown in Figure 31.



Figure 31: Allegro GUI Waiting for Connecting Interface

14. Turn on the output of the DC power supply. Once powered on, the GUI will automatically switch to the interface shown in Figure 32.

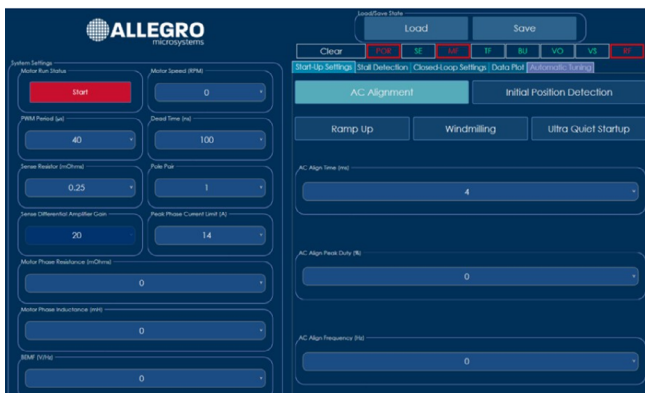


Figure 32: Allegro GUI Motor Control Tuning Interface

15. Details on how to tune motor control parameters using the Allegro GUI are not covered in this document. For comprehensive guidance, refer to the application manual titled A89211_12_24_Motor Control Quick Starter Guide. This document provides step-by-step instructions and detailed parameter configuration procedures, which are helpful for rapid system debugging and performance optimization.
16. Upon completing the above steps, the user will have successfully initiated motor operation and obtained a reasonably

optimized set of control parameters. Then, these configured parameters should be saved as an XML file. This stage marks the completion of basic system functionality debugging and establishes a stable foundation for subsequent LIN communication control and feature expansion.

```
<state>
  <MotorControlSettings>
    <ObserverProportionalGain>3</ObserverProportionalGain>
    <ObserverIntegralGain>7</ObserverIntegralGain>
    <SpeedLoopIntegralGain>7</SpeedLoopIntegralGain>
    <CurrentLoopProportionalGain>3</CurrentLoopProportionalGain>
    <CurrentLoopIntegralGain>9</CurrentLoopIntegralGain>
    <AccelerationFactor>4</AccelerationFactor>
    <DecelerationFactor>4</DecelerationFactor>
  </MotorControlSettings>
  <DataPlot>
    <PlotSelection>0</PlotSelection>
  </DataPlot>
  <StallDetection>
    <StallThresholdHigh>0</StallThresholdHigh>
    <StallThresholdLow>0</StallThresholdLow>
    <StallFilterTime>0</StallFilterTime>
    <StallTest>0</StallTest>
  </StallDetection>
  <AcAlignment>
    <PeakDuty>2</PeakDuty>
    <HoldTime>159</HoldTime>
    <Frequency>3</Frequency>
  </AcAlignment>
  <InitialPositionDetection>
    <IPDEnable>0</IPDEnable>
    <InductorSaturationPulseWidth>0</InductorSaturationPulseWidth>
    <IPDTest>0</IPDTest>
  </InitialPositionDetection>
  <RampUp>
    <RampUpTime>129</RampUpTime>
    <RampPeakFinalDuty>2</RampPeakFinalDuty>
    <RampUpEndFrequency>9</RampUpEndFrequency>
  </RampUp>
  <Windmilling>
    <WindmillVoltageFactor>0</WindmillVoltageFactor>
    <WindmillMaximumFrequency>0</WindmillMaximumFrequency>
    <WindmillMinimumFrequency>0</WindmillMinimumFrequency>
  </Windmilling>
  <SystemSetting>
    <MotorSpeed>1654</MotorSpeed>
    <MotorDirection>0</MotorDirection>
    <ModulationRatio>0</ModulationRatio>
    <VoltageCompensation>240</VoltageCompensation>
    <DeadTime>14</DeadTime>
    <PWPeriod>20</PWPeriod>
    <PolePair>4</PolePair>
    <SenseResistor>3</SenseResistor>
    <PeakPhaseCurrentLimit>11</PeakPhaseCurrentLimit>
    <SenseDifferentialAmplifierGain>1</SenseDifferentialAmplifierGain>
    <MotorPhaseResistance>23834</MotorPhaseResistance>
    <MotorPhaseInductance>70</MotorPhaseInductance>
    <Km>344</Km>
  </SystemSetting>
</state>
```

Figure 33: Allegro GUI XML File

17. The following section describes how to control the motor using the LIN bus interface tool. To begin, the LIN communication function must be enabled. Refer to the example code in Figure 34.

```

72  */
73
74
75  #include "MotorControlSdkHeaders.h"
76  #include "MotorControlHeaders.h"
77  #include "AppSupport.h"
78
79
80  void System_init(void);
81  static __attribute__( ( always_inline )
82  void SetupInterruptPriority(void);
83
84  #define LIN_COMMUNICATION
85
86  __asm( ".global __ARM_use_no_argv\n\t" );
87  int main(int argc, char **argv)
88  {
89      System_init();

```

Figure 34: Enable LIN Communication Function

18. Rescale the control parameters from the GUI configuration file (as shown in Figure 33), and manually input them into the example code, as shown in Figure 35.

```

UpdMotControlSettings.c
80
81  void UpdMotControlSettings_Initialize()
82  {
83      /*The hard code for Delta Fan test*/
84      /*Sense Amplifier Gain*/
85      MotorControlSettings_SAG = 1;
86      /* Forward Motor Direction */
87      MotorControlInterface_SetDirection(False);
88      /* Current PI Gains */
89      MotorControlInterface_SetPiCurrKi( 9 );
90      MotorControlInterface_SetPiCurrKp( 2 );
91      /* Observer PI Gains */
92      MotorControlInterface_SetPiPositionKi( 6 );
93      MotorControlInterface_SetPiPositionKp( 2 );
94      /* Speed PI Gains */
95      MotorControlInterface_SetPiSpeedKi( 5 );
96      /* Set BEMF Constant */
97      MotorControlInterface_SetBemfConstant( 33 );
98      /* Set Phase Inductance */
99      MotorControlInterface_SetPhaseInductance( 39 );
100     /* Set Phase Resistance */
101     MotorControlInterface_SetPhaseResistance( 12016 );
102     /* 25kHz PwmPeriod */
103     MotorControlInterface_SetPwmPeriod(4000);
104     /* 300ns DeadTime */
105     MotorControlInterface_SetDeadTime(64);
106     /* Ipd Setting */
107     MotorControlInterface_SetIpdEnable( False );
108
109     MotorControlInterface_SetIpdInductorSaturationPulseWidth( 3000 );
110     /* RampUp Setting */
111     MotorControlInterface_SetRampPeakDuty( 6 );
112     MotorControlInterface_SetRampEndFrequency(288);
113     MotorControlInterface_SetRampDuration(16000);
114     /* Ac Alignment Setting */
115     MotorControlInterface_SetAcAlignPeakDuty(3);
116     MotorControlInterface_SetAcAlignFrequency(16);
117     MotorControlInterface_SetAcAlignDuration(8000);
118     /* WindMil Setting */
119     MotorControlInterface_SetWindmilVoltFactor(0);
120     MotorControlInterface_SetWindMilMinFreq(0);
121     MotorControlInterface_SetWindMilMaxFreq(0);
122     /* Stall Detection Setting */
123     MotorControlInterface_SetStallDetHighThresh( 0 );
124     MotorControlInterface_SetStallDetLowThresh( 0 );
125     MotorControlInterface_SetStallDetFiltTime( 0 );
126     MotorControlInterface_SetClosedLoopSpeedCurrLim( 2800 );
127     /* Acceleration Factor */
128     MotorControlInterface_SetClosedLoopSpeedAcceleratorFactor( 145333 );
129     MotorControlInterface_SetClosedLoopSpeedDeceleratorFactor( 145333 );
130     /* Voltage Compensation */
131     MotorControlInterface_SetVoltComp(240);

```

Figure 35: Manually Input Control Parameters into Example Code

Table 4: Mapping Between the GUI Control Parameters and Example Code Control Parameters

Number	GUI Motor Control Parameters	Example Code Motor Parameters
1	SenseDifferentialAmplifierGain	MotorControlSettings_SAG = SenseDifferentialAmplifierGain;
2	MotorDirection	MotorControlInterface_SetDirection(MotorDirection);
3	CurrentLoopIntegralGain	MotorControlInterface_SetPiCurrKi(CurrentLoopIntegralGain);
4	CurrentLoopProportionalGain	MotorControlInterface_SetPiCurrKp(CurrentLoopProportionalGain);
5	ObserverIntegralGain	MotorControlInterface_SetPiPositionKi(ObserverIntegralGain);
6	ObserverProportionalGain	MotorControlInterface_SetPiPositionKp(ObserverProportionalGain);
7	SpeedLoopIntegralGain	MotorControlInterface_SetPiSpeedKi(SpeedLoopIntegralGain);
8	Km	MotorControlInterface_SetBemfConstant(Km * 4591/8345);
9	MotorPhaseInductance	MotorControlInterface_SetPhaseInductance(MotorPhaseInductance * 9030/16148);
10	MotorPhaseResistance	MotorControlInterface_SetPhaseResistance(*3442)/6827);
12	PWPeriod	MotorControlInterface_SetPwmPeriod(3200 + PWPeriod * 40);
13	DeadTime	MotorControlInterface_SetDeadTime((DeadTime + 2) * 4);
14	IPDEnable	MotorControlInterface_SetIpdEnable(IPDEnable);
15	InductorSaturationPulseWidth	MotorControlInterface_SetIpdInductorSaturationPulseWidth(1600 + (InductorSaturationPulseWidth * 480));
16	RampPeakFinalDuty	MotorControlInterface_SetRampPeakDuty(RampPeakFinalDuty * 3);
17	RampUpEndFrequency	MotorControlInterface_SetRampEndFrequency(RampUpEndFrequency * 32);
18	RampUpTime	MotorControlInterface_SetRampDuration((RampUpTime + 1) * 100);
19	PeakDuty	MotorControlInterface_SetAcAlignPeakDuty(PeakDuty * 3);
20	Frequency	MotorControlInterface_SetAcAlignFrequency(Frequency * 8);
21	HoldTime	MotorControlInterface_SetAcAlignDuration((HoldTime + 1) * 100);
22	StallThresholdHigh	MotorControlInterface_SetStallDetHighThresh(StallThresholdHigh);
23	StallThresholdLow	MotorControlInterface_SetStallDetLowThresh(StallThresholdLow);
24	StallFilterTime	MotorControlInterface_SetStallDetFiltTime(StallFilterTime * 80);
25	PeakPhaseCurrentLimit	MotorControlInterface_SetClosedLoopSpeedCurrLim((PeakPhaseCurrentLimit * 3 + 7) * 5459 / 100);
26	AccelerationFactor	MotorControlInterface_SetClosedLoopSpeedAccelatorFactor(1308000/(AccelerationFactor + 1));
27	DecelerationFactor	MotorControlInterface_SetClosedLoopSpeedDecelatorFactor(1308000/(DecelerationFactor + 1));
28	VoltageCompensation	MotorControlInterface_SetVoltComp(VoltageCompensation);

19. After completing steps 8 through 11, turn off the output of the DC power supply and wait until the bus capacitor is fully discharged. Next, connect the communication port of the LIN bus interface tool to the LIN connector on the demo board. Additionally, connect the GND port of the bus tool to the SWDGND connector on the demo board. The LIN bus interface tool used in this document is the Tongxing TC1012P.

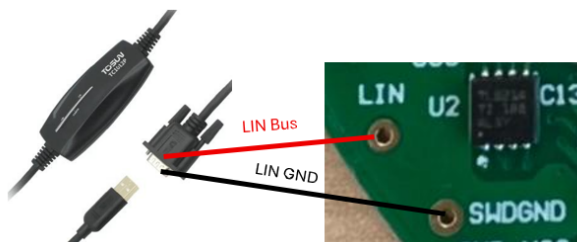


Figure 36: LIN Interface on the Demo Board

20. Launch the LIN bus design tool (this reference design uses Tongxing TSMaster), and configure the basic LIN communication parameters. The A89224 supports LIN transmission rates ranging from 1 Kbps to 20 Kbps. In this example, the selected baud rate is 20 Kbps.

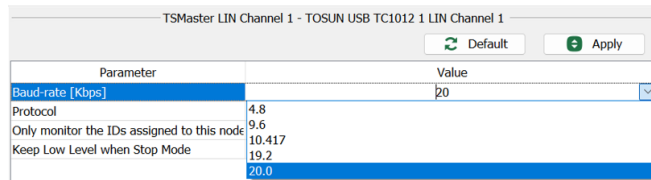


Figure 37: LIN Baud Rate Configuration

21. Next, configure the LIN bus communication protocol. In this example, LIN Protocol Version 2.1 is selected.

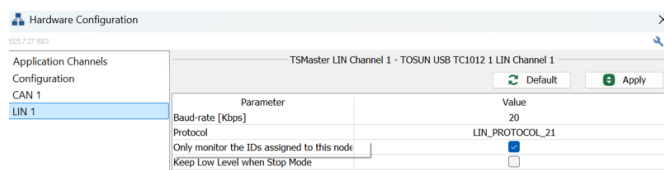


Figure 38: LIN Bus Protocol Configuration

22. In the LIN reference design example code, locate the code related to LIN frame header ID definitions within the LinCommunication.h header file, as shown in Figure 39.

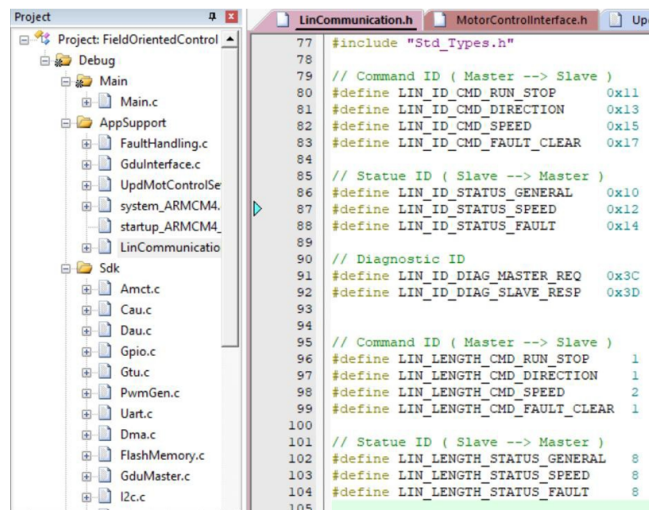


Figure 39: LIN Frame Header ID Definitions Code

Table 5: Description of LIN Frame header ID Definitions

ID	Function Description	Receive/ Transmit	Length of Data (Byte)	Comment
0x11	Run/Stop	Transmit	1	0: Stop 1: Run
0x13	Direction	Transmit	1	0: Forward 1: Reverse
0x15	Speed Reference Command	Transmit	2	In the LIN reference design example code, the speed reference command is represented as a 16-bit hexadecimal value. The high byte (upper 8 bits) is assigned from the first byte of the frame response, denoted as D0. The low byte (lower 8 bits) is assigned from the second byte of the frame response, denoted as D1.
0x17	Clear Fault Statuses	Transmit	1	0: No Action 1: Clear Fault Statuses
0x12	Current Motor Speed	Receive	8	Data structure same as speed reference command.
0x14	Reading Fault Statuses	Receive	8	D0: Power-on Reset D1: System Fault (communication, memory, watchdog, etc.) D2: Stall Fault D3: Overtemperature Fault D4: Bootstrap Undervoltage D5: VDS Fault D6: LDO Undervoltage D7: Reserved
0x3C	Diagnostic Frame, No Definition	Receive	8	Reserved
0x3D	Diagnostic Frame, No Definition	Transmit	8	Reserved

23. Based on the LIN frame header ID definitions described above, create the corresponding message frames within the LIN bus design tool. This step ensures that the communication structure aligns with the firmware implementation and enables proper data exchange between the LIN master and the A89224.

Row	Enable	Message Name	Id	Chn	Dir	DLC	D0	D1	D2	D3	D4	D5	D6	D7	Delay Time (s)	Comment
1	☐	FaultClear	17	1	Tx	1	01								10	
2	☐	Run/Stop	11	1	Tx	1	01								10	
3	☒	SpeedRef	15	1	Tx	2	04	D0							10	
4	☐	Direction	13	1	Tx	1	00								10	
5	☒	SpeedMeas	12	1	Rx	8									10	
6	☐	FaultStatus	14	1	Rx	8									10	

Figure 40: Created LIN Message Frame

24. After completing the LIN bus configuration, turn on the DC power supply for the demo board.
25. After each power cycle, it is necessary to send the “Fault-Clear” LIN message frame with a data payload of 0x01 to clear all faults. This ensures the system is ready for normal operation.
26. To start the motor, send the “Run/Stop” message frame with a data payload of 0x01. Simultaneously, send the “SpeedRef” and “Direction” message frames. The motor will begin rotating in the

specified direction and gradually accelerate to the target speed. Eventually, the motor speed will stabilize near the target speed.

27. Continuous transmission of the “SpeedMeas” message frame allows real-time reception of the current motor speed data from the A89224.
28. To obtain the current fault statuses, send the “FaultStatus” message frame.
29. To stop the motor, send the “Run/Stop” message frame with a data payload of 0x00. Once A89224 receives this command, the motor will begin to decelerate and eventually come to a complete stop.
30. The A89224 will automatically enter sleep mode, when no LIN message frames are transmitted within the configured maximum idle timeout period. In sleep mode, all internal power modules of the A89224 are disabled, enabling low-power consumption operation.
31. Once the A89224 enters sleep mode, any subsequent LIN message frame transmitted on the bus will automatically trigger the LIN transceiver to wake up. In rapid sequence, the A89224 will be immediately awakened. All its internal power modules will resume output. Then, the firmware will re-enter the power-on initialization sequence.

TEST WAVEFORM

Test Condition

- LIN Master Device: Tongxing TC1012P
- LIN Slave Device: Allegro A89224 LIN Reference Design Demo Board
- LIN Bus Design Tool: Tongxing TSMaster
- Motor Control Firmware: Allegro LIN Reference Design Example Code
- Prerequisites: Motor control parameters have been successfully tuned and imported into the example code. The LIN message frames have also been created.

LIN Transmit														
Settings		CH1/Unknown/Node [Master]			Mode: Schedule Table		Single Frame		Deploy					
Row	Enable	Message Name	Id	Chn	Dir	DLC	D0	D1	D2	D3	D4	D5	D6	D7
1	☐	FaultClear	17	1	Tx	1	01							10
2	☐	Run/Stop	11	1	Tx	1	01							10
3	☒	SpeedRef	15	1	Tx	2	04	D0						10
4	☐	Direction	13	1	Tx	1	00							10
5	☒	SpeedMeas	12	1	Rx	8								10
6	☐	FaultStatus	14	1	Rx	8								10

Figure 41: Created LIN Message Frames in TSMaster

Waveform

After the demo board is powered on again, the LIN master transmits a frame with ID 0x14 named “FaultStatus”, which allows reading the current fault statuses, as shown in Figure 42.

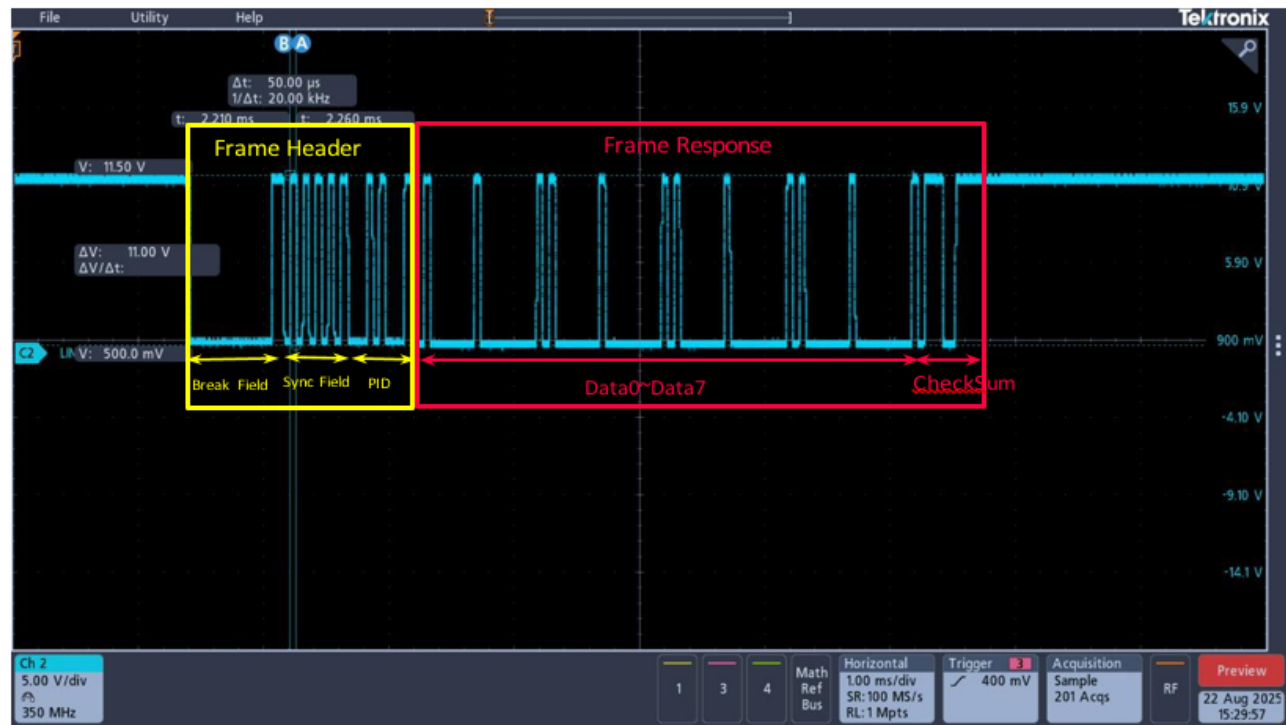


Figure 42: LIN Message Frame Structure (Frame Header and Frame Response) at 20 Kbps Baud Rate

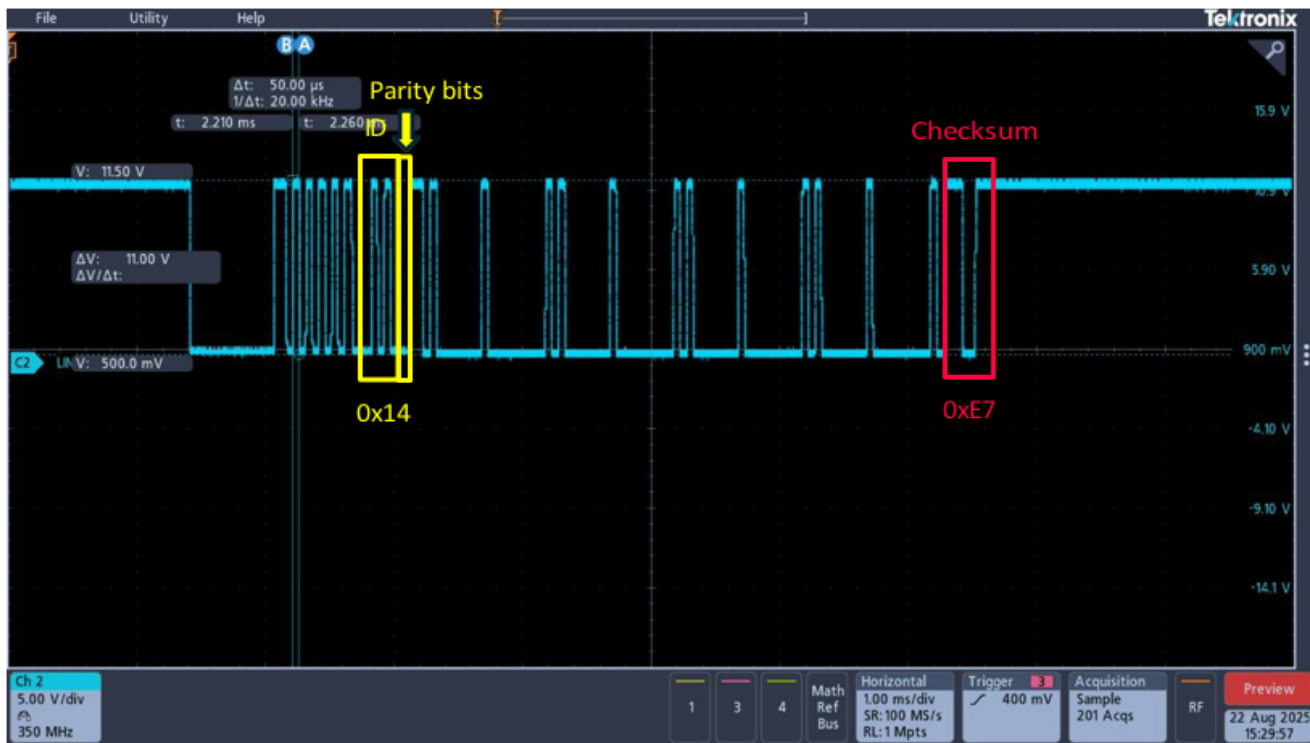


Figure 43: Received Frame Response on LIN Bus During Fault Conditions

Open the LIN Trace window in TSMaster to view the received fault statuses data, as shown in Figure 44.

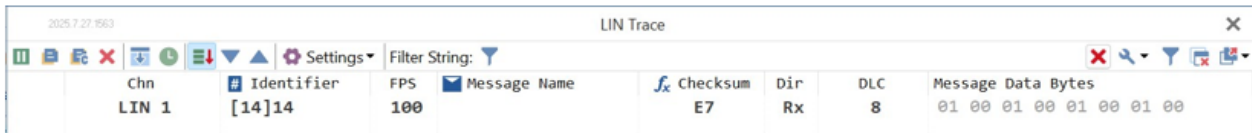


Figure 44: View Received Fault Statuses Data in LIN Trace Window

Table 6: Description of the Fault Codes

Datax	Fault	Description
Data0	POR	Power-on reset
Data1	SE	System error (communication issue, memory issue, watchdog, etc.)
Data2	MF	Motor fault (stall condition)
Data3	TF	Overtemperature fault
Data4	BU	Bootstrap undervoltage
Data5	VO	Overcurrent fault
Data6	RF	Regulator fault
Data7	—	

The LIN master transmits a frame with ID 0x17 named “FaultClear” (data payload 0x01) to clear the current fault statuses.

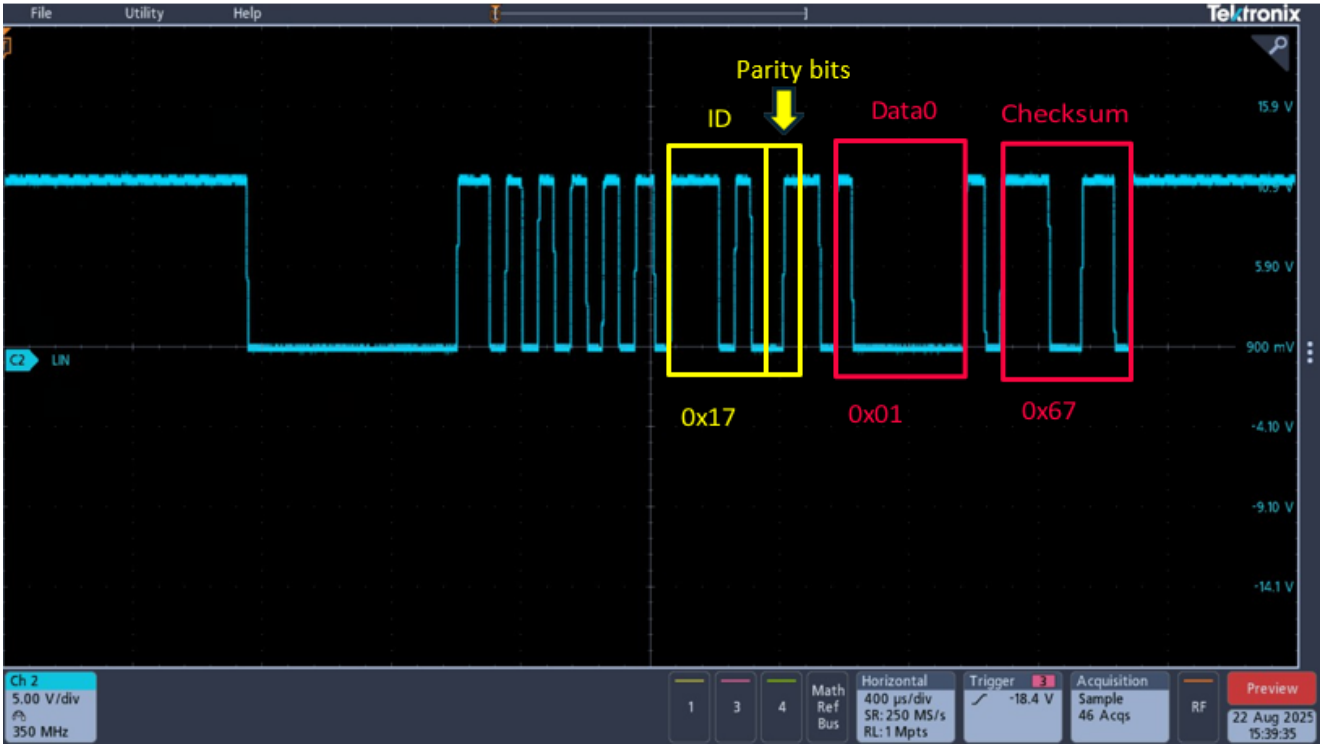


Figure 45: LIN Frame: Fault Clear

The transmitted “FaultClear” frame data can be viewed in the LIN Trace window, as shown in Figure 46.

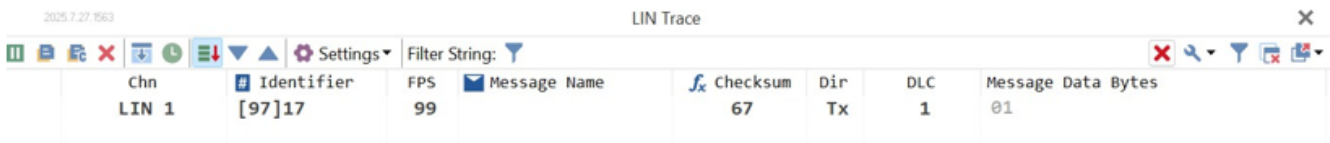


Figure 46: Transmitted “FaultClear” Frame Data

If the “FaultStatus” frame is transmitted again at this point, the frame response will confirm that all fault status have been successfully cleared.

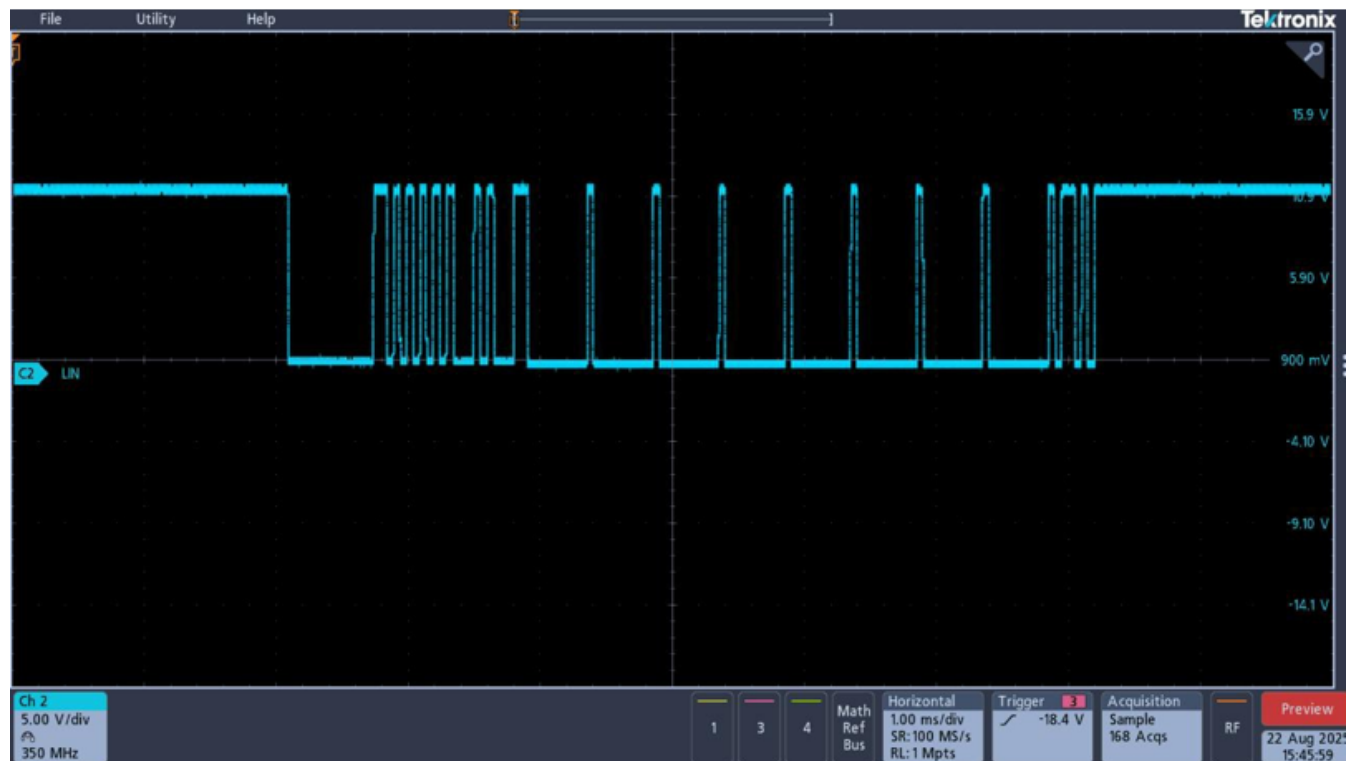


Figure 47: Received Frame Response on LIN Bus without Fault Conditions

All fault status data displayed in the LIN Trace window have been reset to zero, indicating that the fault status have been successfully cleared and the system has resumed normal operation.

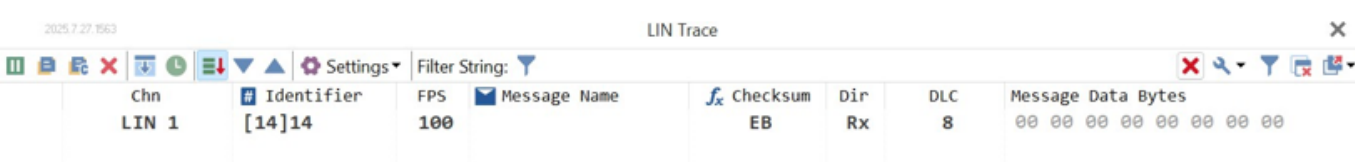


Figure 48: Received Fault Status Data without Fault Conditions

The LIN master transmits a frame with ID 0x11 named “Run/Stop” (data payload 0x01), when the system is in a non-fault idle state, indicating a start command. Upon receiving this command, the A89224 controller will start up the motor and transit into normal operating mode.

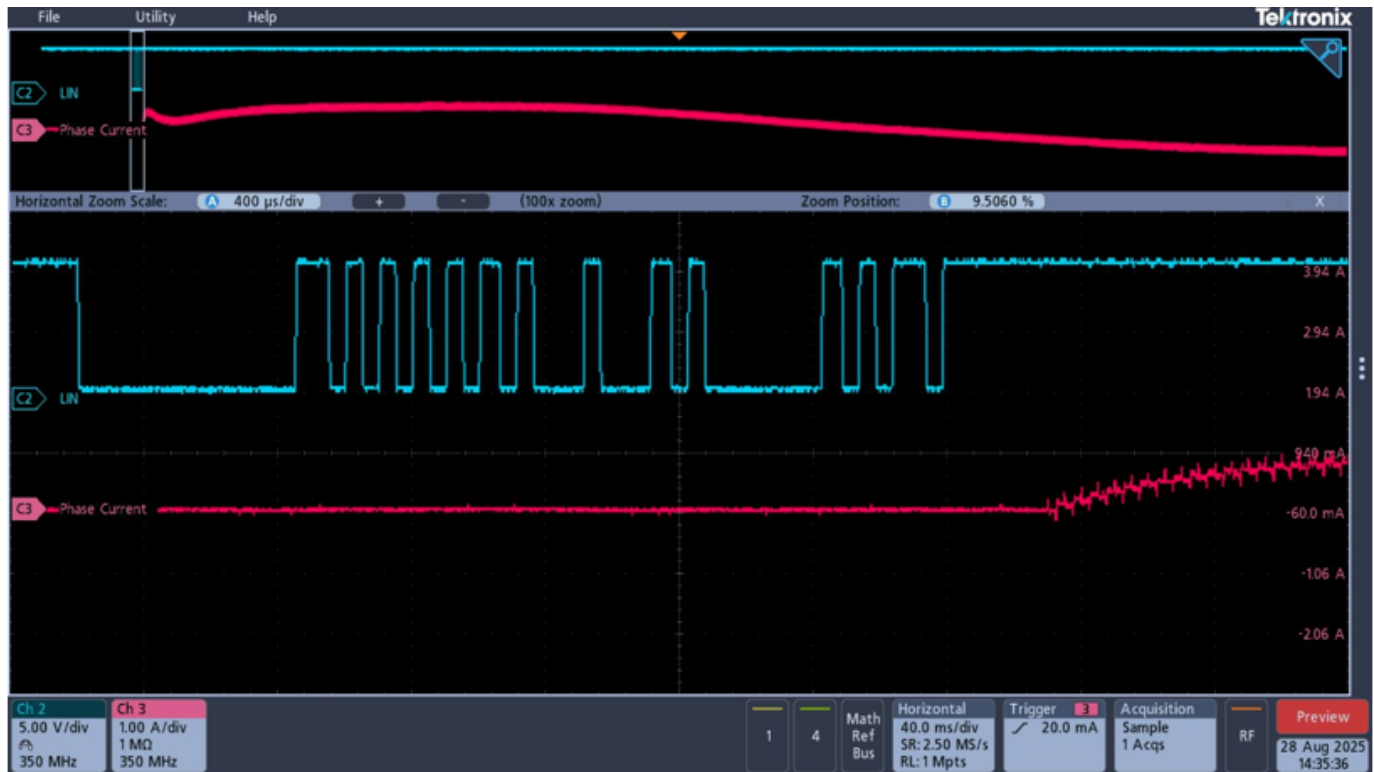


Figure 49: Motor Start-up Process via LIN Communication

Figure 49 illustrates the motor start-up process controlled via LIN communication.

In the figure, the blue waveform represents the LIN bus message frames, while the red waveform corresponds to the phase current of the motor.

The LIN master transmits a frame with ID 0x11 named “Run/Stop” (data payload 0x00), when the system is in normal operating mode, indicating a stop command. Upon receiving this command, the A89224 controller will decelerate the motor until it comes to a complete stop.

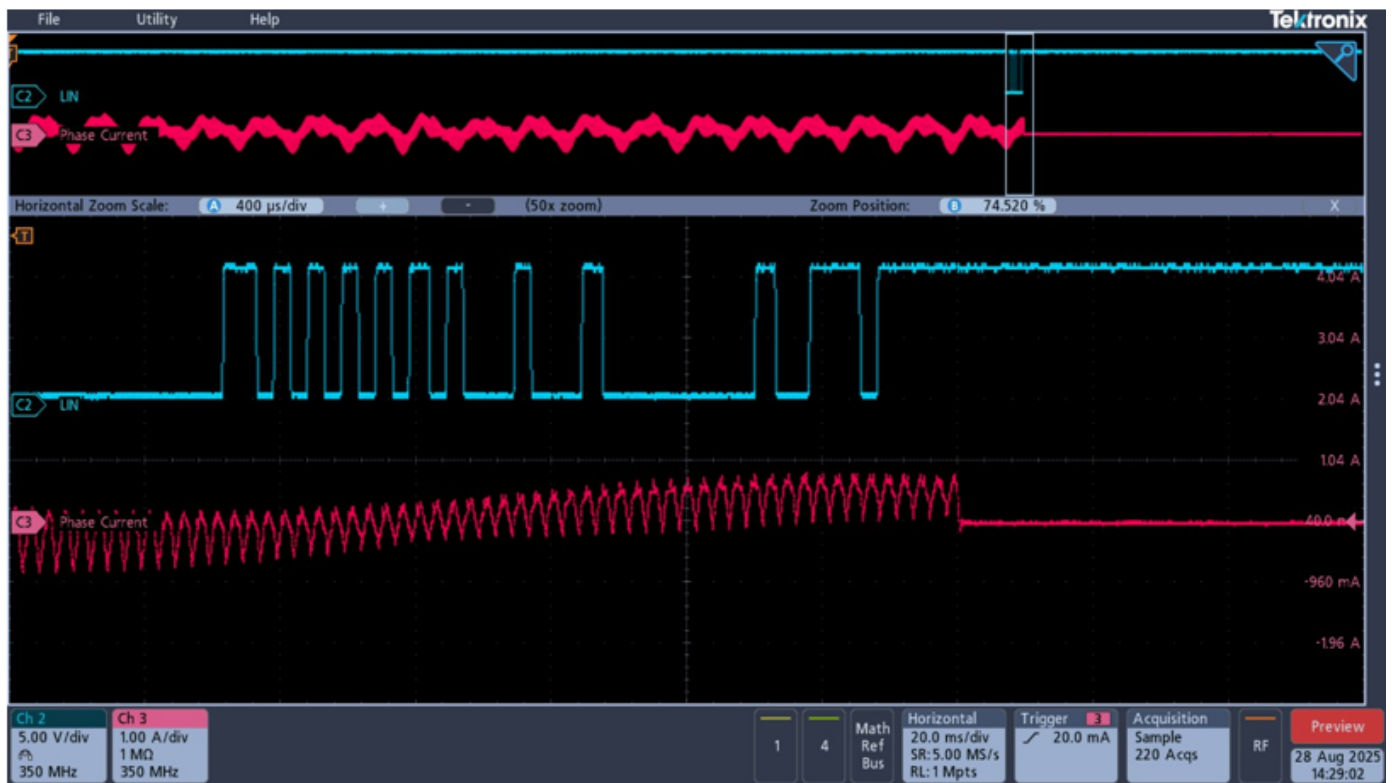


Figure 50: Motor Stop Process via LIN Communication

Figure 50 illustrates the motor stop process controlled via LIN communication. In the figure, the blue waveform represents the LIN bus message frames, while the red waveform corresponds to the phase current of the motor.

When the LIN master simultaneously transmits the following three LIN message frames:

1. A “Run/Stop” frame with ID 0x11 and a data payload of 0x01.
2. A “SpeedRef” frame with ID 0x15 and a data payload of 0x04 and 0xD0.
3. A “SpeedMeas” frame with ID 0x12.

The A89224 controller will start up the motor and gradually accelerate to the target speed. Eventually, the motor speed stabilizes near the configured value of 0x04D0. At this stage, opening the LIN Trace window reveals three key data sets:

1. Real-time feedback of the motor speed.
2. Run/Stop command value.
3. Current target speed reference.

LIN Trace							
Chn	Identifier	FPS	Message Name	Checksum	Dir	DLC	Message Data Bytes
LIN 1	[92]12	33		94	Rx	8	04 D4 00 00 00 00 00 00
LIN 1	[11]11	33		ED	Tx	1	01
LIN 1	[55]15	33		D5	Tx	2	04 D0

Figure 51: Spin Motor to Target Speed via LIN Communication

If no message frames are transmitted on the LIN bus within the predefined maximum idle timeout (10 seconds in this example), the A89224 controller automatically transitions into sleep mode. In this state, all internal LDO power modules—including VLR, VREG, and VD33—are disabled, enabling the system to enter a low-power consumption operation.

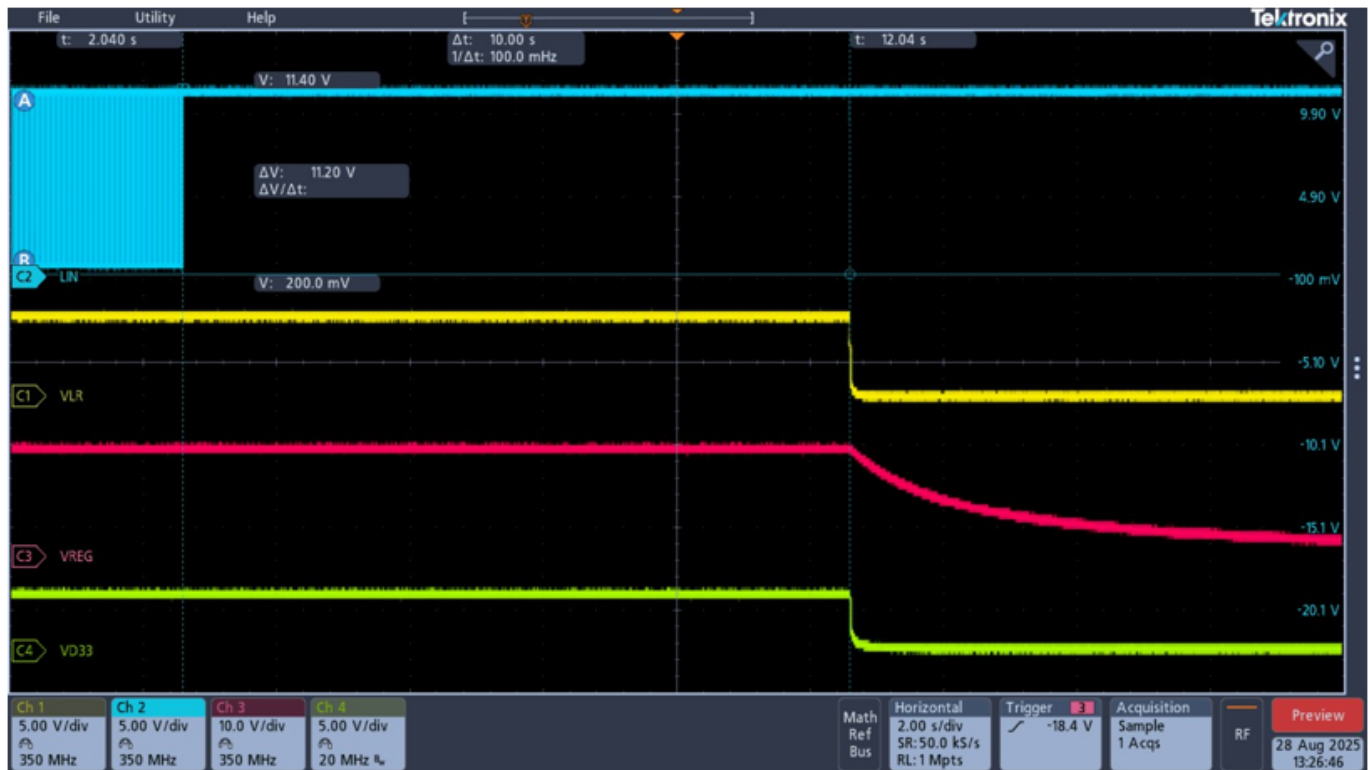


Figure 52: Triggering A89224 Sleep Mode via LIN Bus Idle State

As shown in Figure 52, the blue waveform represents the LIN bus message frame, the yellow waveform represents VLR, the red waveform represents VREG, and the green waveform represents VD33.

When the A89224 controller enters sleep mode, all internal power modules will be shut down. As a result, the GPIO1 pin—used to control the ENABLE pin of the LIN transceiver—is automatically pulled low due to power-off conditions. This causes the LIN transceiver to immediately enter sleep mode as well.

At this point, the INH (Inhibit) pin output of the LIN transceiver transitions from high level to low level, indicating that the entire system has entered a low-power state.



Figure 53: Triggering LIN Transceiver Sleep Mode via LIN Bus Idle

As shown in Figure 533, the blue waveform represents the LIN bus message frame, the yellow waveform represents VLR, the red waveform represents INH (Inhibit) pin output of the LIN transceiver, and the green waveform represents VD33.

When the LIN bus transitions from an idle state and begins transmitting message frames again, the LIN transceiver automatically wakes up from sleep mode. At this moment, its output of INH (Inhibit) pin switches from low to high level.

Since the INH (Inhibit) pin of the LIN transceiver is connected to the WAKE pin of the A89224, the A89224 is immediately awakened when the output of INH pin transitions to high level. Upon wake-up, all internal LDO power modules of the A89224—such as VLR, VREG, and VD33—resume output, and the system returns to normal operating mode.

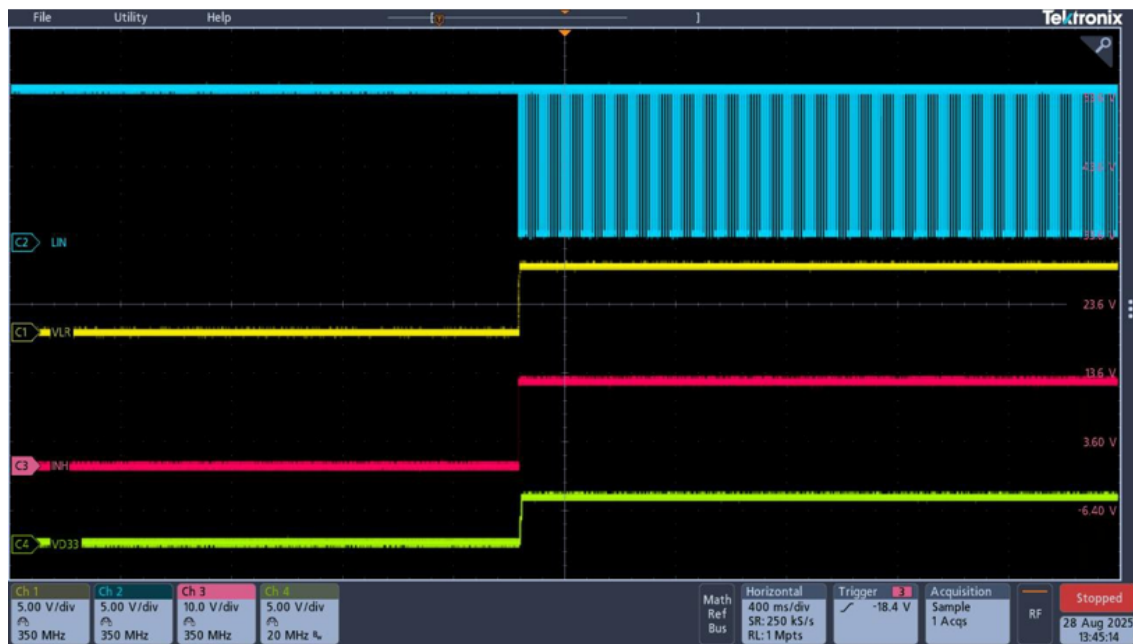


Figure 54: LIN Bus Wake-Up LIN Transceiver and A89224 Controller

As shown in Figure 54, the blue waveform represents the LIN bus message frame, the yellow waveform represents VLR, the red waveform represents the INH (Inhibit) pin output of the LIN transceiver, and the green waveform represents VD33.

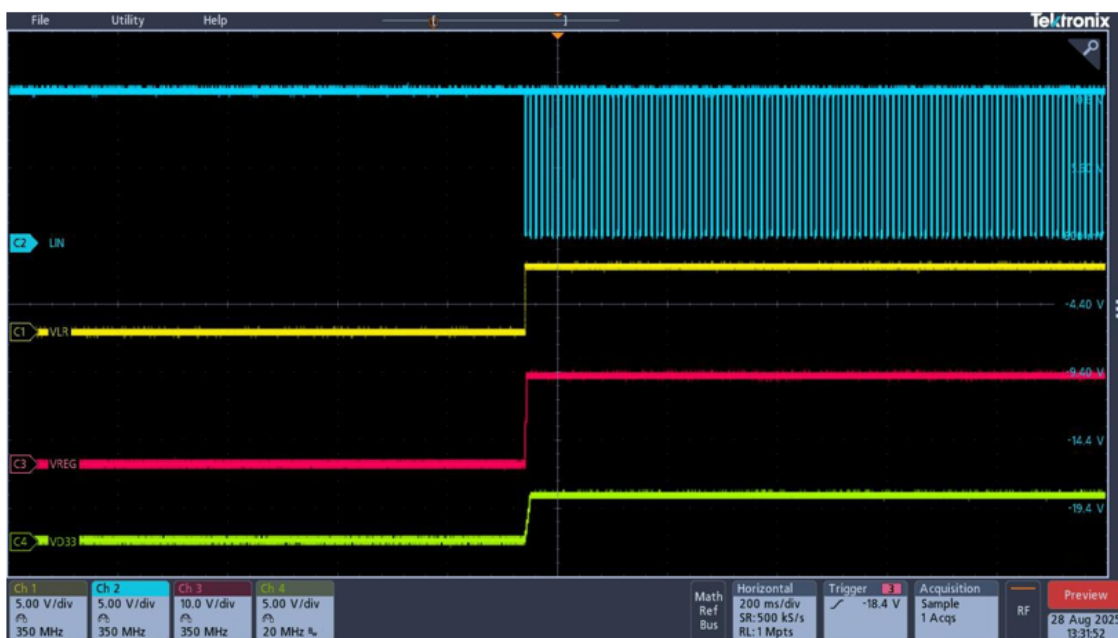


Figure 55: Internal Power Module Recovery After A89224 Wake-Up

As shown in Figure 55, the blue waveform represents the LIN bus message frame, the yellow waveform represents VLR, the red waveform represents VREG, and the green waveform represents VD33.

Revision History

Number	Date	Description
–	February 2, 2026	Initial release

Copyright 2026, Allegro MicroSystems.

Allegro MicroSystems reserves the right to make, from time to time, such departures from the detail specifications as may be required to permit improvements in the performance, reliability, or manufacturability of its products. Before placing an order, the user is cautioned to verify that the information being relied upon is current.

Allegro's products are not to be used in any devices or systems, including but not limited to life support devices or systems, in which a failure of Allegro's product can reasonably be expected to cause bodily harm.

The information included herein is believed to be accurate and reliable. However, Allegro MicroSystems assumes no responsibility for its use; nor for any infringement of patents or other rights of third parties which may result from its use.

Copies of this document are considered uncontrolled documents.

For the latest version of this document, visit our website:

www.allegromicro.com