AN205253

F^2MC-8L/16LX/FR Family with LIN-USART

This application note describes how to use the Cypress's LIN USART to control LIN devices and the basics of the LIN bus protocol.

Contents

1 Introduction

This application note describes how to use the Cypress's LIN USART to control LIN devices.

The software example is based on a 16-bit microcontroller of the MB90350 Series[1], acting as a LIN master that will be connected to a Stepper Motor Driver IC from the company
AMI Semiconductor. The AMIS-30621 is a bipolar 2-Phase stepper motor driver with position controller and LIN control/diagnostics interface integrated in a single chip. The AMIS-30621 acts as a slave on the bus and the master can fetch specific status information like actual position, error flags, etc. from each individual slave node.

The basics of the LIN bus protocol will be explained within this application note.

2 LIN-Bus

Short Specification

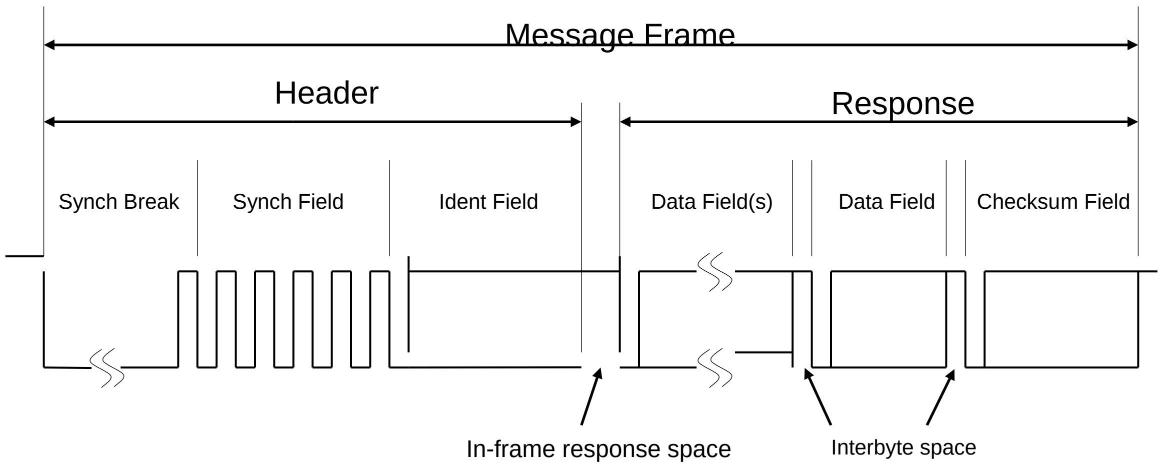
[1] The software example can easily be adapted to all 8-, 16- and 32-bit MCUs with LIN-USART.

2.1 Short LIN Specification

LIN uses NRZ-8N1L data format in a baud rate range from 9600 Bit/s to 19200 Bit/s. A LIN bus is a 1 master to n slaves bus.

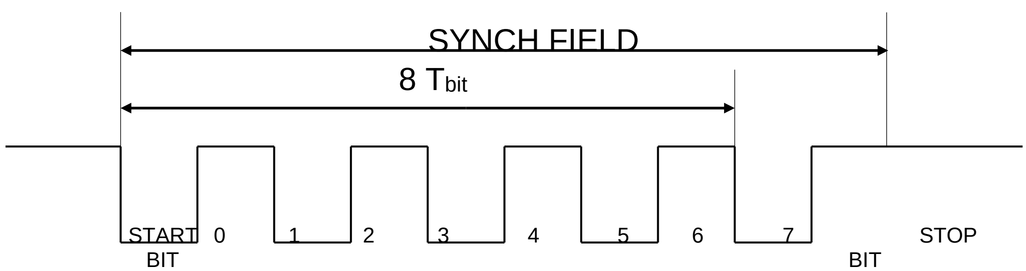
A LIN message frame consists of a header and a response like in the graphic below:

LIN Message Frame



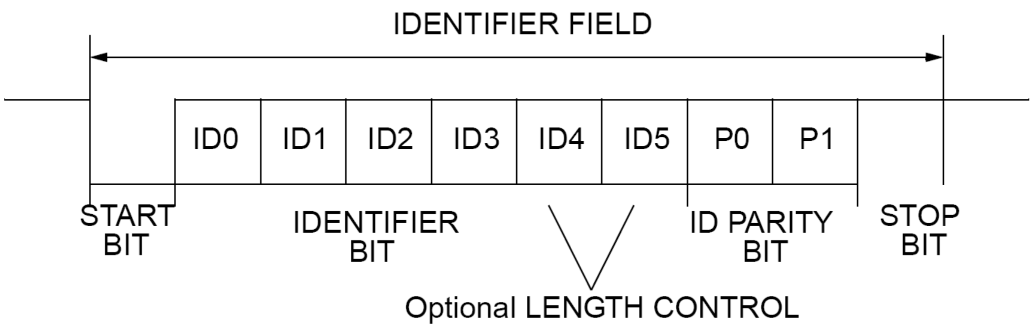
Except for the Synchronization Break all Fields are simple 8N1L data, this means 1 start bit, 8 data bits (LSB first), no parity, 1 stop bit.

The Synchronization Field is a simple 0x55 byte (LSB first). Thus it consists of alternately 5 dominant and 5 recessive bits:



2.1.1 Identifier Field

The Identifier filed consists of 6 ID bits and 2 parity bits:



The Parity bits are calculated by:

$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4 \qquad P1 = \overline{ID1 \oplus ID3 \oplus ID4 \oplus ID5}$$

The identifiers **0x3C**, **0x3D**, **0x3E**, and **0x3F** with their respective IDENTIFIER FIELDS **0x3C**, **0x7D**, **0xFE**, and **0xBF** (all 8-byte messages) are reserved for command frames (e.g. sleep mode) and extended frames

The identifier **0x3C** is a Master Request-frame to send commands and data from the master to the slave node.

The identifier **0x3D** is an Slave Response frame that triggers one slave node (being addressed by a prior download-frame) to send data to the master node.

2.1.2 Checksum

The checksum is calculated over all data bytes (LIN 1.3) or over all data bytes and the identifier byte (LIN 2.0). The calculation is the inverted sum with carry. This means, if the new sum over the last sum is greater than 255 an additional “1” is added. After the last addition, the result is inverted.

The formula for this calculation is:

$$Checksum = 0xFF \oplus \left(\left(\sum_{i=1}^{n} data_i \right) \bmod 0x100 + \left\lfloor \frac{\sum_{i=1}^{n} data_i}{0x100} \right\rfloor \right)$$

Note, that in LIN 2.0 $data_i$ also contain the identifier field.

For more Information about Using the LIN-Bus check our Application Note **mcu-an-390088-e-uart_lin.pdf**.

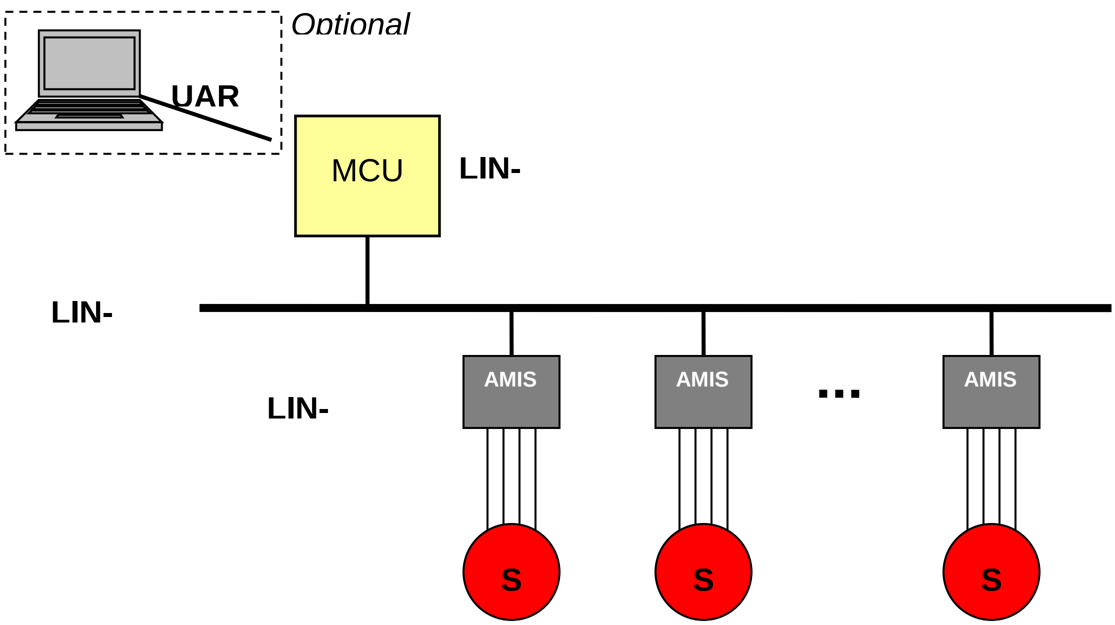
3 Hardware Setup

This Chapter describes a minimal Hardware Setup

3.1 Block diagram

This diagram shows a simple configuration of the functional blocks.

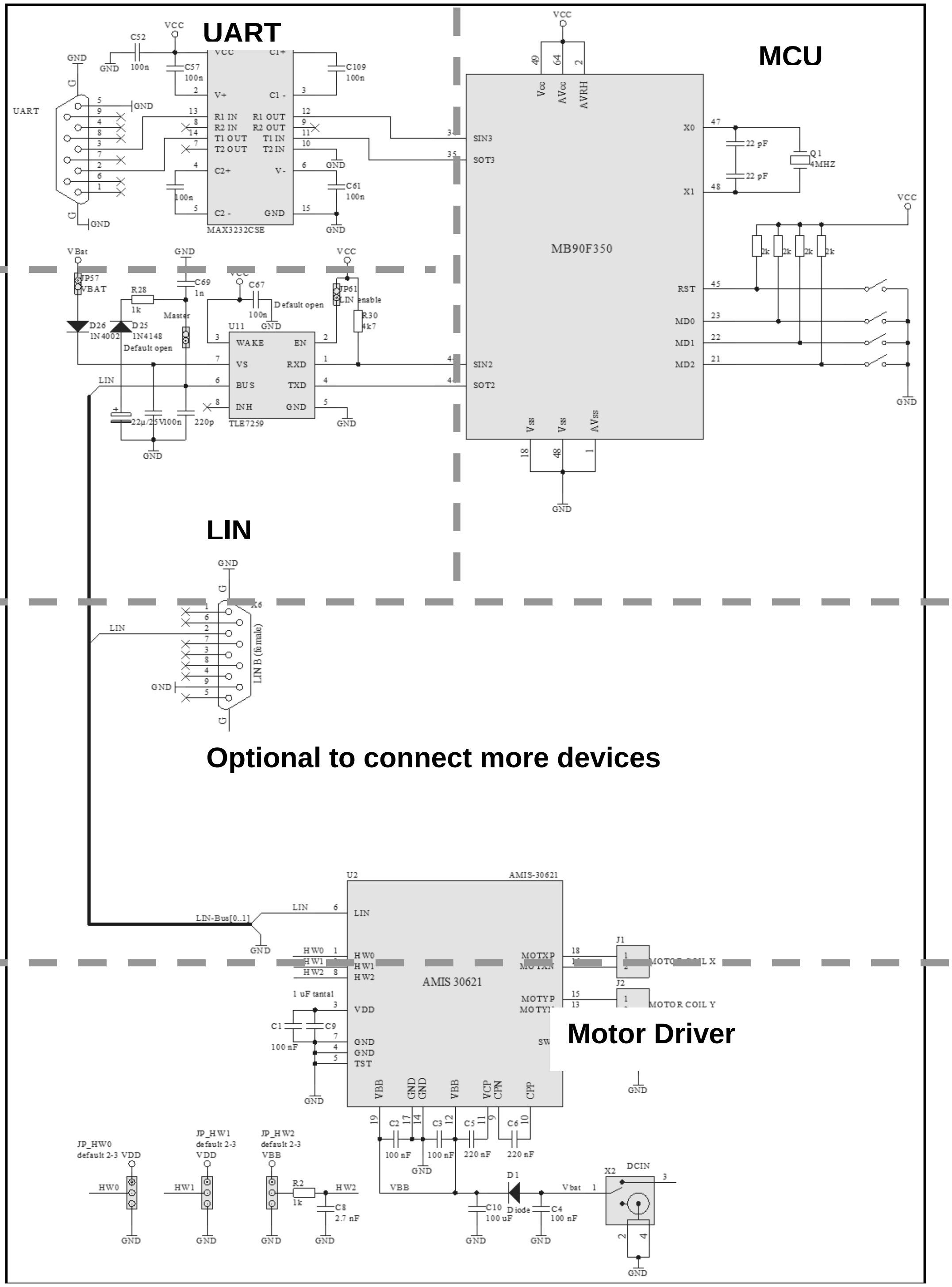
Figure 1. Block diagram



3.2 Schematics

The Schematic shows the MCU Part including UART- and LIN-Transceivers and the Part of the AMIS Motor Driver. In each case a basic setup is presented.

Figure 2. Schematic



3.3 Short Description AMIS – 30621

The AMIS – 30621 LIN Microstepping Motordriver is able to drive a bipolar stepper motor with up to 800 mA. The used motors need to have low impedance. A minimal current of 59 mA have to be drawn by the motor, otherwise the AMIS driver detects an open loop and switches off the output.

The motor is current controlled by a 20 kHz PWM and it can be chosen between several stepping modes. (Half Stepping to 1/16 micro stepping)

It supports LIN rev. 1.3 with 19.2 kbaud and acts as a slave on the bus. 128 node addresses can be used. The address is set by programming 5 bits of the address internal and setting 3 hardwired bits external (HW0 to HW2).

It features protection functions and several positioning and driving configurations.

The AMIS - 30621 needs an 8V to 29V supply voltage. An internal 5V regulator produces the voltage for the control logic. No further power supply is needed.

3.4 Cypress's MCU

The MCU controls the Stepper motor driver over the LIN-Bus. It acts as Master. In this example a 16bit MCU the MB90F352 is used, but you can use any controller that has at least one LIN_UART Interface.

To connect the MCU physically to the Bus Line, you need a LIN Transceiver. Here a TLE7259 is used.

The PC Communication is realized via UART.

4 Controlling the AMIS 30621

This Chapter describes some basic commands of the AMIS 30621

4.1 LIN Frames

The AMIS 30621 uses 8 Types of LIN frames that are subdivided into Writing-, Reading-, and Preparing-Frames.

- A writing frame is sent by the LIN Master to send commands and/or information to the Slave nodes.

In the example type 4 is used for writing frames. It starts with **0x3C** Identifier followed by a Command Indicator, the Command itself, the physical address and parameters.

Figure 3. Writing frame type 4

ID		Data1	Data2		Data3		Data4	Data5	Data6	Data7	Data8
0x3C	00	0x80	CMD[6:0]	1	AD[6:0]	B					
		AppCmd	command		physical address				parameters		

AppCMD : 0x80 indicates that Data2 contains an application command byte

CMD[6:0]: Command byte **AD[6:0]** : Slave node's physical address

- A preparing frame is a writing frame that warns a particular slave node that it will have to answer in the next frame (hence a reading frame).
- A reading frame uses an in-frame response mechanism. That is: the master initiates the frame (synchronization field + identifier field), and one slave sends back the data field together with the check field.

Figure 4. Preparing frame type 8

ID		Data1	Data2		Data3		Data4	Data5	Data6	Data7	Data8
0x3C	00	0x80	CMD[6:0]	1	AD[6:0]	B	0xFF	0xFF	0xFF	0xFF	0xFF
		AppCmd	command		physical address		0x80				

4.1.1 Dynamically assigned identifier

The above mentioned frames use the identifiers **0x3C** (writing) and **0x3F** (reading). Apart from these identifiers the LIN specification does not indicate how identifier can be allocated. To keep slave nodes adaptable to a given LIN Network, the idea is to implement a dynamic assignment of the identifier by the LIN Master. This is done at start-up by writing identifier and the desired corresponding command in the slave's RAM.

Figure 5. Dynamic ID assignment

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
0	0	1	1	1	1	0	0	Identifier
AppCmd = 0x80								Data1
1	CMD[6:0] = 0x11							Data2
Broad	AD6	AD5	AD4	AD3	AD2	AD1	AD0	Data3
ID1[3:0]				ROMp1[3:0]				Data4
ID2[1:0]		ROMp2[3:0]				ID1[5:4]		Data5
ROMp3[3:0]				ID2[5:2]				Data6
ROMp4[1:0]		ID3[5:0]						Data7
ID4[5:0]						ROMp4[3:2]		Data8

The LIN frame shown above uses the **0x3C** identifier and links four dynamic IDs to four ROM pointers that represent the actual command. The AMIS 30621 has nine ROM pointers, so nine commands can be used via dynamic IDs.

Command Mnemonic	Command Byte (CMD)		Dynamic ID (example)	ROM Pointer
`GetActualPos`	000000	`0x00`	100xxx	0010
`GetFullStatus`	000001	`0x01`	n.a.	
`GetOTPparam`	000010	`0x02`	n.a.	
`GetStatus`	000011	`0x03`	000xxx	0011
`GotoSecurePosition`	000100	`0x04`	n.a.	
`HardStop`	000101	`0x05`	n.a.	
`ResetPosition`	000110	`0x06`	n.a.	
`ResetToDefault`	000111	`0x07`	n.a.	
`RunInit`	001000	`0x08`	n.a.	
`SetMotorParam`	001001	`0x09`	n.a.	
`SetOTPparam`	010000	`0x10`	n.a.	
`SetPosition (16-bit)`	001011	`0x0B`	010xxx	0100
`SetPositionShort (1 motor)`	001100	`0x0C`	001001	0101
`SetPositionShort (2 motors)`	001101	`0x0D`	101001	0110
`SetPositionShort (4 motors)`	001110	`0x0E`	111001	0111
`Sleep`	n.a.	n.a.		
`SoftStop`	001111	`0x0F`	n.a.	
Dynamic ID assignment	010001	`0x11`	n.a.	
General purpose 2 Data bytes			011000	0000
General purpose 4 Data bytes			101000	0001
Preparation frame			011010	1000

xxx allows to address physically a slave node. Therefore, these dynamic IDs cannot be used for more than 8 stepper motors.

Only 9 ROM pointers are needed for the AMIS-30621.

So for example if you want to perform a **GetStatus** Command you have to set ID1[5:0] = 000000 and ROMp1[3:0] = 0011. Same commands are only available via dynamic IDs. Most of them use **General purpose 2 Data bytes**. So again you have to link the ID with the ROM pointer (e.g. ID4[5:0] = 011000 and ROMp4[3:0] = 0000) and then the command is sent as part of the LIN frame.

A HardStop command then looks like this:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
* = 1	* = 1	0	1	1	0	0	0	Identifier
1	CMD[6:0] = 0x05							Data1
Broad	AD[6:0]							Data2

The bits 6 and 7 are set to one according to parity computation.

4.2 Dummy Frame

According to LIN Spec Rev 1.3 the Slave assumes a bus sleep state if there is no activity on the bus for 25000 T_{bit} (1.3 s @19.2 kbaud). In sleep mode the AMIS 30621 drives the motor to the secure position and goes to Shutdown Mode. To leave the Shutdown mode you have to send a “GetFullStatus” – Command. To prevent this you have to send a dummy frame that keeps the driver awake.

In the example a reload timer is used to trigger a LIN message every 400 ms.

To use this needed traffic, the periodic message is used for a GetStatus Command. The Slave (AMIS device) answers with some status flags, like the state of the external Switch Input (ESW).

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
ESW	AD[6:0]							Data1
VddReset	StepLoss	ElDef	UV2	TSD	TW	Tinfo[1:0]		Data2

4.3 GetFullStatus

This command is provided to the circuit by the LIN Master to get a complete status of the circuit and of the Stepper motor. It corresponds to one preparing frame and two successive LIN in-frame responses with **0x3D** indirect ID.

The response frames contain the actual status and parameters of the motor driver according to the following figure:

Figure 6. Status Response Frames

In-frame Response 1 (type #6)								
0	1	1	1	1	1	0	1	Identifier
1	AD[6:0]							Data1
Irun[3:0]				Ihold[3:0]				Data2
Vmax[3:0]				Vmin[3:0]				Data3
AccShape	StepMode[1:0]		Shaft	Acc[3:0]				Data4
VddReset	StepLoss	ElDef	UV2	TSD	TW	Tinfo[1:0]		Data5
Motion[2:0]			ESW	OVC1	OVC2	1	CPFail	Data6
Lin error s r (see 9.2.4.4)								Data7
0xFF								Data8
In-frame Response 2 (type #6)								
0	1	1	1	1	1	0	1	Identifier
1	AD[6:0]							Data1
ActPos[15:8]								Data2
ActPos[7:0]								Data3
TagPos[15:8]								Data4
TagPos[7:0]								Data5
SecPos[7:0]								Data6
1	1	1	1	1	SecPos[10:8]			Data7
0xFF								Data8

The parameters and flags are explained in the AMIS-30621 Datasheet

4.4 SetMotorParam

This command is used to set the values for the Stepper motor parameters. It corresponds to a LIN writing frame type 4.

The most important parameters are:

- `Irun [3:0]` Peak operating current
- `Ihold [3:0]` Hold current
- `Vmax [3:0]` Maximum velocity
- `Vmin [3:0]` Minimum velocity
- `Shaft` Direction of movement for positive velocity
- `StepMode [1:0]` Stepping mode (Half stepping, ¼ micro step, 1/8 micro step, 1/16 micro step)

The `Irun` and `Ihold` parameter have to be set according to the used stepper motor. By `Irun` the highest current is defined, that is allowed to drive the motor in case of maximum load. The `Ihold` parameter defines the motor current in stop mode. Note that if this value cannot be reached, because the impedance of the motor is too high, or the voltage too low, the AMIS 30621 interprets this as an open loop and switches off the output.

The current values can be set to 16 different levels that go from 59 mA to 800 mA.

To set the parameters to the LIN frame format required by the motor driver, the `SetMotorParameter()` function is used. See section 5.3.2

4.5 SetPosition

This command is provided to the circuit by the LIN Master to drive one or two motors to a given absolute position. The used frame type is of type 4 and contains the address and the new position of one or two motors.

Figure 7. SetPosition LIN frame

ID		Data1	Data2		Data3		Data4	Data5	Data6		Data7	Data8
0x3C	00	0x80	CMD[6:0]	1	AD1[6:0]	B	Pos1[15:8]	Pos1[7:0]	AD2[6:0]	B	Pos2[15:8]	Pos2[7:0]
		AppCmd	SetPosition command 0x0B		physical address		new Position Motor1		physical address 2		new Position Motor2	

For example and according to figure Figure 7 a message containing the following bytes:

(0x3C | 0x80 | 0x8B | 0x80 | 0x01 | 0x00 | 0x81 | 0x00 | 0x00)

drives the Motor at address 0 to Position 0x0100 and if connected a motor at address 1 to position 0x0000.

4.6 Summary

This chapter showed the principles of controlling the AMIS 30621. The format of the LIN frames depends on the task and the used command.

To get a first movement of the stepper motor you have to follow three steps:

1. Wake up the device via **GetFullStatus**
2. Set the appropriate motor values with **SetMotorParam**
3. Set a new Position different from zero with **SetPosition**
4. (Keep the device awake with periodic frames)

For further ways of programming the AMIS 30621 and a complete command list check the AMIS Datasheet.

5 Software

This Chapter describes the Software Principles and its Implementation

The complete software source code can be found in the zip file mcu-an-300037-e-v11-lin_stepper_amis.zip. This chapter describes only the most important parts.

5.1 LIN Protocol Handling

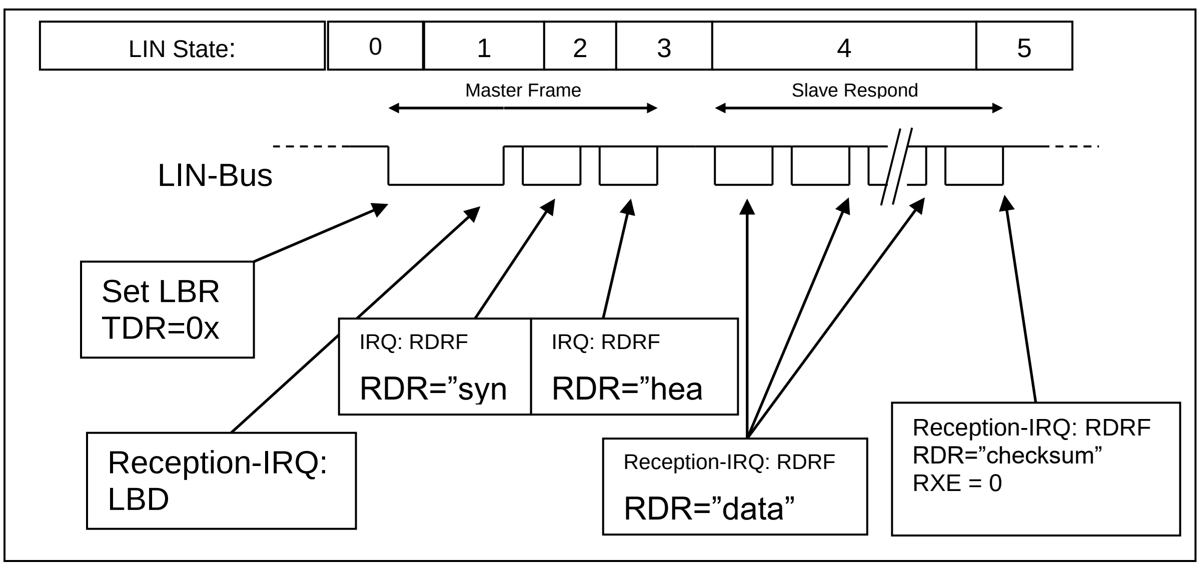
The communication according to the LIN Protocol is realized by one Interrupt Service Routine. This Routine is called when a LIN Break is detected via Master Read back. The same Routine is used for sending and receiving LIN messages. It is subdivided into 6 states:

1. Bus is idle. Ready to send synch break.
2. After a LIN break is detected, the flag is cleared, the Synch field is sent and the Reception is enabled.
3. With the next interrupt the Synch field is read back and checked. Then the LIN Header (Identifier Field) is sent.
4. Again the LIN Header is read back and checked. If the master wants to send, the first data Byte is sent and added to the Checksum Calculation.
5. In State 4 the Master sends the 2nd to last data byte or receives data bytes from the slave. If the last byte is sent or received, the checksum is sent or checked.
6. Read back of the Checksum.

All interrupts handled by the LIN ISR are reception Interrupts.

5.1.1 Interrupts during LIN frame

The following illustration gives an overview about the interrupt effort when LIN-UART is bus master:



5.1.2 State chart of the Interrupt Service Routine

Figure 8. ISR State chart

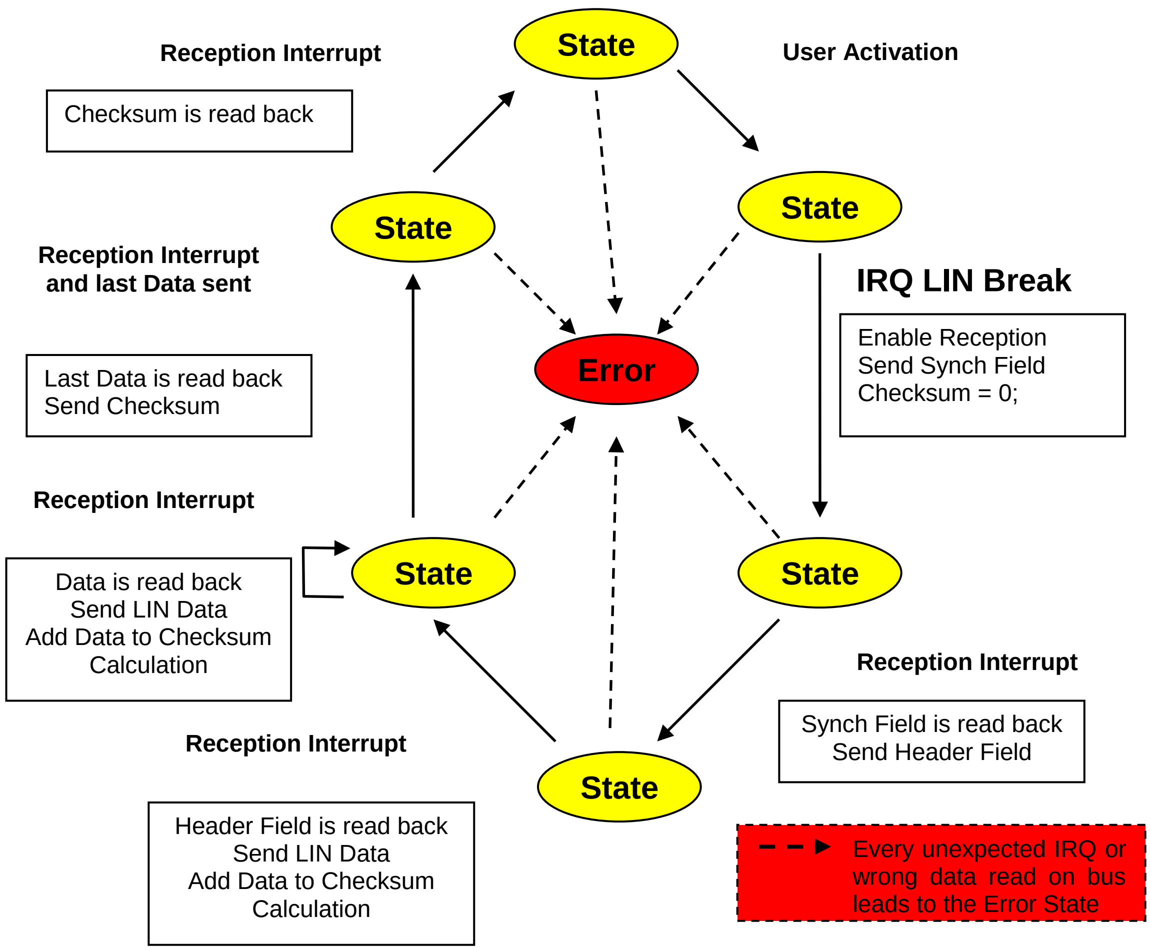


Figure 9. ISR State chart

```
/*--------------------- INTERRUPT SERVICE ROUTINE ---------------------------*/

__interrupt void RxIRQHandler(void) {
if (ESCR2_LBD) {                                  // LIN Break Detection (Read back)?
       ESCR2_LBD = 0;                             // Clear flag

       if (LIN_State == 1) {
              SCR2_RXE = 1;                       // Enable reception
              TDR2 = 0x55;                        // Send Synchfield
              LIN_State = 2;
              LIN_Checksum = 0;
       }
       else    Rx_Error = 1;                      // Unexpected reception of break
}
else if (SSR2_RDRF) {                             // Reception?
       Rx_Data = RDR2;                            // Get reception data

       if (SSR2_ORE || SSR2_FRE) Rx_Error = 2;    // Reception errors?
```

```
        else if (LIN_State == 2) {                        // Synch field read back?
                if (Rx_Data != 0x55) Rx_Error = 3;
                else {
                        TDR2 = LIN_Header;                // Send LIN_Header
                        LIN_State = 3;
                }
        }

        else if (LIN_State == 3) {                        // Header read back?
                if (Rx_Data != LIN_Header) Rx_Error = 4;
                else {
                        if (Master_Send) {                // Master wants to send?
                                TDR2 = LIN_Data[LIN_Count];    // Send LIN Data
                                LIN_Checksum = LIN_Data[LIN_Count];
                        }
                        LIN_State = 4;
                }
        }
        else if (LIN_State == 4) {                // LIN Data read back / Slave Data
                if (Master_Send) {                        // Master sent data?
                        if (Rx_Data != LIN_Data[LIN_Count]) Rx_Error = 5;
                        LIN_Count++;
                        if (LIN_Count == DATALENGTH) {        // End of message reached?
                                LIN_Count = 0;
                                LIN_State = 5;
                                LIN_Checksum = LIN_Checksum ^ 0xFF;
                                TDR2 = LIN_Checksum;
                        }
                        else {
                                TDR2 = LIN_Data[LIN_Count];    // Send next LIN Data
                                LIN_Checksum = LIN_Checksum + LIN_Data[LIN_Count];
                                if (LIN_Checksum > 0xFF) LIN_Checksum -=0xFF;
                        }
                }
                else {                                    // Receive Data from Slave
                        LIN_Data[LIN_Count] = Rx_Data;
                        LIN_Checksum = LIN_Checksum + Rx_Data;
                        if (LIN_Checksum > 0xFF) LIN_Checksum -=0xFF;
                        LIN_Count++;
                        if (LIN_Count == (reply_count+1)*DATALENGTH)
                        {                                 // End of message reached?
                                LIN_Count = reply_count*DATALENGTH;
                                LIN_State = 5;
                                LIN_Checksum = LIN_Checksum ^ 0xFF;
                        }
                }
        }

        else if (LIN_State == 5)              // LIN Checksum read back / Slave Checksum
        {
                if (Rx_Data != LIN_Checksum)  Rx_Error = 6;

                SCR2_RXE = 0;
                LIN_State = 0;
                reply_count = 0;
        }
}

else                                          // Not recognized interrupt cause
{
        Rx_Error = 7;
        SSR2_RIE = 0;                         // disable reception interrupt
}
} //end of ISR
```

The static char array `LIN_Data[]` is used to store incoming and outgoing data. After the LIN Header is sent, the variable `Master_Send` decides if the ISR will send data (`Master_Send == 1`) or if the ISR will store the incoming data from a slave (`Master_Send == 0`).

5.2 Generating LIN frames

5.2.1 Master sends LIN frame

According to the software example you have to follow these steps to send a LIN frame from master to slave:

1. Set LIN Header
2. Set Master_Send = 1
3. Set Number of data bytes. (Note that the Identifier field contains the maximum number of data bytes)
4. Write the data bytes in `LIN_Data[]`.
5. Call Start_LIN_Message()

In Start_LIN_Message() the LIN Break is generated.

After the LIN Break is generated, the ISR controls the Handling of the LIN message.

```
void Start_LIN_Message(void)
{
    while (LIN_State >0);  //waits until all messages are handled without error
    Rx_Error = 0;
    LIN_State = 1;
    LIN_Count = DATALENGTH;

    ESCR2_LBD = 0;          // clear possible LIN-Break detection
    ESCR2_LBIE = 1;         // enable LIN Break detection (for read back)

    ECCR2 = 0x40;           // Set LIN-Break via byte access
}

void Send_frame() {

       LIN_Header    = 0x3C;
       Master_Send   = 1;
       DATALENGTH    = 8;

       Set_LIN_Data_Bytes(0x80, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0xFF);

       Start_LIN_Message();

}
```

The `Send_frame()` function is not used in the software example. It just shows the structure of all Functions that are used to send LIN frames.

5.2.2 Master requests LIN frame from Slave

To get a desired response from the Slave, you have to initiate a Slave Response frame. This is done by the reserved Identifier **0x3D** (according to the Parity bits, the ID Field is **0x7D**).

The Slave is addressed by a prior Master Request frame.

Like in the example above, you have to set LIN Header, reset Master_Send and generate a LIN Break:

1. Set LIN Header (0x7D)
2. Set Master_Send = 0
3. Set Number of data bytes
4. Call `Start_LIN_Message()`
5. The incoming data bytes will be stored in `LIN_Data[]`.

5.3 Implemented Features

The Example Software provides the following features:

- Periodical sending of Dummy Frame (GetStatus)
- Stall Detection with automatic stop
- Setting Motor Parameters individually:
  - Changing Speed
  - Changing max Operation Current (`Irun`)
  - Changing Hold Current (`Ihold`)
  - Toggle Direction
  - Setting Step Mode
- Reading the Full Status
- Setting a new Position

5.3.1 DoGetFullStatus

This function uses a master request frame to get two response frames from the AMIS Device. The Data is stored in LIN_Data[0 to 7] and LIN_Data[8 to 15].

```
#define MASTERSEND = 0x3C
#define SLAVESEND  = 0x7D

void DoGetFullStatus () {
       DATALENGTH = 8;
       reply_count = 0;
              Set_LIN_Data_Bytes(AppCMD,GetFullStatus,AD,0xFF,0xFF,0xFF,0xFF,0xFF);

       LIN_Header = MASTERSEND; // Master sends data to slave
       Master_Send = 1;
       Start_LIN_Message();

              if (!Rx_Error)
              {
                     wait(10000);
                     reply_count = 0;
                     LIN_Header = SLAVESEND;    // Master wants data from slave
                     Master_Send = 0;
                     Start_LIN_Message();
              }

              if (!Rx_Error)
              {
                     wait(10000);
                     reply_count = 1;
                     LIN_Header = SLAVESEND;    // Master wants data from slave
                     Master_Send = 0;
                     Start_LIN_Message();
              }

              while (LIN_State > 0);

// all information stored in LIN_Data[] now
// ...
```

5.3.2 SetMotorParameter

This function is used to set the motor parameters. According to the AMIS Datasheet some data bytes contain more than one parameter. These are placed to the appropriate Bit positions. At the end all parameter are send via one LIN frame.

```
BYTE SetMotorParameter(BYTE adress, BYTE Irun, BYTE Ihold, BYTE Vmax, BYTE Vmin,
                           unsigned short SecPos, BYTE Shaft, BYTE Acc,
                           BYTE AccShape, BYTE StepMode) {
       BYTE data4, data5, data6, data7, data8;
       if ((Motion & 0x03) > 0) return 0; // if motor is in motion break
       else {
              data4 = (Irun << 4) + Ihold;
              data5 = (Vmax << 4) + Vmin;
              data6 = (BYTE)((SecPos & 0x0700) >> 3) + (Shaft << 4)
                       + (Acc & 0x0F);
              data7 = (BYTE)(SecPos & 0x00FF);
              data8 = 0xE3 + (AccShape<<4) + (StepMode << 2);

              Set_LIN_Data_Bytes(AppCMD,SetMotorParam,adress,data4,
                                    data5,data6,data7,data8);

              reply_count = 0;
              DATALENGTH = 8;
              LIN_Header = MASTERSEND; // Master sends data to slave
              Master_Send = 1;

              Start_LIN_Message();
              while (LIN_State > 0);   // process complete
       return 1;
       }
}
```

5.3.3 Set Dynamic Identifier

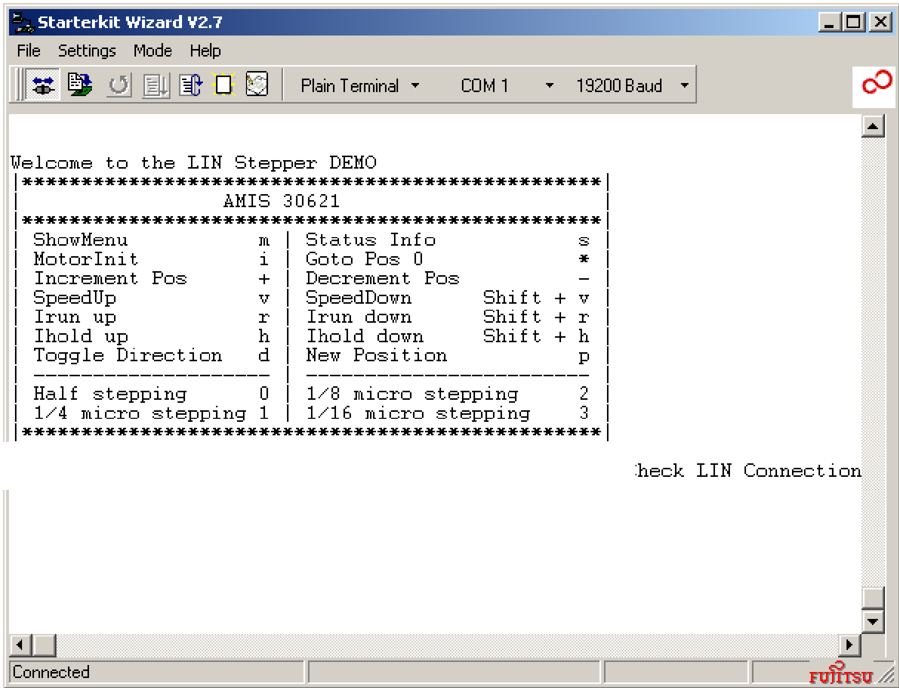
In the example four dynamic Identifiers are used: GetStatus, SetPosition (16-bit), General Purpose 2 Data bytes and General Purpose 4 Data bytes. The following source code shows how these IDs are linked to the appropriate ROM pointers

```
void Set_Dynamic_ID() {

       LIN_Header = MASTERSEND;
       Master_Send = 1;
       DATALENGTH = 8;
       LIN_Data[0] = AppCMD;
       LIN_Data[1] = DynamicIDassignment;
       LIN_Data[2] = 0x80;              //AD = 0, non broad
       LIN_Data[3] = 0x81;              //ID1 = 101000 ROMp1 = 0001
       LIN_Data[4] = 0x0E;              //ID2 = 000000 ROMp2 = 0011
       LIN_Data[5] = 0x40;              //ID3 = 010000 ROMp3 = 0100
       LIN_Data[6] = 0x10;              //ID4 = 011000 ROMp4 = 0000
       LIN_Data[7] = 0x60;
       reply_count = 0;

       Start_LIN_Message();

       while ((LIN_State>0) && (timeout < 10000)) timeout++;
       if (timeout < 10000) timeout = 0;
       else Rx_Error = 8;
}
```

To understand the values of LIN_Data[2..7] see the preceeding code snippet and how the IDs and ROM pointer are inserted in the LIN frame.

5.4 User Menu

The program is controlled via UART Interface. Connect your PCs COM Port to UART 3 and connect with 19200 baud. The following window should appear:



To get a movement you have first to press ‘i’ to run MotorInit. By this the AMIS device is switched to normal operating mode and some basic motor parameters are set. Then press ‘+’ to drive the motor to a new position. The new position is the actual position + 0x100.

The implemented features are listed in the menu and explained by itself. The `Speed-`, `Irun-` and `Ihold-`Functions increase and decrease the value by one respectively. The value range from 0 to 15. If you call the “New Position”-Function you have to set the new position value by 4 digits in hex.

5.5 Example Project Structure

The Example software is subdivided into several files:

`AMIS30621.c +.h`	Functions to control the AMIS Device
`LIN_ISR.c`	The LIN Interrupt Service Routine
`uart.c`	The LIN and UART concerning functions
`main.c`	The main program and menu

5.6 Source Code

Please find the source codes of this application note adapted for MB90350 series in the zip

file: mcu-an-300037-e-v11-lin_stepper_amis.zip

5.7 Information in the WWW

Dedicated information on Cypress Microcontroller products including datasheets and manuals, software examples, application notes and tools can be found here:

http://www.cypress.com/cypress-microcontrollers

Information about AMI Semiconductor can be found here:

www.amis.com

Specific information about the LIN Motor driver (AMIS-30621) can be found here:

http://www.onsemi.com/PowerSolutions

6 Document History

Document Title: AN205253 - F²MC-8L/16LX/FR Family with LIN-USART

Document Number: 002-05253

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	01/15/2007 07/13/2007	Initial Release Corrected reference file names
*A	5080057	MKEA	01/11/2016	Migrated Spansion Application Note from MCU-AN-300037-E-V11 to Cypress format
*B	5844537	AESATP12	08/04/2017	Updated logo and copyright.
*C	6056603	NOFL	02/02/2018	Removed references of obsolete specs across the document. Updated hyperlinks across the document. Updated to new template. Completing Sunset Review.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709