MOTOROLA
intelligence everywhere™

digital dna™

LIN-bus HID Lamp Levelling Stepper Motor Control Using Motorola 908E625 Reference Design

Designer Reference Manual

M68HC08 Microcontrollers

DRM047
Rev. 0, 12/2003

MOTOROLA.COM/SEMICONDUCTORS

LIN-bus HID Lamp Levelling Stepper Motor Control Using Motorola 908E625 Reference Design

Designer Reference Manual — Rev 0

by: Libor Prokop, Petr Cholasta
MCSL, Rosnov

Table of Contents

List of Figures

[redacted]

List of Tables

Section 1. Introduction

1.1 Overview

This reference design describes the development of a LIN based High Intensity Discharge (HID) headlamp levelling system, which controls the stepper motors in the lamp module to compensate for the motion of the vehicle. (In this implementation, the vehicle's movement is simulated on a PC). The design consists of a master control board that is based on a 16-bit HCS12 MCU, a PC with graphical user interface (GUI) and slave nodes that control the levelling stepper motors (see **Section 2. System Concept** for a full description). The slave nodes are driven by an innovative dual-die product (908E625) that contains an industry-standard FLASH based, M68HC08 MCU and an SMOS power die that includes VReg, LIN interface, Hall Sensor interface and high-side and low-side drivers. (See appendices and data sheet for additional information.)

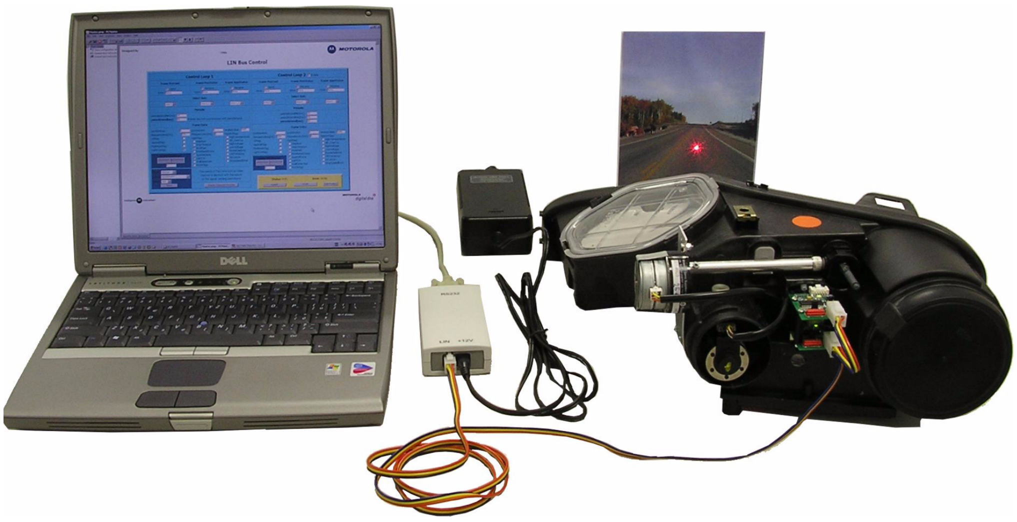


Figure 1-1. System Concept

The concept of HID lighting levelling, the LIN-bus protocol, and the general system concept are given to provide the reader with some valuable background information. The hardware and software (for both master and slave) are described in detail to allow the design and implementation to be fully understood. Finally, a description of the user interface is provided to demonstrate the ease of use and flexibility of the system.

1.2 Summary

The reference design demonstrates that the HID lamp levelling system can be controlled over the LIN-bus, and that several system benefits can be achieved using this method, compared with the conventional wired implementation. These benefits include system configuration, as the software and parameters can be updated over the LIN-bus, and scalability, as it is easier to add functions to a bus-based application. In addition, using the 908E625 dual-die device offers a low-cost implementation, as the system cost is reduced due to the minimal external hardware required. The 908E625 provides all the functions necessary to implement the slave nodes, and its small footprint and on-board FLASH make this device ideal for many stepper motor control applications.

1.3 HID Headlamp Levelling

At the present time, car front lighting systems are changing rapidly. There are many techniques that can improve visibility under low light conditions. One of the requirements is automatic vertical beam control. This is necessary for headlamps based on discharge lamps.

Other systems are Bi-Xenon headlamps. Today's Bi-Xenon headlamp operates with one single Xenon bulb and creates both low beam and high beam. The cut-off is generated via a special shield, which can be flipped. The shield control is provided by means of an actuator, which can be a motor or a solenoid.

The low beam of today's headlamps is characterized by a specific shape and distribution regulated by ECE regulations. Independent of the speed, the type of road, and the weather conditions, the headlamps of today are always constant. But we will have next generations Advanced Front Lighting (AFS) systems soon. A new lighting system can be adopted to this various conditions. The target is to achieve better visibility at night, when directing the lights according to the steering wheel angle or due to the speed. To see where the car is going, rather than putting the light always straight, is the background of this idea. So we have horizontal beam control. Other systems use an auxiliary bending lamp. See **Section 9. References**: **3**, **4**, **5**, **6**, **7**, and **8**.

Advanced headlamp systems are quite complex. They need sophisticated optics, sensors and actuators. So, today the system costs targets their implementation to high-end car segments. If we lower the cost, we spread them to all car segments. A key factor in lowering the system cost is integration and the use of reasonable components and protocols.

This reference design describes a good solution for headlamp levelling systems based on stepper motor actuators with a possibility of vertical and also horizontal beam control. The system can benefit from a single chip solution and communication via low-cost LIN-bus protocol.

1.4 LIN-bus

All modern car electronic communication is based on serial bus protocols. These have many advantages over classical wired systems. For example, a control system with stepper motor actuators can be split into distributed controllers connected with a single-wire bus. There is no doubt that this bus system saves on wiring and connectors, so the system cost is significantly reduced.

The LIN-bus serial communication protocol (see **Section 9. References**, **2**) was designed for automotive applications, but it can also be used for other devices (white-goods, printers, and copiers, for example). In the case of car-body electronics, it is used for air-conditioning, mirror control, seat control, and light levelling. The advantage of LIN-bus over other bus protocols (like CAN) is low system cost. This is because the LIN-bus protocol is based on standard and cost-effective serial SCI (UART compatible) hardware modules. These are implemented on most Motorola MCU/DSP devices. An enhanced SCI is called ESCI.

Other serial bus protocols like CAN require a specialized hardware module. They can have higher communication speed than LIN-bus. But the overall system cost of such systems is much higher. Therefore many of car electronics systems should be based on LIN-bus protocol.

This reference design shows that the LIN-bus protocol speed is fully sufficient for a headlamp leveller with a stepper motor actuators. The advantage versus other bus protocols, such as CAN-bus, is low system cost.

1.5 Definitions and Acronyms

The definitions and acronyms used in this reference design are listed below.

(signal) Acceptor	the device which receives and responses to a bus signal
AFS	Advanced Front Lighting System
Cairone	see Power Die
ESCI	Enhanced Serial Communication Interface
DSP	Digital Signal Processor
ECT	Enhanced Capture Timer
HID Lamp	High Intensity Density headlamp
LIN Leveller	the system described in this reference design. It consists of the LIN master and LIN stepper controllers
LIN-bus	a local interface network standard for serial asynchronous communication (see **Section 9. References**, 2)

LIN Master	master board with LIN master software
LIN Master Software	the LIN-bus master software for stepper motors control and communication
LIN SIO Wire	LIN-bus signal (Serial Input Output) wire
LIN Stepper Controller	LIN-bus slave board with General Purpose IC 908E625 with LIN Stepper software
LIN Stepper Software	the LIN-bus slave stepper motor control software for 908E625 described in this AN
LIN Stepper Board	slave stepper board hardware for LIN slave stepper controller. The PCB layout is same as LIN Enhanced Stepper Board. Some parts are not populated.
LIN Enhanced Stepper Board	slave stepper board hardware for LIN Master with sensor connector and other components
LIN Master Board	master board hardware for LIN Master
MCU	microcomputer unit
PCB	printed circuit board
Power Die	part of 908E625 with H-bridges LIN physical layer high-side switch connected to the MCU part with SPI interface. In some references, the Power Die chip is called Cairone
SCI	Serial Communication Interface module (outside Motorola called also UART)
(signal) provider	the device which send the slave task of the bus signal

Section 2. System Concept

The system application was designed to control stepper motor actuators from a GUI running on a PC. The PC is connected to the LIN Master board via RS232 serial ports and they form the master controller. The LIN Master board is then connected to the LIN Stepper Controller slaves via a serial single-wire LIN-bus. The master controller can handle several stepper motor actuators, so it can demonstrate levelling of two headlamps around vertical and horizontal axes. Each actuator consists of one stepper motor and LIN Stepper Controller slave node. The actuators control positioning around a dedicated axis.

The control system consists of the following modules:

- Personal Computer
- LIN Master – LIN master node
- LIN Stepper Controller – LIN slave node

Each module is programmed with dedicated software.

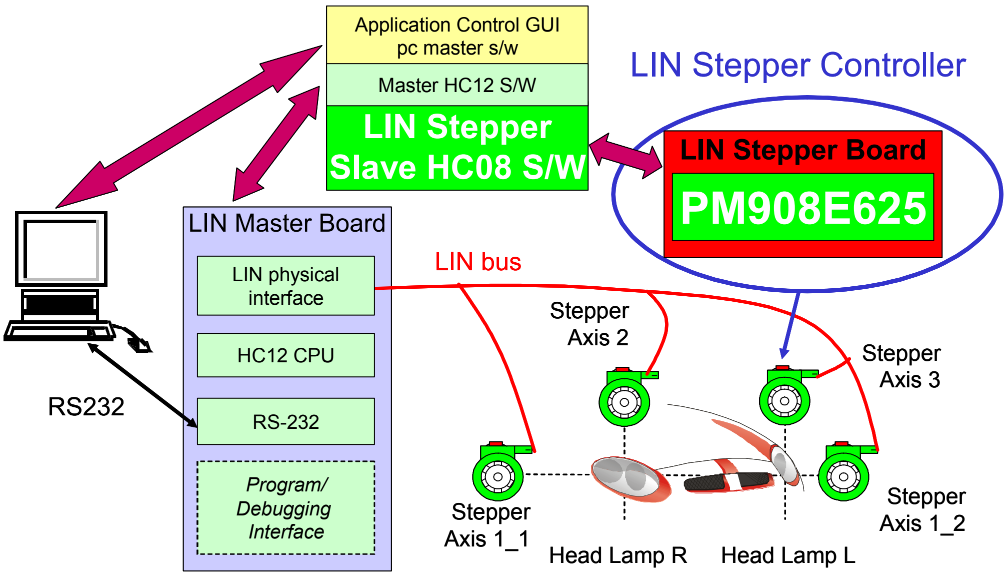


Figure 2-1. System Concept

2.1 System Features

- LIN-bus Interface rev 1.2
- Bus speed 19.2 kbps
- Slave IC without external crystal or resonator
- Slave node clock synchronization ±15%
- Each LIN slave controls one biphase bipolar stepper motor
- Motor phase current limitation up to 700 mA
- Supply voltage 12 V d.c.
- Stepper motor control with stepping acceleration and deceleration ramp
- Stepping frequency up to 2500 Hz
- Slave parameters configuration via LIN-bus
- Slave LIN signals reconfiguration via LIN-bus
- LIN signals defined for 2D control with 3 (and “half”) axes
- Embedded code written in C-language

2.2 LIN Stepper Controller

The LIN Stepper Controller is the LIN-bus slave node. It does the following:

- controls bi-phase bipolar stepper motors to a required position with automatic speed acceleration and deceleration
- communicates with the master node via LIN-bus
- provides LIN-bus clock synchronization (the slave node uses internal on-chip oscillator with no external components)
- provides parameters configuration/programming via LIN-bus when requested by LIN-bus configuration signals
- provides LIN signals reconfiguration via LIN-bus to a required axis when requested by LIN-bus configuration signals

All the necessary hardware of this LIN-bus slave node is comprised in one SOIC 54-lead packaged 908E625, with some external connectors and capacitors. The 908E625 includes the Motorola M68HC08 core, and its functionality is determined by the LIN Stepper software.

The software provides all control functionality for stepper positioning control. The absolute required position and maximum speed are determined by LIN signals from the master.

The LIN Stepper Controller clock is based on an internal RC on-chip oscillator. Therefore, the LIN-bus driver (using the ESCI module on 908E625) can handle the LIN-bus clock synchronization range according to the LIN-bus specification 1.2. (see **Section 9. References**, **2**)

The LIN stepper controller has the capability to change some configuration parameters via the LIN-bus. These parameters can be stored in FLASH memory. For this purpose, there are LIN-bus configuration signals (Master Request and Slave Response frames — see **Section 5. LIN Master Software Description**) defined for the system.

- node ID number (0 to 255)
- uAppConfigByte1

- motor block and run current limitation
- motor stepping start frequency
- motor stepping acceleration
- period motor stop time-out
- motor stall position
- motor parking position
- motor position correction

Four groups of LIN signal frames are defined to control the dedicated axis:

- Axis 1_1 (to be used as horizontal axis, right lamp) signals group
- Axis 1_2 (to be used as horizontal axis, left lamp) signals group
- Axis 2 (to be used as vertical axis, right lamp) signals group
- Axis 3 (to be used as vertical axis, left lamp) signals group

The reconfiguration of the LIN signals (see above) means that the slave can be programmed to be active only on one of the four signal groups (so it ignores signals for other actuators). This signal group can also be chosen from the LIN-bus master with the LIN-bus configuration signals (Master Request and Slave Response frames). The benefit of this solution is that there can be one universal controller software for any axis actuator. It can then be configured via LIN-bus for any axis, as required.

2.3 LIN Master

The function of the master board depends on the selected mode, chosen by means of a jumper on the board (see **Section 3.1. Master Board**).

The modes are as follows:

- PC master mode (PCM)
- Master mode (M)
- Debug mode (D)
- Pass mode (P)

2.3.1 PC Master Mode

The master board is connected via an RS232 line to the PC (with installed PC master software), and acts as a LIN Master node controlled by the user interface (HTML page).

The LIN Master performs the following functions:

- LIN-bus Run/Stop/Sleep/Wake-up control
- Periodical sending/receiving of up to 2*3 LIN-bus frames within two timing loops
- Possibly fully control the position of up to two independent LIN steppers
- Manual or automatic generation of the required position of LIN stepper. In the case of automatic generation, a read-out of a predefined signal curve (simulation of real application) is provided
- LIN Stepper configuration and programing (using master request, slave response frames defined for this application)

2.3.2 Master Mode

The master board acts as a an autonomous LIN Master node (without using a personal computer). It is similar to the PC master mode in automatic mode (automatic generation of the required position of the LIN Stepper).

2.3.3 Debug Mode

In this mode it is possible to program and debug any LIN Stepper controller via a special 10-pin connector. The LIN Master performs the following two functions:

- it gives some defined signals to the 10-pin connector to put the LIN Stepper controller into MON08 debugging mode (see **Section 9. References**, **10**, Section 10. Monitor ROM).
- it provides a serial communication gateway between the personal computer (using RS232 line) and the 10 pin connector

The personal computer provides software download or debugging with a dedicated programming (e.g. Pemicro) or debugging software (e.g. Metrowerks Hiwave Debugger)

2.3.4 Pass Mode

Master board acts as a gateway between RS232 and LIN-bus (copy signals between RS232 and LIN-bus). The mode can be used, if LIN-bus protocol is implemented in the personal computer.

2.4 Personal Computer

The personal computer is used for application control using a graphical user interface. The PC host computer communicates with the LIN Master via the RS232 serial port.

The graphical user interface is implemented as an HTML script running on PC master software (see **Section 9. References**, 1). The PC master software is a universal software tool for communication between the personal computer and embedded applications based on an MCU or DSP. The principle of the PC master software is shown in **Figure 2-2**.

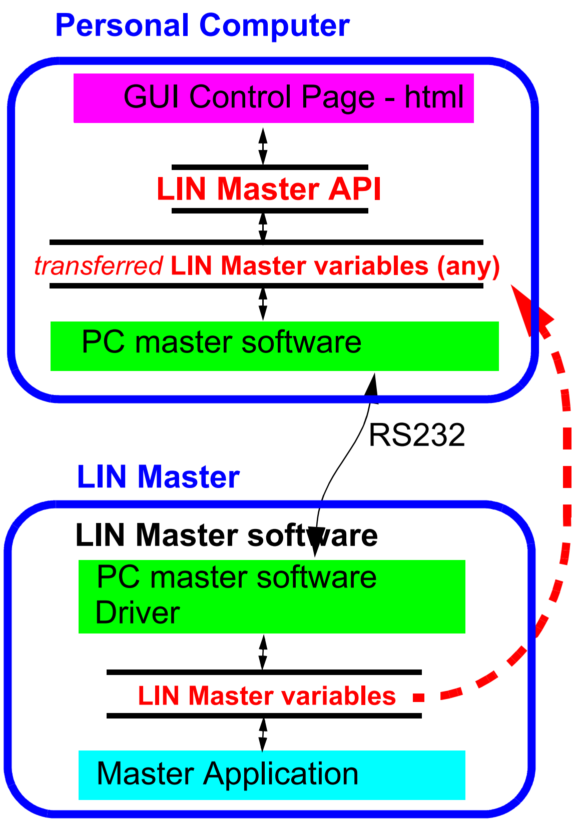


Figure 2-2. PC Master Software Principle

It consists of a PC master software running on a PC and a PC master software driver with protocol implementation running on the LIN Master. The driver is implemented as a resident software routine (interrupt based) included in the LIN Master software. The communication medium is RS232 in this headlamp levelling application.

The basic feature of the PC master software is that all the MCU/DSP variables can be easily transferred to the personal computer for reading or modification. The user can simply specify which of them will be read/modified and the period of each variable reading.

NOTE: *The PC master software provides a communication layer between any LIN Master software variables and the graphical interface control page, which is written in HTML language.*

The LIN Master application interface (API) is then a defined set of LIN Master variables. The GUI is then realized as an HTML script file, which reads/modifies the variables in the API. The graphical user interface is described in **Section 7. User Interface Description**.

Section 3. Hardware Description

3.1 Master Board

The master board (**Figure 3-1**) is supplied with 12 V from an external source and can switch LIN supply currents up to 5 A. It can be used in four different modes, as described in **Section 2.3. LIN Master**, depending on position of the jumper on the MODE SELECTION header (currently PCM -> PC master mode). After each change of mode, the RESET button must be pressed.

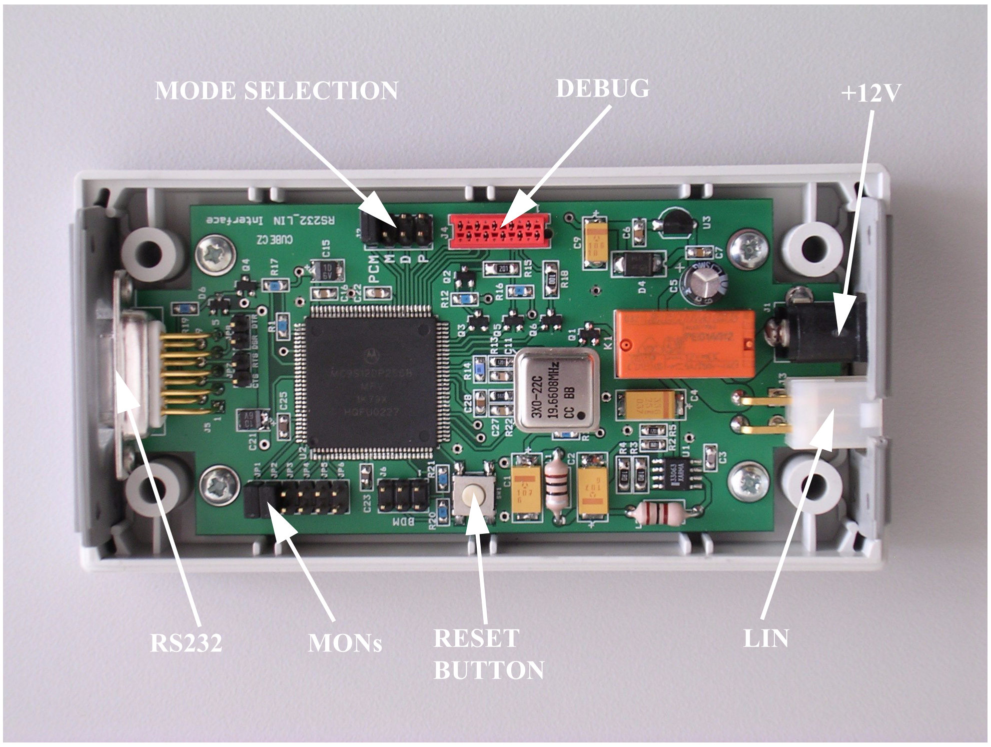


Figure 3-1. Master Board

The heart of the system (see **Figure 3-2**) is the 16-bit MC9S12DP256B MCU (see **Section 9. References**, **12**), which is supported by the bus drivers and power stage. The MC33399 (see **Section 9. References**, **11**) is used as the LIN interface, and can drive up to 16 slaves.

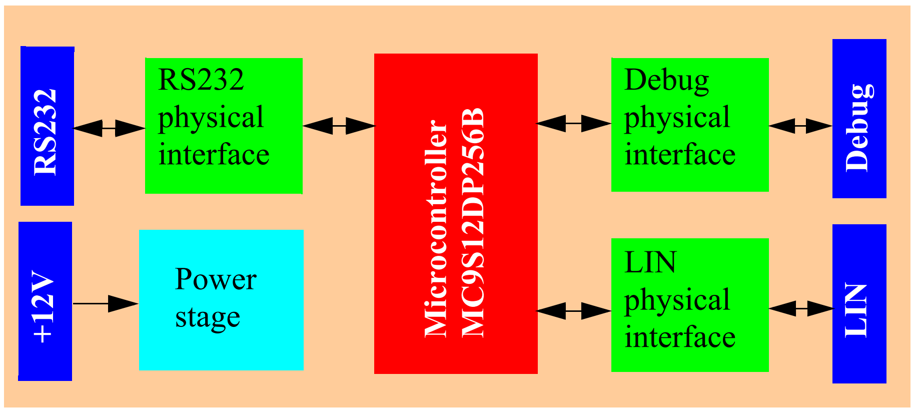


Figure 3-2. Master Board Concept

This board is protected against incorrect supply voltage polarity and provides this feature to all LIN Stepper Controllers supplied by the Master Board.

3.2 Slave Board

The LIN Stepper Controller hardware is based on the 908E625 device. The hardware consists only of few components as shown in **Figure 3-3**. It is due to the fact that all the functionality is provided by the 908E625 device.

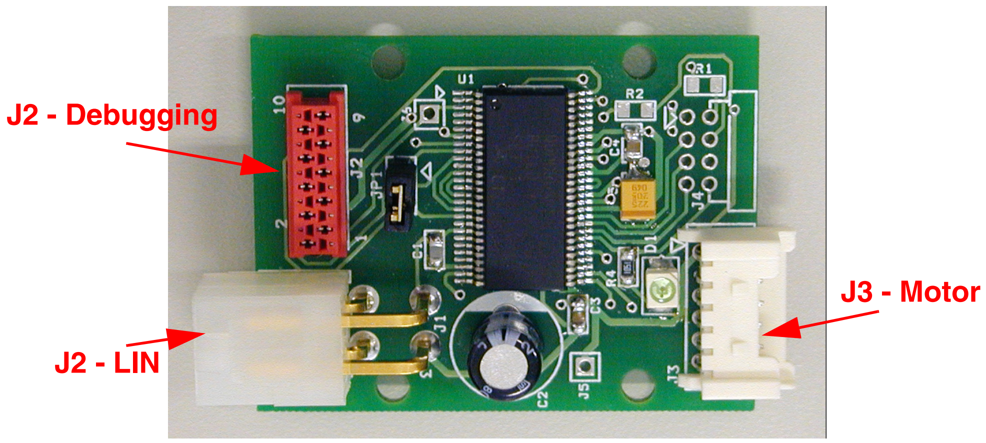


Figure 3-3. LIN Stepper (Slave) Board

CAUTION: *A slave board based on 908E625 can be even smaller than the LIN Enhanced Stepper Board. The PCB from* ***Figure 3-3*** *is universal. The sensor support - connector J4 and resistors R2, R1 from* ***Figure A-2*** *are not populated. It 's because they are not used for current LIN Stepper Controller with the LIN Stepper software.*
Also the LED diode D1, R4 and headers J5, J6 are not necessary for system functionality.

The PCB layout was designed as an universal LIN Enhanced Stepper Board according the schematics in **Figure A-2**. It has some additional sensor inputs. This could be used for some applications with a Hall sensor or analog signal feedback.

The LIN Stepper Controller does not use any sensor feedback. The schematics of the LIN Stepper Board s displayed in **Figure 3-4**. It uses the LIN Enhanced Stepper Board PCB layout, but some components are not populated.

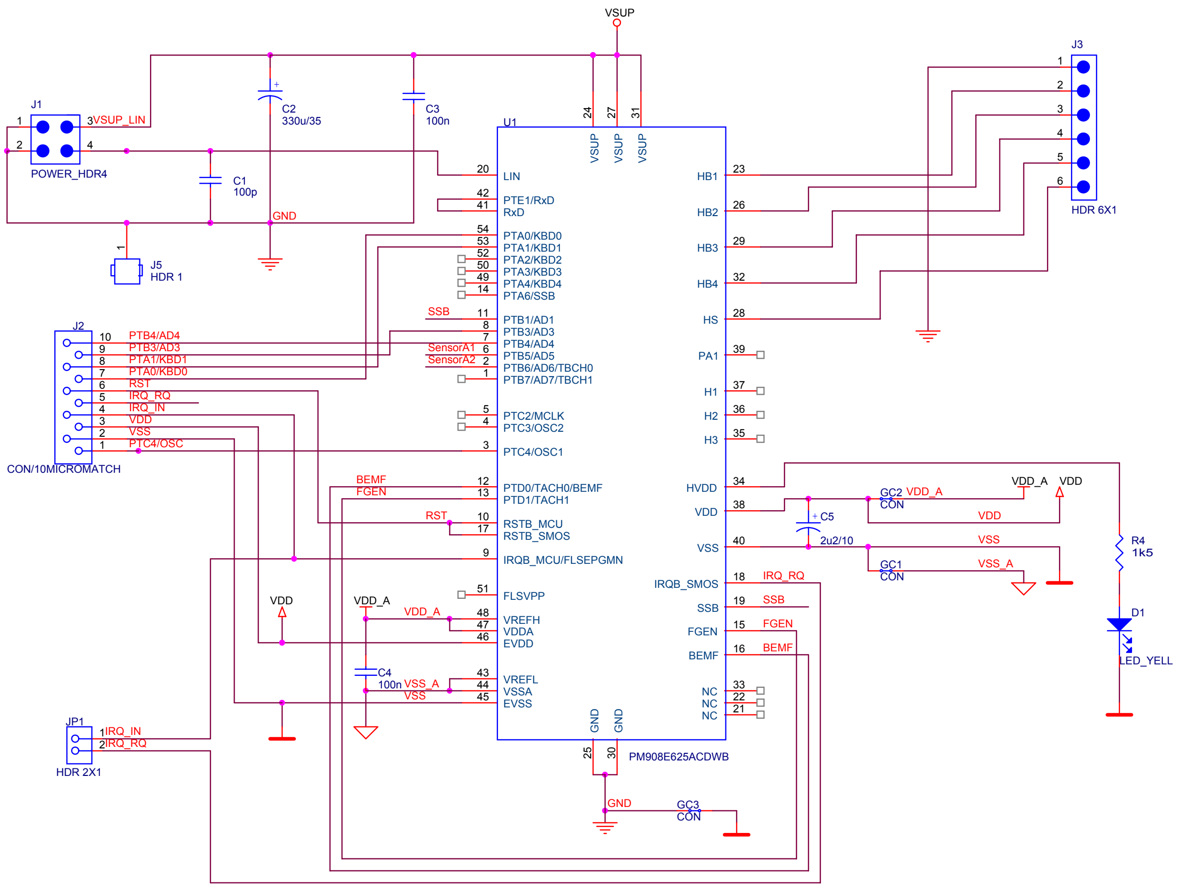


Figure 3-4. LIN Stepper Controller (Slave) Board Schematic

The 908E625 schematics with the LIN Stepper Controller functional blocks is in **Figure 3-5**. The he functional blocks are described below.

3.2.1 MCU and Power Die with SPI

MCU 908EY16 chip and Power Die chip (Cairone) forms the 908E625 device in one package. These two chips are connected with SPI signals and some other signals. So the control of the Power Die (like Half-bridges control) is provided with SPI communication. The SPI communication pins MISC, MOS, SPCLK are connected inside of the 908E625 package (see **Section 9. References**, **9**).

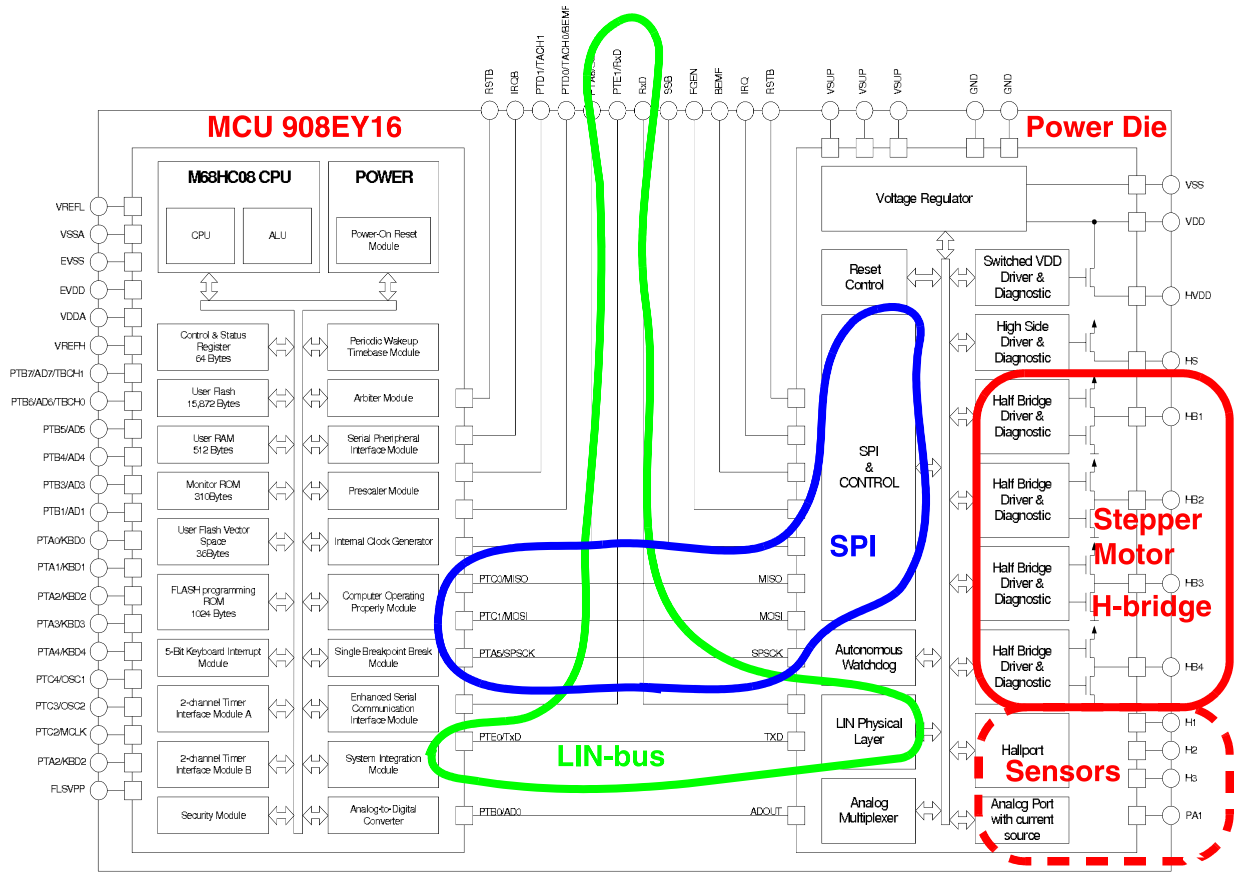


Figure 3-5. 908E625 Blocks Usage

3.2.2 LIN-bus

The LIN-bus is connected to the connector J1. The capacitor C1 filters the bus and the signal is connected to the pin LIN (20) of the physical layer. The physical layer is internally connected to the PT0/TXD pin of the MCU chip. The PTE1/RXD pin 40 is connected to RxD pin 41 externally. The IRQB_SMOS pin 18 from the Power Die module and IRQB_MCU pin 9 are used to initiate MCU interrupt from the Power Die chip. In LIN Stepper Controller this is used for MCU wake up from the sleep via wake-up. Therefore jumper JP1 must be connected for user (standard operational) mode.

3.2.3 Software Download and Debugging

Connector J2 is used for software download or debugging. This is based on so called MON08 mode (see **Section 9. References**, **10**, Section 10 Monitor ROM)

The signals PTB4/AD4, PTB3/AD3, PTA1/KB1, PTA0/KB0, IRQ_IN must be set according to **Table 3-1** to put the MCU into MON08 mode for software download or debugging (see **Section 9. References**, **10**). RST is the MCU reset pin. The MON08 mode must be timed with external clock - PTC4/OSC.

There must be 9V for debugging on the IRQ_IN pin. Therefore the jumper JP1 must be open. In user (standard operational) mode the IRQ_IN must be attached to the IRQ_OUT from the Power Die module. This is used for some operations like wake-up condition, where the Power Die module. Therefore JP1 must be closed for user (standard operational) mode.

Pin1 PTC4/OSC is precise clock input for MON08 mode. There must be external clock source for the software download and debugging.

Table 3-1. Connector J2 Signals

Pin No	Input/ Output	Pin Name	Description	MON08 mode
1		PTC4/OSC		19,6608kHz
2		VSS	Ground	GND
3		VDD	5V supply	5V
4	In	IRQ_IN	MCU IRQ Input	9V
5	Out	IRQ_RQ	Power Die IRQ output	jumper JP1 open
6	In	RST	MCU Reset input	falling edge
7	In/Out	PTA0/KB0	MON08 mode	serial communication 19.200 kBaud
8	In	PTA1/KB1	MON08 mode	GND
9	In	PTB3/AD3	MON08 mode	GND

Table 3-1. Connector J2 Signals

Pin No	Input/ Output	Pin Name	Description	MON08 mode
10	In	PTB4/AD4	MON08 mode	Vdd = 5V (via 10k resistor)

3.2.4 Stepper Motor Dual H-bridge and High Side Switch

The bi-phase bipolar stepper motor is powered with four half-bridges. They are attached to the connector J3. The connector pin 6 is high side switch which can possibly be used for lamp on/off control

Table 3-2. Connector J3 Signals

Pin No	Input/Output	Signal	range
1	GND	GND	GND
2	Out	HB1 motor phase 1-1	0-5V
3	Out	HB2 motor phase 1-2	
4	Out	HB3 motor phase 2-1	0-5V
5	In	HB4 motor phase 2-1	
6	In	High Side	0-5V

3.2.5 Power Supply and Decoupling

The LIN Stepper Controller is powered from LIN-bus connector. The Power Die has internal voltage regulator with the outputs VDD(pin30) and VSS (pin 40). This is used to power the analog VDDA,VSSA and digital part EVDD,EVSS of the MCU chip. The connections and capacitors C5, C4 were used for decoupling VDD,VSS_A from VDD_A,VSS_A

3.2.6 Hall Port and Sensor

The LIN Stepper Controller does not use any sensors. Therefore the connector J4 is not displayed in **Figure 3-4**. However the LIN Enhanced Stepper Board (**Figure A-2**) was designed for possible use of Hall sensors or analog signals. There is a place for connector J4, resistors R2, R1.

Table 3-3. Connector J4 Signals

Pin No	Input/Output	Signal	range
1	GND	VSS_A	GND
2	In	Sensor Analog 1	0-5V
3	In	Sensor Hall 1	
4	In	Sensor Analog 1	0-5V
5	In	Sensor Hall 2	
6	In	Sensor Power Analog 1	0-5V
7	In	Sensor Hall 3	
8	Output	HVDD Switchable 5V output to power sensors	5V

Section 4. Messaging Scheme Description

This section describes LIN messaging.

4.1 Axis and Signal Providers and Acceptors

As shown in **Figure 2-1**, each LIN Stepper Controller node has a relation to the concrete controlled axis. The LIN messaging scheme was designed to support this concept. Most of the signal in **Table C-1** has 3 or 4 modification, which differs only with the identifier byte. Each node is programmed to be an acceptor (acts upon) and providers (sends the response fields) for one exclusive signal set. The signal sets are marked according to the controlled axis:

- A1_1
- A1_2
- A2
- A3

The signal provider and acceptor is determined by the preprogrammed axis.

A special meaning is given to the horizontal Axis1. It is expected that one control signal is sufficient for both right and left headlamp. Therefore two nodes with signal sets Axis1_1 and Axis1_2 are acceptors for Axis1 signals. This uses and demonstrates the LIN-bus multicast concept.

There are two ways to program the LIN Stepper Controller node to a dedicated axis:

1. Setting appropriate target Axis1_1, Axis1_2, Axis2, Axis3 in the lin_stepper.mcp file. This will create the compiled code with the required setting.

2. using LIN reconfiguration as described in **Appendix C.3. LIN Leveller Configuration Frames** or **Section 6.1.9. Reconfig LIN**.

NOTE: *The user must guarantee that there will be no other nodes with the same axis connected to one LIN-bus.*

4.2 LIN Leveller Basic Messaging

The basic message frames are used for standard control operation. The frame provided by master node is:

- frmPosCmd

The frames provided by slave nodes are:

- frmPosStatus
- frmAppStatus

A detailed description of the basic messaging frames and signals is in **Appendix C.1. LIN Leveller Basic Frames**.

The scheduling of the signals is determined by the LIN Master. The LIN Master setting is provided from the PC computer control page and is described in the **Section 7. User Interface Description**. The master serves two messaging loops. Each loop can handle up to three signal frames (frmPosCmd, frmPosStatus, frmAppStatus). Each of the three signal frames can be chosen according to a required axis. Any of the three frames can be disabled.

Loop1 sends signals with period **periodeSend_Loop1**. This period can be modified by the control page (see **Figure 7-2**). If all three frames are enabled, the LIN Master controls consecutive sending of Loop1 frames **frmPosCmd**, **frmPosStatus**, **frmAppStatus** immediately after each other. Then it waits to get the **periodeSend_Loop1** between sending. If Loop2 is also enabled, the three frames of Loop2 are sent after Loop1. The details are described in the **Section 7. User Interface Description**. The default value of the **periodeSend_Loop1** is 30 ms.

4.3 LIN Leveller Configuration Messaging

The Master Request and Slave Response frames are used for the LIN Stepper Controller configuration. The configuration allows adaptation of the LIN Stepper software. Each configuration frames is used to configure the LIN Stepper Controller with node ID equal to the l_u8_rd_nodeID signal (see **Appendix C.3. LIN Leveller Configuration Frames**).

The configuration process covers two functions:

- Parameters Configuration

provides upload and download of the control parameter.

- LIN Reconfiguration

changes the dedicated LIN Stepper Controller configuration. It sets its LIN driver to select the frames and signals according to the defined axis.

A detailed description of the configuration messaging frames and signals is in **Appendix C.1. LIN Leveller Basic Frames**, **Appendix C.3. LIN Leveller Configuration Frames**.

Section 5. LIN Master Software Description

The software is described in terms of:

- General State machine diagram
- Data flow chart for each Master Board mode

5.1 State Machine

Figure 5-1 presents a general description of the implemented software. The main routine consists of *MCU Init*ialization and *Mode selection* procedures.

5.1.1 MCU Initialization

Provides initialization of the microcontroller:

- Ports A, H (pull-up/pull-down), M (wired-or), P (pull-up), S initialization
- Phase Locked Loop setup (Core is running on 48MHz)
- Enable global interrupt mask bit

5.1.2 Mode Selection

Act as a device mode selection module. It tests the Port A (MODE SELECTION header, see **Section 3.1. Master Board**) as long as one out of four possible values (each of them is corresponding to one mode) is recognized. The corresponding subroutine is initialized. In that routine the program is running until reset or power die event. The modes were described in **Section 2.3. LIN Master**.

5.1.3 PC Master Mode Initialization

The actions are following:

- Turn on LIN supply voltage
- Initialize SCI1 and PC master software driver
- Initialize LIN driver (including initialization of SCI0 and ECT channel0)
- Set up ECT (each channels acts as an output compare and causes the corresponding interrupt):
  - channel1 (Loop1 timing)
  - channel2 (Loop2 timing)
  - channel3 (PC master software recorder)
- Set all program flow control and state variables

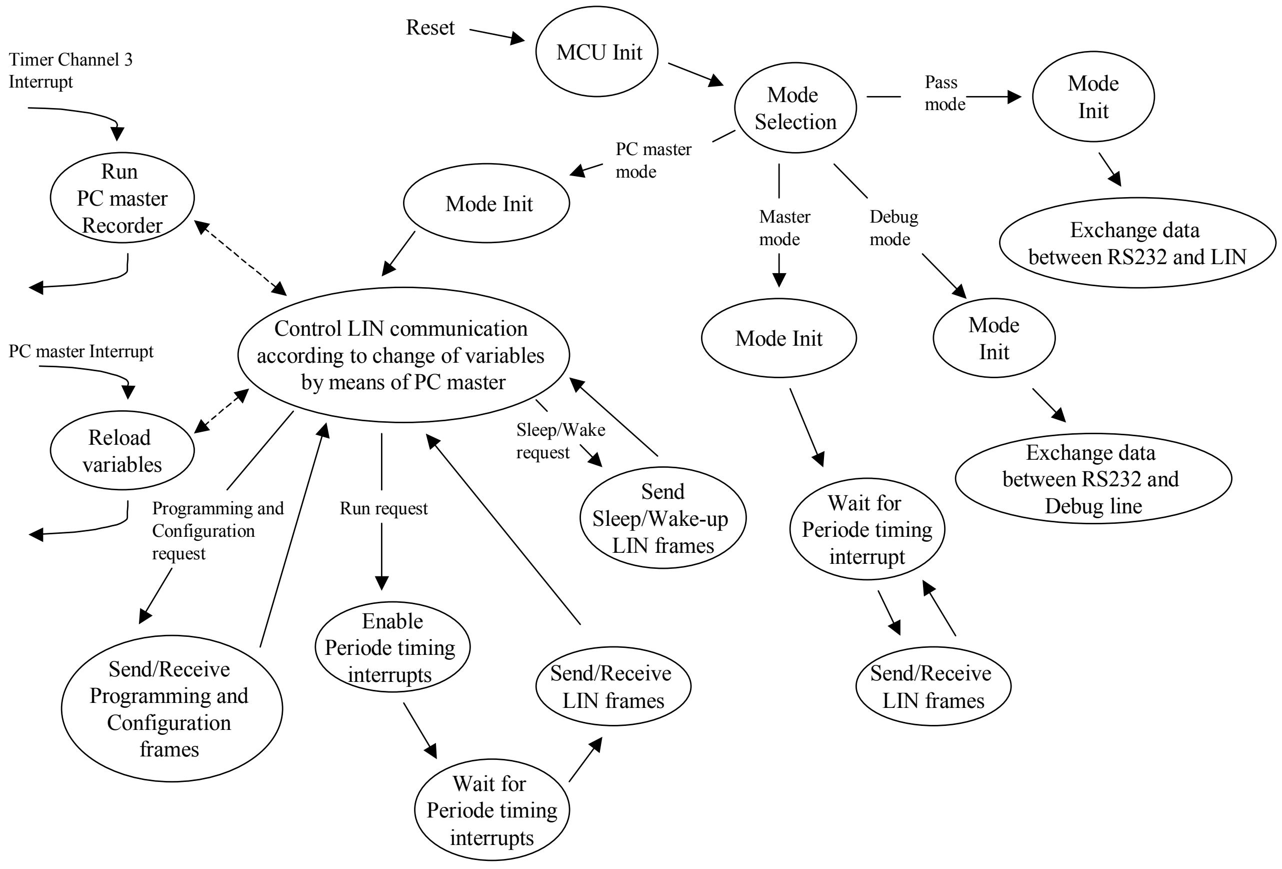

Reset
MCU Init
Mode Selection
Pass mode
Mode Init
Exchange data between RS232 and LIN
PC master mode
Mode Init
Master mode
Debug mode
Mode Init
Mode Init
Exchange data between RS232 and Debug line
Timer Channel 3 Interrupt
Run PC master Recorder
PC master Interrupt
Reload variables
Control LIN communication according to change of variables by means of PC master
Sleep/Wake request
Send Sleep/Wake-up LIN frames
Programming and Configuration request
Run request
Send/Receive Programming and Configuration frames
Enable Periode timing interrupts
Wait for Periode timing interrupts
Send/Receive LIN frames
Wait for Periode timing interrupt
Send/Receive LIN frames

Figure 5-1. Software State Diagram

5.1.4 Master Mode Initialize

Performs:

- Turn on LIN supply voltage
- Initialize LIN driver (including initialization of SCI0 and ECT channel0)
- Set up ECT channel1 (Loop1 timing) as an output compare
- Enable ECT channel1 output compare interrupt
- Set all program flow control and state variables

5.1.5 Debug Mode Initialization

Sets:

- MONs according to pin states on Port P (see **Figure 3-1**)
- All program flow control and state variables

5.1.6 Pass Mode Initialization

- Turns on LIN supply voltage

5.2 Data Flow

Four possible modes of the Master Board are discussed below. Each mode is described with its dedicated flow charts. General overview is possible gain over **Figure 5-1**.

5.2.1 PC Master Mode Program Flow

After initialization, the Master Board performs the actions described in **Section 2.3.1. PC Master Mode**. The data flow charts are on **Figure 5-2** and **Figure 5-3**. The meanings of the bubbles (states) are explained below.

5.2.2 Slave Sleep/Wake-up, Programming and Configuration

Replaces a function that is able to send and receive the frames below according to the states of the control variables (displayed above the bubble) by the LIN-bus:

- Sleep frame
- Wake-up frame
- Frames for configuration of the LIN-bus network
- Programing parameters (displayed below the bubble) of LIN Stepper Controller

The meanings of the variables above and below the bubbles are described in **Section 7.7. Programming and Configuration** excluding *LIN_SleepWakeReq*, that is an element of **Section 7.3. LIN-bus Control**.

5.2.3 Loop1/Loop2 Priority Selector

If there is a request to communicate via the LIN-bus (*LIN_RunStopReq*), the Loop1 timer (ECT channel1 output compare register) is set according to the value of *periodeSend_Loop1*. That determines the next LIN communication time. The Loop2 timer (ECT channel2 output compare register) is set in the same way (*periodeSend_Loop2*) but only if the Loop2 is enabled (*enableLoop2*). Whenever the ECT channel1 or channel2 interrupt arises, the priority of the service routine executing (*Send and Receive LIN messages*) is resolved. Loop1 has the main priority. If a Loop2 interrupt arises, the time remaining to the Loop1 timer interrupt request is calculated. If it is recognized that the remaining time is greater or equal to the time needed to service the Loop2 service routine, then this process is enabled. In the opposite event, the task is deferred. Then, as soon as the Loop1 request is satisfied, it immediately services the Loop2 communication request. Note that there is one exception to this rule; i.e. when the time between Loop1 timer interrupts is always shorter than the time needed to service communication initialized by Loop2. Then the frames of both loops follow each other with minimal distances between them. The distance is determined by the program flow delay, and the distance is negligible in comparison to the time needed to transmit one LIN frame. Those states are indicated by a note on the LIN-bus Control page. The page and the variables mentioned in this subsection are described in **Section 7.3. LIN-bus Control**.

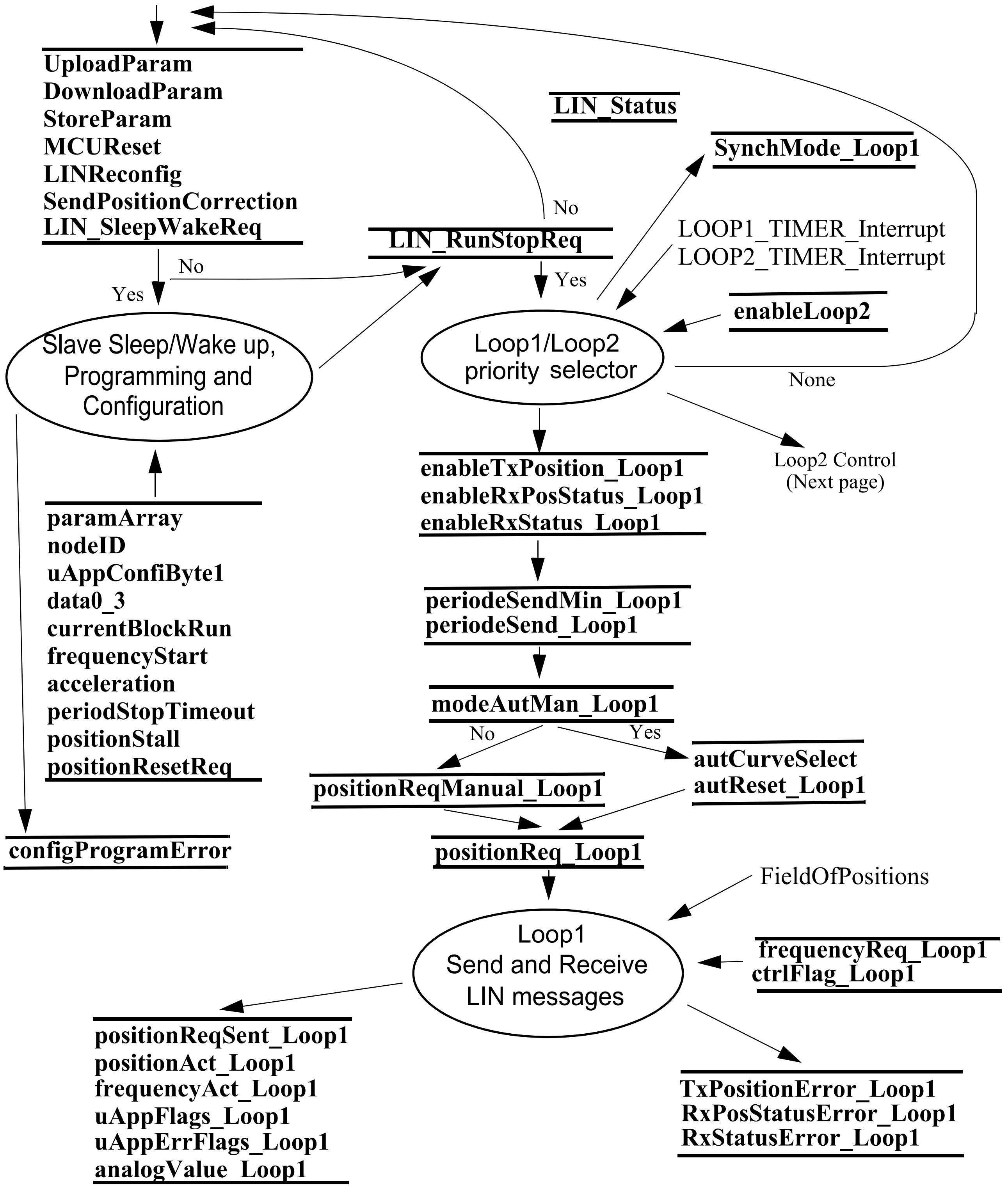


Figure 5-2. PC Master Mode Data Flow Chart - Part1

5.2.4 Loop1/Loop2 Send and Receive LIN Messages

Replaces function that provides (see **Section 5. LIN Master Software Description**):

- Send *frmPosCmd*
- Receive *frmPosStatus* and *frmAppStatus*

All variables surrounding this bubble are described in **Section 7.3. LIN-bus Control**.

5.2.5 Error Handling

Modules providing communication by the LIN-bus have incorporated error handling in terms of recognizing the presence or absence of the LIN Stepper Controller, and of no possibility of transmission (the LIN SIO wire is held by the supply source, e.g. shorted to ground or to supply wire).

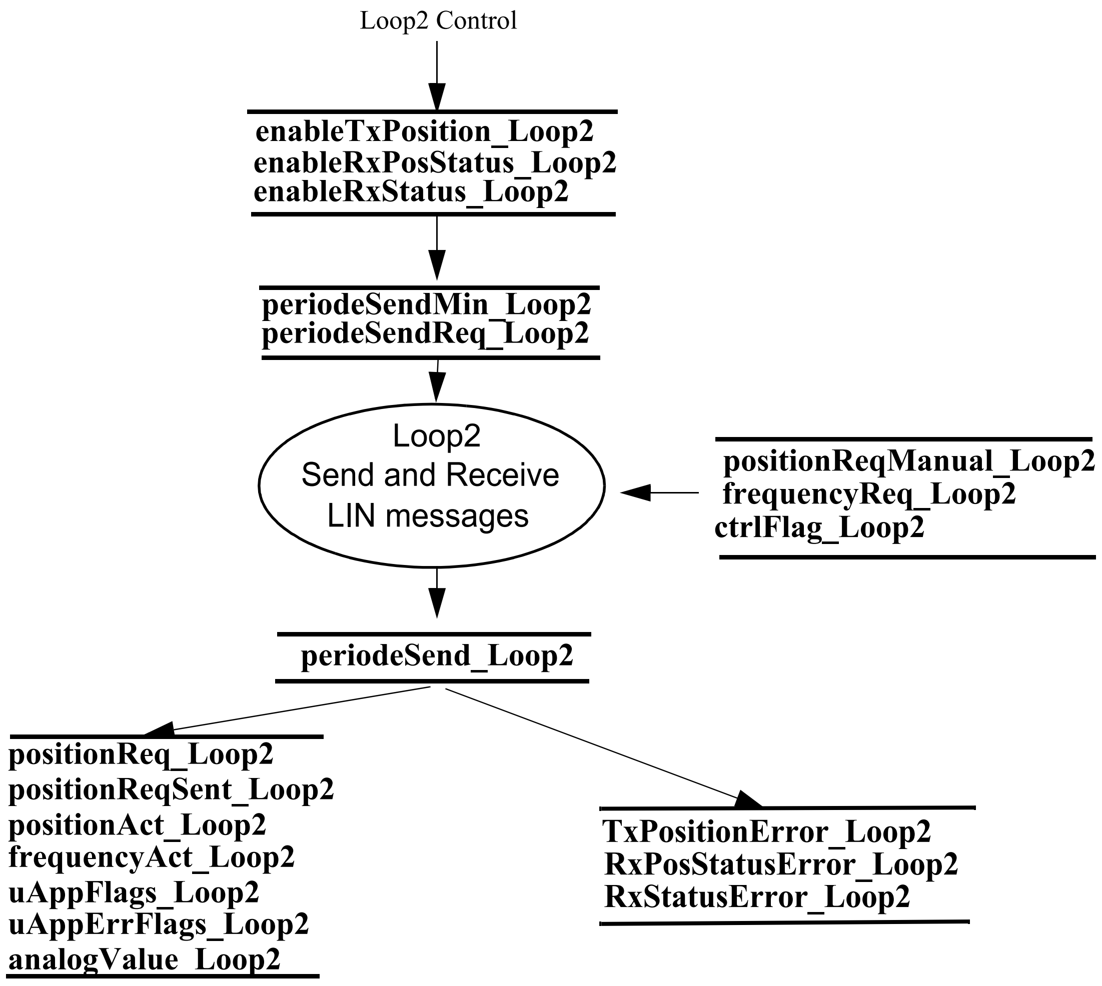


Figure 5-3. PC Master Mode Data Flow Chart - Part2

5.2.6 Master Mode Program Flow

Provides actions described in **Section 2.3.2. Master Mode**. From **Figure 5-4**, it can be seen that in this mode the functions dedicated to Loop1 (*Timing* and *Send and Receive LIN messages*) are successfully applied. These are used in PC master mode.

5.2.7 Loop1 Timing

Sets ECT channel1 output compare register and waits for interrupt; set by means of *periodeMasterMode* variable to 20ms.

5.2.8 Loop1 Send and Receive LIN Messages

Send *frmPosCmd,* receive *frmPosStatus* and *frmAppStatus* (see **Section 5. LIN Master Software Description**). All data in transmitted frames are predefined in MCU memory, including the required position of the LIN Stepper Controller HID lamp, which is the read-out from *FieldOfPossition* array - curve SLOW-FAST.

5.2.9 Error Handling

See **Section 5.2.5. Error Handling**

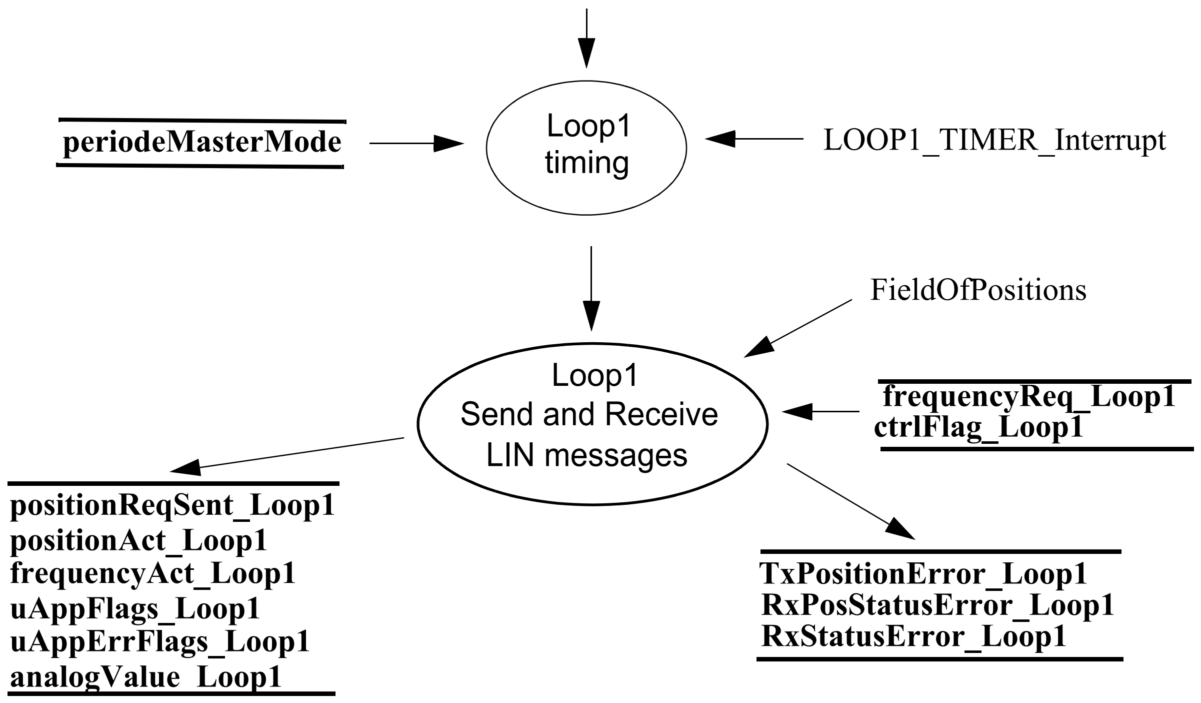


Figure 5-4. Master Mode Data Flow Chart

5.2.10 Debug Mode Program Flow

A general overview may be gained from **Section 2.3.3. Debug Mode** and **Figure 5-5**.

5.2.11 Debug/Programming Control

The Master Board hardware (see **Appendix A. Hardware Schematics**) is prepared for using RxD, TxD, DTR and RTS RS232 signals, but software implementation is based on RxD, TxD, and DTR signals, where the DTR state is the determining function, i.e. either Debugging / Programming (DTR - low level, LIN supply voltage on) or Reset (DTR - high level, LIN supply voltage off). Similarly, on the Debug line output, where the RSTB signal is not used, hardware allows it. **Table 5-1** shows the Debug line pin assignments and the relationship to the RS232 line signals.

Table 5-1. Debug Line Output

Pin number	Pin name	Pin type	Master Board signals	
			DTR - High level (Reset)	*DTR - Low level (Debugging and programming)*
1	CLOCK	Output	High impedance state	19,6608MHz clock
2	GND	Power	Ground	Ground
3	VDD	Output	Connected to ground (discharging capacitors of Slave Board -> push supply voltage line to 0V)	Open collector output (transistor is turned off)
4	\IRQOUT	Output	\IRQIN low level -> 0V \IRQIN high level -> +5V	\IRQIN low level -> 0V \IRQIN high level -> +9V
5	\IRQIN	Input	+5V (pull-up)	+5V (pull-up)
6	RSTB	Input (not used)	0V (pull-down)	0V (pull-down)
7	DATA	Bidirectional	TxD signal is held in low level	Communication is opened, RxD and TxD signals are held in TTL levels
8	MON1	Output	High impedance state	0V (Set by JP1 on Master Board)

Table 5-1. Debug Line Output

Pin number	Pin name	Pin type	Master Board signals	
			DTR - High level (Reset)	*DTR - Low level (Debugging and programming)*
9	MON2	Output	High impedance state	0V (Set by JP2 on Master Board)
10	MON3	Output	High impedance state	+5V (Set by JP3 on Master Board)

Debug mode functionality was successfully tested at frequencies up to 20 kHz on the DATA pin.

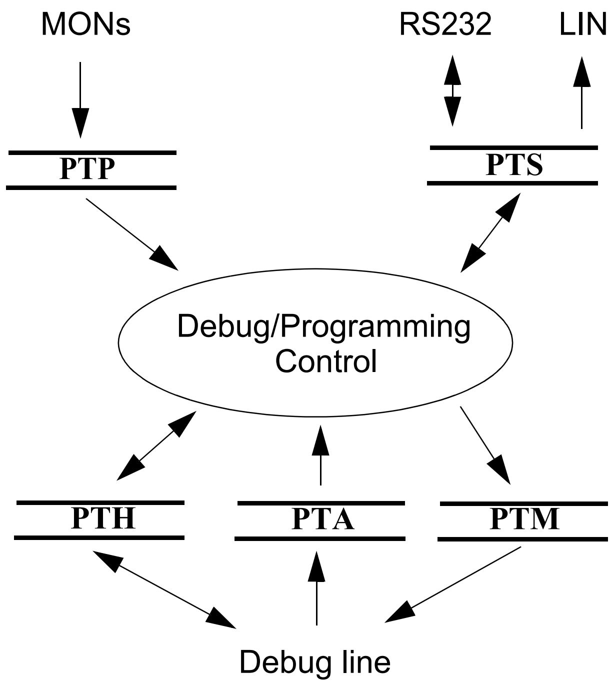


Figure 5-5. Debug Mode Data Flow Chart

5.2.12 Pass Mode Program Flow

Is based on an endless software loop, that copies the RxD signal (RS232) to the LIN SIO wire and then copies the LIN SIO wire signal back to TxD (RS232) (see **Figure 5-6**). Functionality of the *Exchange* *data* procedure was successfully tested with data speeds up to 20 kBaud.

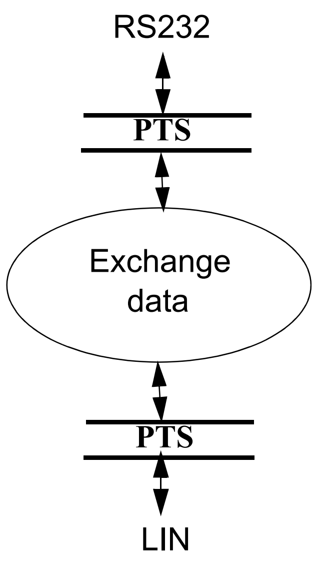


Figure 5-6. Pass Mode Data Flow Chart

Section 6. LIN Stepper Software Description

LIN Stepper Software is described in terms of:

- **LIN Stepper Software Data Flow**
- **LIN Stepper Software Application State Diagram**
- Flow Charts
- **LIN Stepper Software Implementation**

on the following pages

6.1 LIN Stepper Software Data Flow

Figure 6-1 and **Figure 6-5** show the slave software data flow. It consists of the processes described in following subsections.

The slave application control is processed according to LIN messages from the LIN driver. According to received messages x_rd_y, the application variables and states are set. The LIN messages are described in **Section 5. LIN Master Software Description**.

Detailed descriptions of the data variables are in **Appendix D. LIN Stepper Software Data Variables**. A description of the application processes starts here.

6.1.1 Slave Application Control

The Slave Application Control is the software top level which determines other processes states. It controls the application states according to **Figure 6-6**. The **eAppState** variable reflects the application control state. The Slave Application control process also interprets the LIN signals and sets some Power Die variables using the SPI Driver.

The other process's states are controlled by functional calls from the Slave Application control process and by dedicated variables. Variables **uAppFlags1** and **uAppErrFlags** reflect the system status and are set and read by the three essential processes. The structure **sParameterRAM** includes the system parameters that determine the application behavior. The components of the **sParameterRAM** structure can be changed using parameter configuration (see **Section 6.1.8. Config Param**). Required position **positionReq** is derived from **l_u16_rd_positionReq** signal and is used for Position and Speed control process. Similarly, **frequencyReq is** created from **l_u8_rd_frequencyReq**. Actual position **positionAct** from the Position Sensing process is passed to the LIN message **l_u16_wr_positionAct** signal. Similarly, for **frequencyActLowHigh** and **l_u16_wr_frequencyAct**.

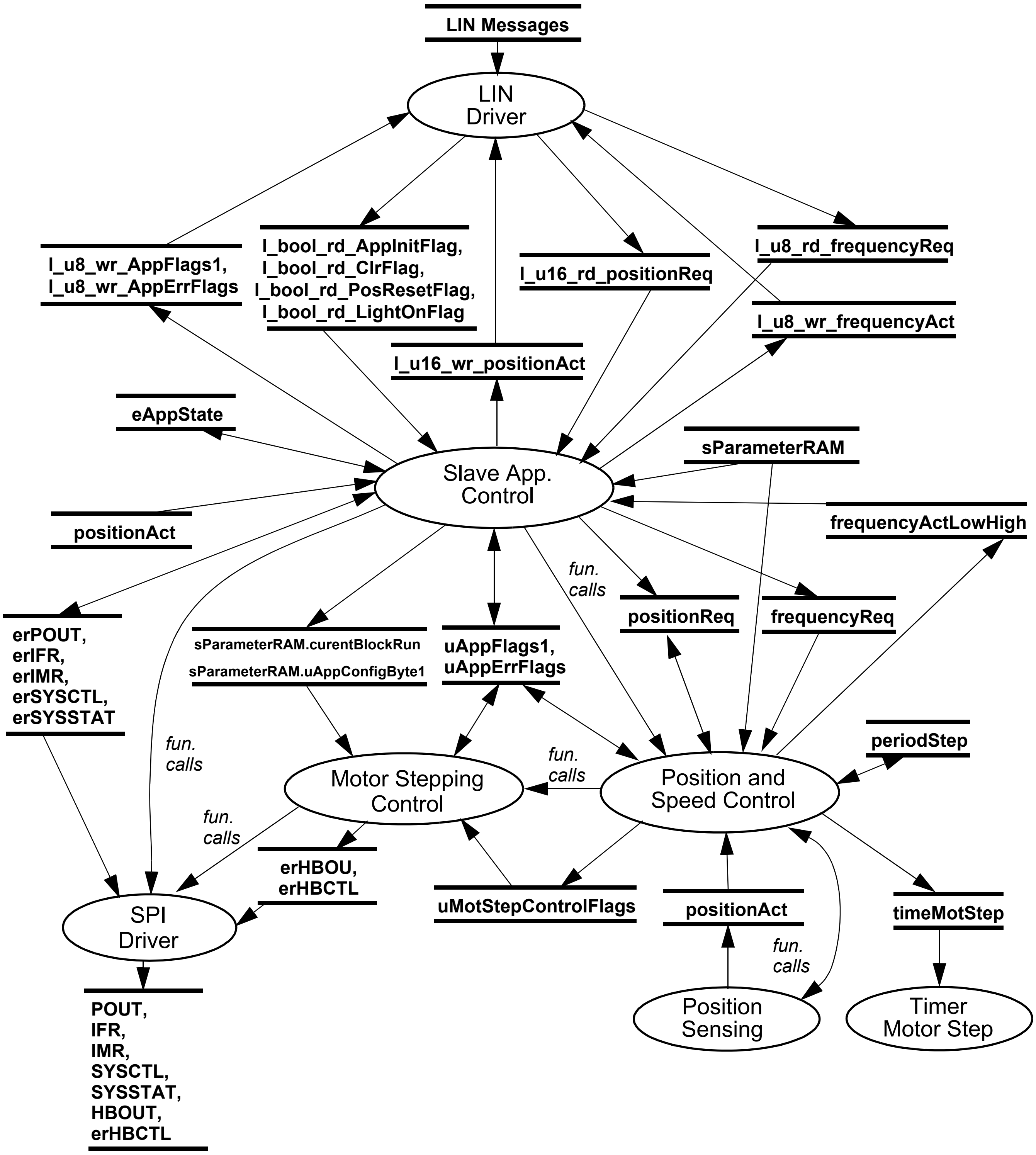


Figure 6-1. LIN Stepper Software - Data Flow 1

6.1.2 Position and Speed Control

The Position and Speed Control provides stepper motor control to a defined absolute position **positionReq** with the following functions:

- linear acceleration and deceleration ramps from **frequencyStart** to **frequencyReq** with ramp steepness
  acceleration = deceleration
- handles changes after required **positionReq** position update
- handles changes after actual position **positionAct** update
- handles changes after required frequency **frequencyReq** update
- ads time instant **PERIOD_STOP_TIMEOUT_MS** after deceleration ramp

The functionality of the Position and Speed Control Processes can be explained using **Figure 6-2**, **Figure 6-3**, and **Figure 6-4**.

NOTE: *Speed (frequency) control is based on the fact that identical constant acceleration is used for acceleration as for deceleration. Therefore, the number of steps for speed acceleration is almost the same as for speed deceleration.*

1. Before the motor stops, the speed must be at **frequencyStart**, in order not to lose the position.

*In addition, according to dynamic behav*ior *(oscillation), the motor is better stabilized at stop if it provides few steps with the frequencyStart speed.*

2. Therefore we apply the reserve constant DECEL_AFTERRAMP_RESERVE.

*The last issue is a numerical reserve DECEL_NUMERICAL_RESERVE. Because acceleration and deceleration ramps are calculated from previous commutation step reserve, the number of steps for acceleration ramp is higher than deceleration ramps steps. In order to fulfil condition **1**, there is*

3. DECEL_NUMERICAL_RESERVE

The variable ***positionDecelDistance*** *is incremented during speed acceleration. The actual and required position difference is compared with the* ***positionDecelDistance*** *(with the* DECEL_RESERVE*) to find the point where the speed deceleration ramp must start down to frequencyStart.*

$$\text{DECEL\_RESERVE} = \text{DECEL\_AFTERRAMP\_RESERVE} + 2 + \text{DECEL\_NUMERICAL\_RESERVE}$$

where:

DECEL_RESERVE	overall deceleration ramp reserve [steps]
DECEL_AFTERRAMP_RESERVE	number of steps with starting frequency after deceleration ramp [steps]
2	reserve for following acceleration requires1 acceleration step and 1 deceleration step [steps]
DECEL_NUMERICAL_RESERVE	deceleration reserve to cover the difference between acceleration and deceleration ramps caused by numerical rounding [steps]

Service of the Updated Requests is shown in **Figure 6-2**. It provides the necessary Position and Speed control functionality when the position request value is updated (by LIN message).

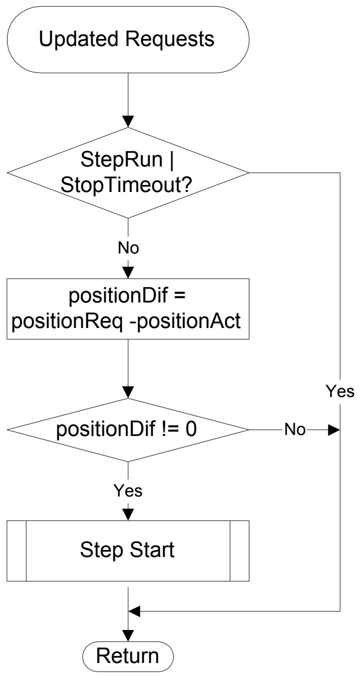


Figure 6-2. Motor Position and Speed Control - Service Updated Requests

Service of the Updated Actual Position is shown in **Figure 6-3**. It provides the necessary Position and Speed control functionality when the actual position value is updated (step done) or when stop timeout is scheduled.

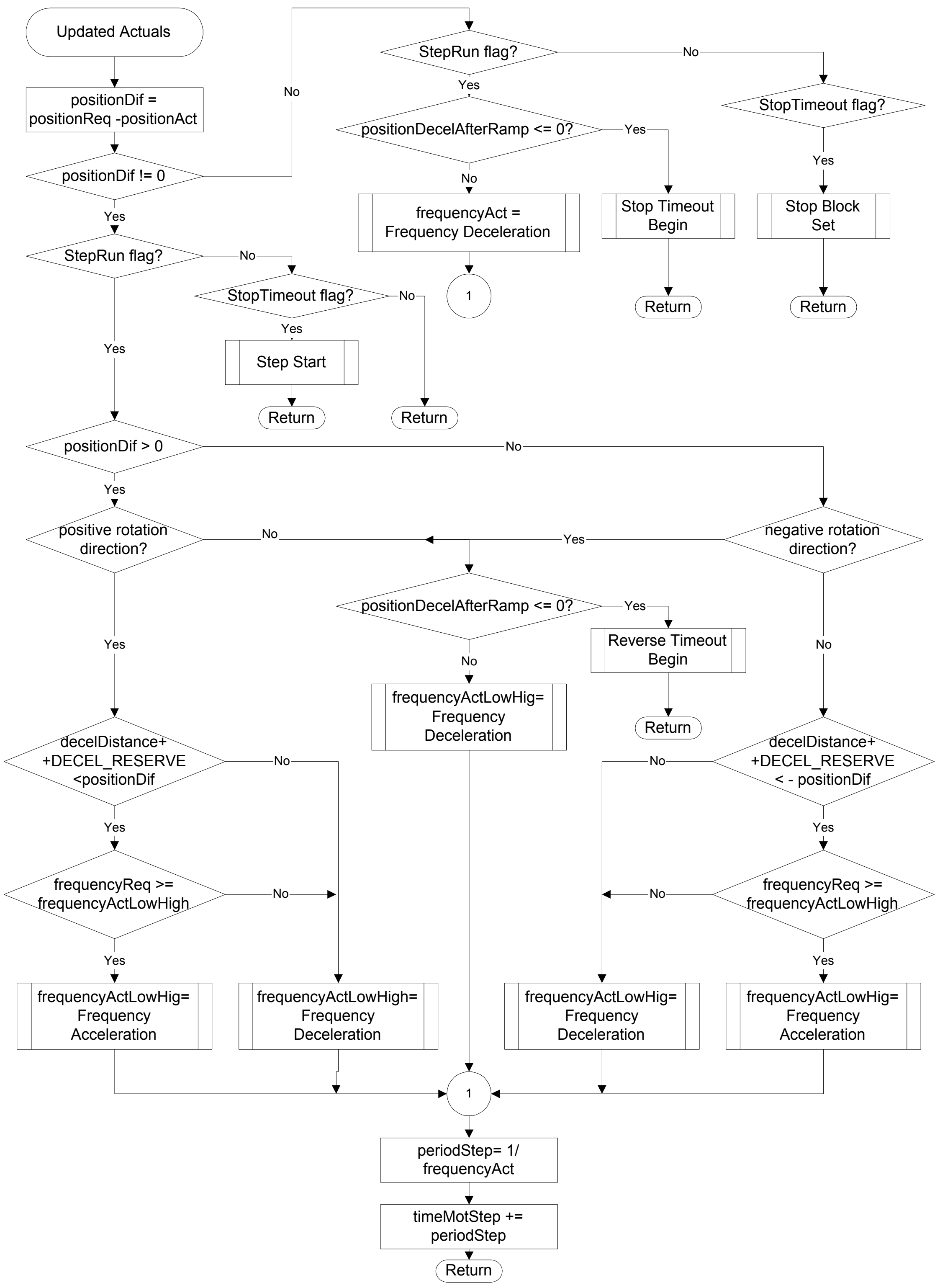

Updated Actuals
positionDif = positionReq -positionAct
positionDif != 0
No
Yes
StepRun flag?
No
Yes
StopTimeout flag?
No
Yes
Step Start
Return
Return
StepRun flag?
No
Yes
positionDecelAfterRamp <= 0?
Yes
No
frequencyAct = Frequency Deceleration
1
Stop Timeout Begin
Return
StopTimeout flag?
Yes
Stop Block Set
Return
positionDif > 0
No
Yes
positive rotation direction?
No
Yes
negative rotation direction?
Yes
No
positionDecelAfterRamp <= 0?
Yes
No
Reverse Timeout Begin
Return
frequencyActLowHig= Frequency Deceleration
decelDistance+ +DECEL_RESERVE <positionDif
No
Yes
decelDistance+ +DECEL_RESERVE < - positionDif
No
Yes
frequencyReq >= frequencyActLowHigh
No
Yes
frequencyReq >= frequencyActLowHigh
No
Yes
frequencyActLowHig= Frequency Acceleration
frequencyActLowHigh= Frequency Deceleration
frequencyActLowHig= Frequency Deceleration
frequencyActLowHig= Frequency Acceleration
1
periodStep= 1/ frequencyAct
timeMotStep += periodStep
Return

Figure 6-3. Motor Position and Speed Control - Service Updated Actual Position

NOTE: *To stop or reverse the motor, the motor must slow down to frequencyStart and provides DECEL_AFTERRAMP_RESERVE steps with frequencyStart. This is provided by the condition positiondecelAfterRamp =< 0. When positiondecelAfterRamp is initialized to DECEL_AFTERRAMP_RESERVE by FrequencyAcceleration routine (see **Figure 6-4**).*

The speed acceleration and deceleration subroutines are used for linear acceleration and deceleration ramping. They are shown in **Figure 6-4**.

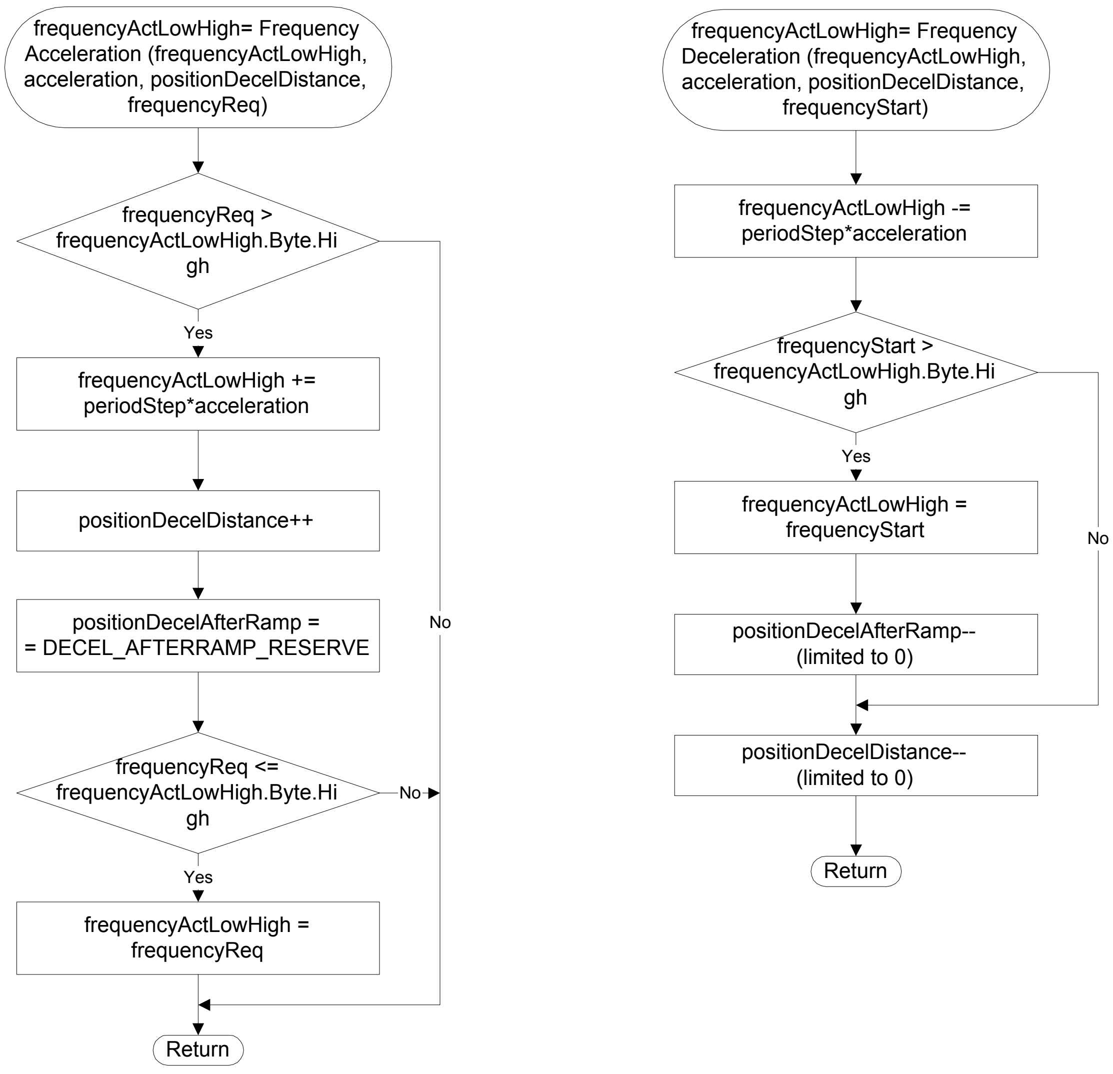

frequencyActLowHigh= Frequency Acceleration (frequencyActLowHigh, acceleration, positionDecelDistance, frequencyReq)
frequencyReq > frequencyActLowHigh.Byte.High
Yes
No
frequencyActLowHigh += periodStep*acceleration
positionDecelDistance++
positionDecelAfterRamp = = DECEL_AFTERRAMP_RESERVE
frequencyReq <= frequencyActLowHigh.Byte.High
No
Yes
frequencyActLowHigh = frequencyReq
Return
frequencyActLowHigh= Frequency Deceleration (frequencyActLowHigh, acceleration, positionDecelDistance, frequencyStart)
frequencyActLowHigh -= periodStep*acceleration
frequencyStart > frequencyActLowHigh.Byte.High
Yes
No
frequencyActLowHigh = frequencyStart
positionDecelAfterRamp-- (limited to 0)
positionDecelDistance-- (limited to 0)
Return

Figure 6-4. Frequency Acceleration and Deceleration - Flow Chart

The frequency acceleration subroutine provides the actual frequency **frequencyActLowHigh** ramp with the maximum frequency limit at **frequencyReq**. If the actual frequency is lower than the required, the new actual frequency is calculated from the last actual frequency **frequencyActLowHigh**, last stepping period **periodStep** and acceleration constant **ParametersRAM.acceleration.** After that the **positionDecelDistance** is incremented. (This value is used in the Position and Speed Control Process to determine the point where the speed deceleration starts). Then the **positionDecelAfterRamp** variable is initialized. (positionDecelAfterRamp is used to guarantee that the motor stop after ramp down to **frequencyActLowHigh** as shown in Figure 6-3). Finally **frequencyActLowHigh** is limited to **frequencyReq**.

The frequency deceleration subroutine provides the actual frequency **frequencyActLowHigh** deceleration ramp with the minimum frequency limit at **ParametersRAM.frequencyStart**. First new decreased actual frequency is calculated from the last actual frequency **frequencyActLowHigh**, last stepping period **periodStep** and acceleration constant **ParametersRAM.acceleration**. If after the deceleration the actual frequency **frequencyActLowHigh** is below minimum starting frequency **ParametersRAM.frequencyStart**, then it is limited to **ParametersRAM.frequencyStart** and variable **positionDecelAfterRamp** is decremented as shown in **Figure 6-3**, the motor is allowed to stop after the variable is zero). Finally **positionDecelDistance** is incremented (the speed deceleration start point changes with the actual speed).

6.1.3 Motor Stepping Control

The process controls the motor stepping. It prepares the **erHBOUT** variable and, using the SPI Driver calls, provides the Power Die H-bridge setting to energize the stepper motor coils. The process also controls current limitation via **erHBCTL**. The motor stepping control is provided according to the control flags in registers **uMotStepControlFlags**, **sParameterRAM.uAppConfigByte1,** and using the **stepIndex** as a pointer to tables **fullStepTable** or **halfStepTable**.

6.1.4 Position Sensing

Position sensing handles the actual position variable **positionAct**. The process should guarantee that the actual position is actualized (increment/decrement) each motor step provided by the Motor Stepping Control Process.

The process also handles position initialization and correction.

NOTE: *The stepper motor is controlled as an open loop system in the application described in this document. So the position is updated each motor step according to the motor stepping direction. If any kind of position sensing or position initialization (e.g. a Hall sensor with a defined position) is used, the current software could be adapted simply by changing of the Position Sensing process.*

6.1.5 LIN Driver

LIN drivers provide all the processes for the LIN-bus protocol, which handles the transmitting/receiving of LIN frames. The application software communicates with the drivers using LIN API as shown in **Figure 6-1**. The LIN messages are described in the **Section 5. LIN Master Software Description**.

6.1.6 SPI Driver

The MCU part of the 908E625 device communicates with the Power Die using the SPI module (See **Section 9. References**, **9**, **10**).

6.1.7 Timer Motor Step

The Timer Motor Step sets the timer for the The Motor Stepping Control and Motor Position and Speed Control processes.

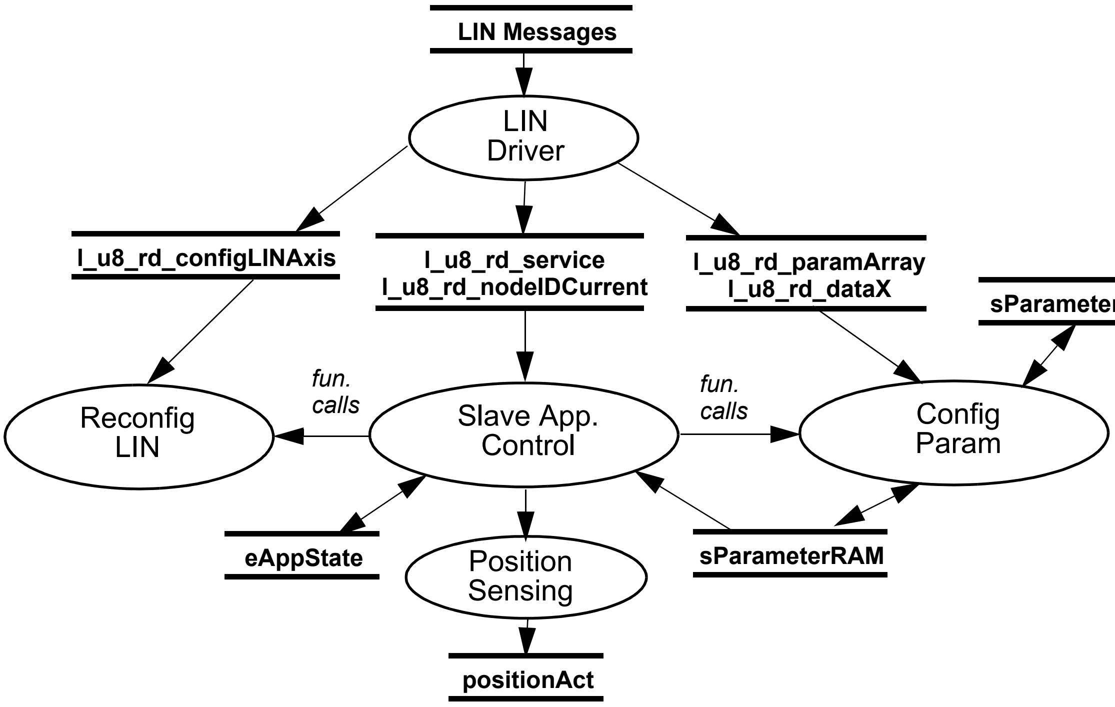


Figure 6-5. LIN Stepper Software - Data Flow 2 - Configuration

6.1.8 Config Param

The process includes the functions necessary for parameter configuration. The functions are:

- parameter transfer from FLASH ROM to parameter RAM
- parameter upload from parameters RAM to LIN l_u8_wr_dataX signals according to l_u8_rd_paramArray
- parameters download from l_u8_rd_dataX signals to parameter RAM according to l_u8_rd_paramArray
- FLASH programming parameters from RAM to FLASH ROM

6.1.9 Reconfig LIN

The process includes the functions necessary for LIN signal reconfiguration.

As shown in **Section 5. LIN Master Software Description**, each LIN Stepper Controller node is programmed as an acceptor for one set of the LIN signals according to the controlled Axis. The LIN-bus signalling scheme can be reconfigured to a different Axis

The Reconfig LIN process automatically provides reconfiguration with the following steps:

- Initializes the LIN reconfiguration RAM buffer fLINReconfigBufRAM
- Presets the LIN reconfiguration RAM buffer according to **l_u8_rd_configLINAxis** message
- Finally the LIN reconfiguration RAM buffer fLINReconfigBufRAM is programmed into the LIN configuration tables in FLASH ROM

6.2 LIN Stepper Software Application State Diagram

Figure 6-6 shows the application states. After an MCU reset, the MCU Initialization states provide the initialization of all processes.

6.2.1 MCU Init

In this state, the application provides all system module initialization after reset:

- sets CONFIG2 register
- sets clock module to 4.9152 MHz bus frequency
- copies configuration parameters from FLASH memory sParameterRAM = sParameterROM
- initializes uAppFlags1.Byte = 0; uAppErrFlags.Byte = 0
- initializes SPI

- initializes LIN drivers
- initializes stepIndex = STEP_INDEX_INIT_DEFAULT
- initializes actual position
  positionAct = sParameterRAM.positionPark

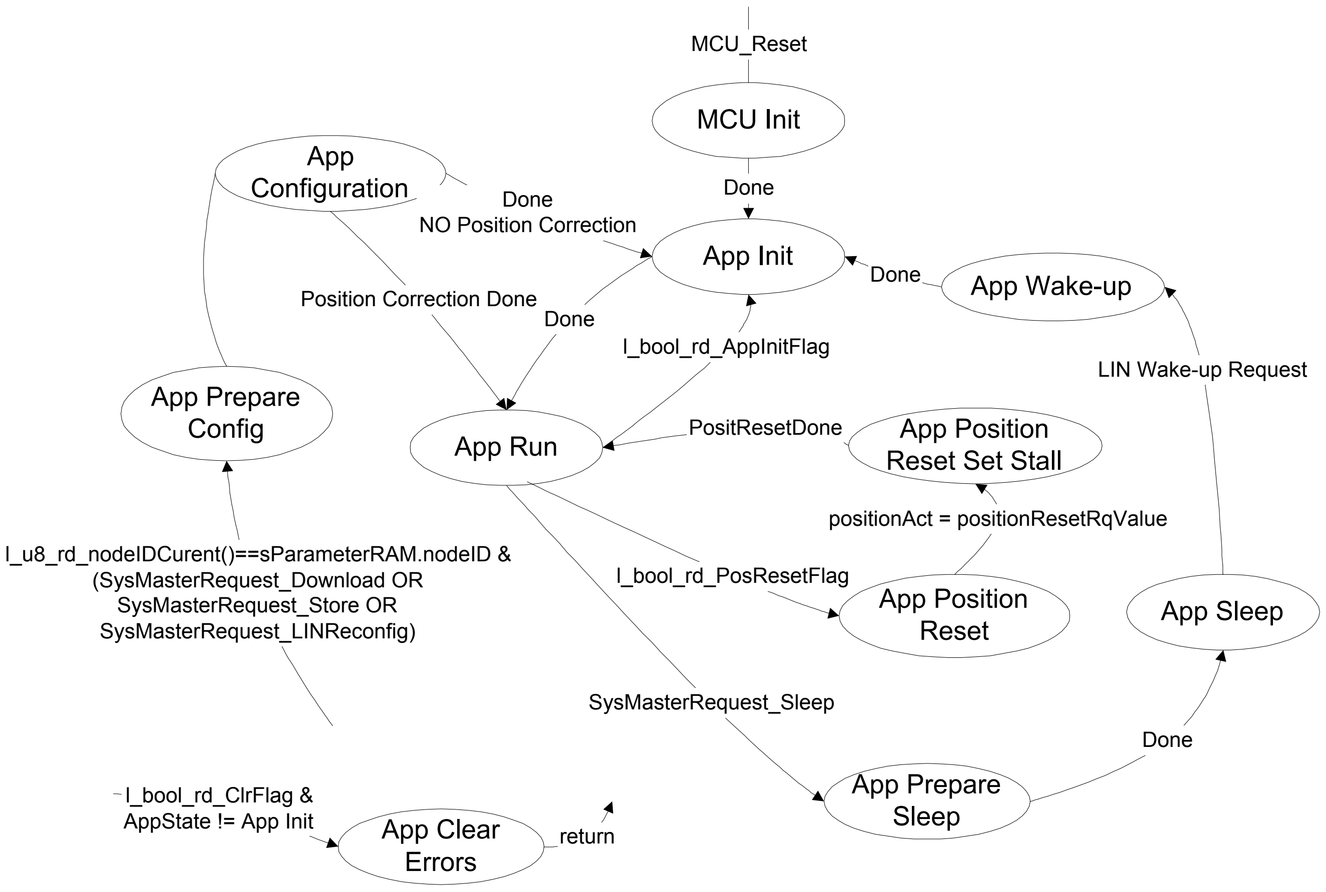


Figure 6-6. LIN Stepper Software Application State Diagram

6.2.2 iApp.Init

The Application Initialization state is a defined state which the application enters after MCU Init, or can be forced by the l_bool_rd_AppInitFlag message. The application performs the following:

- initializes timers

- clears system errors
- provides position and speed control application initialization

NOTE: *The application initialization state App. Init. does not set actual motor position positionAct.*

6.2.3 App.Run

In the Application Run state, the application controls the actual position **positionAct** to the required position **positionReq** with current limited to the Run or Block current. The **positionReq** is set according to the to **l_u16_rd_positionReq** signal. The motor is in the block state if the **positionAct** = **positionReq** after stop timeout. The LIN signals are also tested in this state. If there are any of the requests shown in **Figure 6-6**, the state is left.

6.2.4 App.Position Reset

Functionality of the Position Reset state is very similar to the App Run state, but in this state the application controls the motor to the motor down to **positionReq** = **sParameterRAM.positionResetRqValue**. So the **positionReq** is not set according to the l_u16_rd_positionReq.

This state is used with the App Position Reset Set Stall for position initialization.

6.2.5 App.Position Reset Set Stall

Provides setting **positionAct** = **sParameterRAM.positionStall** and so provides the actual position reset.

6.2.6 App. Prepare Sleep

Application prepares for sleep. In case the motor is spinning it is decelerated and stopped. This is provided in order not to lose position by going to sleep.

6.2.7 App. Prepare Sleep

Application prepares all MCU modules for minimal consumption and provides Stop instruction. It also sets the Power Die to be able to generate IRQ pin signal for LIN-bus wake-up.

6.2.8 App. Wake-Up

After wake-up message from the bus, the MCU is wakened by the IRQ pin signal from Power Die.

6.2.9 App. Configuration

Application Configuration is described in **Section 6.1.8. Config Param**, **Section 6.1.9. Reconfig LIN** or **Appendix C.3. LIN Leveller Configuration Frames**.

6.2.10 App. Clear Errors

Application clears Power Die errors and uAppErrFlags.Byte = 0

6.3 LIN Stepper Software Implementation

6.3.1 Scaling of Quantities

The LIN Leveller application uses signed 16-bit type **SWord16**, unsigned 16-bit type **UWord16**, signed 8-bit type **SByte,** and unsigned 8-bit type UByte variables.

Any defined Real Variable constant must be recalculated to the system representation (**UWord16 Value, SWord16 Value, UWord8 Value, SWord8 Value**)

The following equation shows the relationship between system (raw value range) and real (physical or normalized range) representation

$$\text{UWord16 Value} = (\text{MAX\_U16} + 1)\frac{\text{Real Value}}{\text{Real Quantity Range Max}} \quad \text{(EQ 1.)}$$

$$\text{SWord16 Value} = (\text{MAX\_S16} + 1) \cdot \frac{\text{Real Value}}{\text{Real Quantity Range Max}} \quad \text{(EQ 2.)}$$

$$\text{UByte Value} = (\text{MAX\_U8} + 1) \cdot \frac{\text{Real Value}}{\text{Real Quantity Range Max}} \quad \text{(EQ 3.)}$$

$$\text{SByte Value} = (\text{MAX\_S8} + 1) \cdot \frac{\text{Real Value}}{\text{Real Quantity Range Max}} \quad \text{(EQ 4.)}$$

where:

UWord16 Value is the unsigned 16-bit system representation of the real value,

SWord16 Value is the signed 16-bit system representation of the real value,

UWord8 Value is the unsigned 8-bit system representation of the real value,

SWord8 Value is the signed 8-bit system representation of the real value,

Real Value is the real value of the quantity [V, A, RPM, etc.],

Real Quantity Range Max is the maximum of the quantity range, defined in the application [V, A, RPM, etc.],

MAX_U16 = 65535 is the maximum of the unsigned 16-bit variable,

MAX_S16 = 32768 is the maximum of the signed 16-bit variable,

MAX_U8 = 255 is the maximum of the unsigned 8-bit variable,

MAX_S8 = 127 is the maximum of the signed 8-bit variable.

From the above equations for the Real Value:

$$\text{Real Value} = \frac{\text{UWord16 Value*Real Quantity Range Max}}{(\text{MAX\_U16} + 1)} \quad \text{(EQ 5.)}$$

$$\text{Real Value} = \frac{\text{SWord16 Value*Real Quantity Range Max}}{(\text{MAX\_S16} + 1)} \quad \text{(EQ 6.)}$$

$$\text{Real Value} = \frac{\text{UByte Value*Real Quantity Range Max}}{(\text{MAX\_U8} + 1)} \quad \text{(EQ 7.)}$$

$$\text{Real Value} = \frac{\text{SByte Value*Real Quantity Range Max}}{(\text{MAXSU8} + 1)} \quad \text{(EQ 8.)}$$

According to the variable type, the equations EQ1 to EQ8 also can be converted to the EQ9 and EQ11

$$\text{System Variable X} = \frac{\text{Real Value X}}{\text{Resolution Quantity X}} \quad \text{(EQ 9.)}$$

where:

$$\text{Resolution Quantity X} = \frac{\text{Real Quantity X Range Max}}{\text{MAX Type Variable X}} \quad \text{(EQ 10.)}$$

where:

System Variable X - is the system variable

Real Value X - is the physical value of the quantity represented by Variable X

Resolution Quantity X - resolution of the system variable X in real. (It represents the physical value represented by LSB bit of the Variable X)

Max Type Variable X - system Variable X maximum according to the Variable Type:

Type **UWord16 MAX_U16** = 65535
Type **SWord16 MAX_S16** = 32768
Type **UByte MAX_U8** = 255
Type **SByte MAX_S8** = 127

And hence the Real Value:

$$\text{Real Value X} = \text{Resolution Quantity X*System Variable X} \quad \text{(EQ 11.)}$$

6.3.2 Scaling of Time

Scaling of time variables is according to EQ12,13

$$\text{Resolution Period} = \frac{\text{Timer Prescaller Division}}{\text{Bus Frequency}} \qquad \text{(EQ 12.)}$$

ResolutionPeriod defines the time of

$$\text{Real Step Period} = \text{periodStep}*\text{Resolution Period} \qquad \text{(EQ 13.)}$$

6.3.3 Acceleration Scaling and Arithmetic Operations

Some of the arithmetic operations used in the LIN Stepper software impact the calling.

A good example is umul_16x8_macro

$$\text{UWord16 Variable3} = \frac{\text{UWord16 Variable1}*\text{UByte Variable2}}{256} \qquad \text{(EQ 14.)}$$

which is used for acceleration ramp calculation:

$$\text{Frequency Difference} = \text{SteppingPeriod}*\text{Acceleration} \qquad \text{(EQ 15.)}$$

And EQ15 can be rewritten to

(EQ 16.)

$$\text{Resolution Frequency}*(\text{UWord16})\ \text{frequencyDif} = \text{Resolution Period}*(\text{UWord16})\ \text{periodStep}*\text{Resolution Acceleration}*\text{acceleratior}$$

if we use the umul_16x8_macro

$$(\text{UWord16})\text{Frequency Dif} = \frac{(\text{UWord16})\text{periodStep}*(\text{UByte})\text{Acceleration}}{256} \qquad \text{(EQ 17.)}$$

Then according to EQ16, EQ17 the Resolution Acceleration is:

(EQ 18.)

$$\text{RESOLUTION\_ACCEL\_DECEL\_S\_S2} = \frac{\text{RESOLUTION\_FREQUENCY\_LOW\_HIGH}}{256.0*(\text{RESOLUTION\_PERIOD\_NS}/1000000000.0)}$$

NOTE: *The actual frequency variable **frequencyActLowHigh** is represented as an union. Then for the acceleration calculations it can be accessed as*

*UWord16 variable **frequencyActLowHigh.Word**. This will guarantee the low resolution (necessary to create low steepness frequency ramps). For the frequency-to-period calculation and comparison with required frequency frequencyReq, it is sufficient to use 8-bit information. This allows to use fast calculations like frequency to speed conversion using udiv16_8to16.*
*Then the high byte **frequencyActLowHigh.Byte.High** is accessed.*

6.3.4 Actual Frequency to Period Conversion

The conversion from frequency to period requires division.

$$\text{Period} = \frac{1}{\text{Frequency}} \tag{EQ 19.}$$

For software execution optimized calculation we wrote a special arithmetic function:

UWord16 udiv16_8to16(UWord16 x, UByte y)

with UWord16 output and UWord16 x, UByte y inputs. It is written in assembler and provides the following calculation:

$$\text{UWord16 Variable3} = \frac{\text{UWord16 Variable1}}{\text{UByte Variable3}} \tag{EQ 20.}$$

The conversion must take into account the variables scaling:

$$\text{Resolution Period}*(\text{UWord16})\ \text{periodStep} = \frac{1}{\text{Resolution Frequency}*(\text{UByte8})\ \text{Frequency}} \tag{EQ 21.}$$

$$(\text{UWord16})\ \text{periodStep} = \frac{(\text{UWord16})(\text{Resolution Frequency/ResolutionPeriod})}{(\text{UByte8})\ \text{Frequency}} \tag{EQ 22.}$$

So the final calculation using the **udiv16_8to16** function is:

$$(\text{UWord16})\ \text{periodStep} = \frac{(\text{UWord16})(\text{CONVERSION\_CONST\_PERIOD\_FERQ})}{(\text{UByte8})\ \text{frequencyActLowHigh.Byte.High}} \tag{EQ 23.}$$

where:

Resolution Period [s] is **RESOLUTION_PERIOD_NS*1000000000.0**

Resolution Frequency [Hz] is **RESOLUTION_FREQUENCY_HZ**

CONVERSION_CONST_PERIOD_FREQ = (1000000000.0/RESOLUTION_FREQUENCY_HZ/RESOLUTION_PERIOD_NS)

6.3.5 LIN Stepper Software Memory Utilization

Table 6-1 shows how much memory is required to run the LIN Stepper Controller with code compiled with Metrowerks CodeWarrior v 2.1.

Table 6-1. Stepper Controller Software Memory Utilization

Memory	Used by Software	Important Sections	Size
FLASH ROM	4039 Bytes	Parameter ROM	16B
		LIN Reconfig ROM	128B
RAM	278 Bytes	Stack	64B
		LIN Reconfig RAM	96B
		Parameter RAM	16B

Section 7. User Interface Description

7.1 Introduction

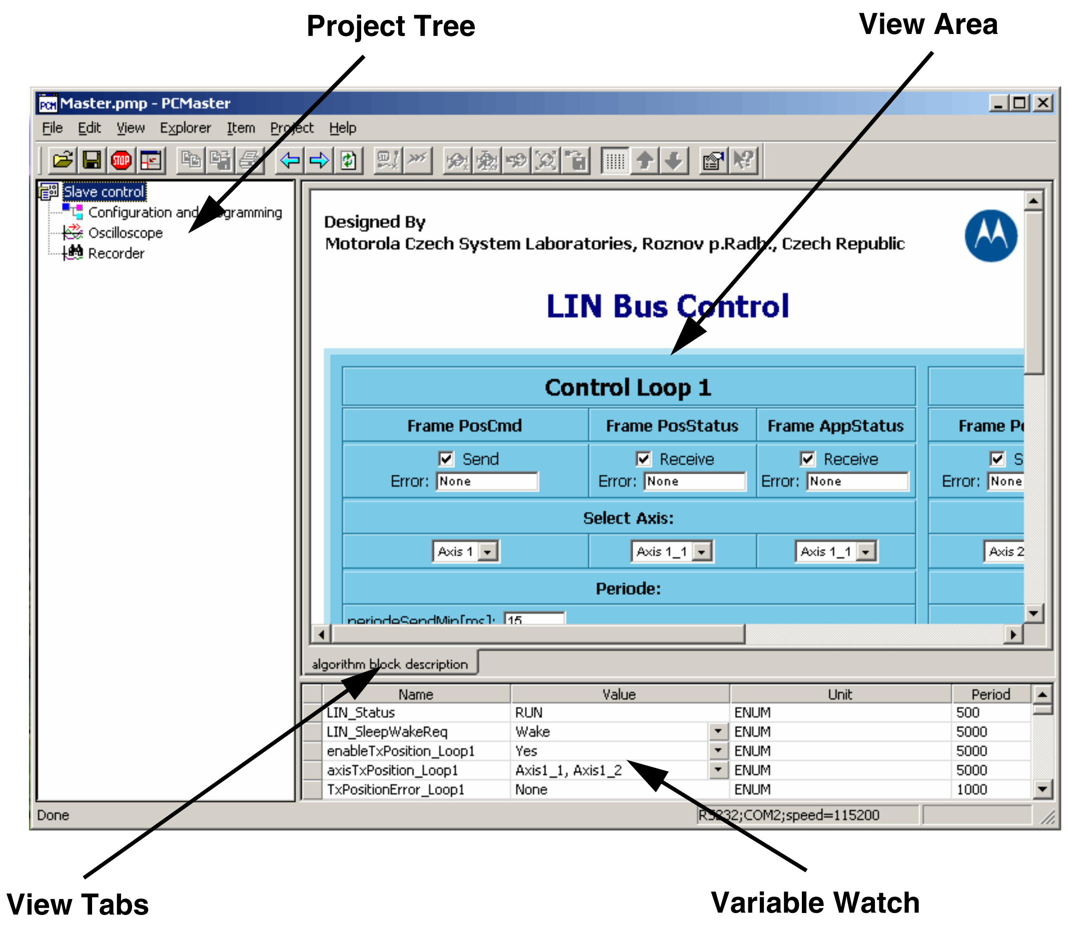
This section describes the Control pages used for LIN-bus control (see **Section 2.3.1. PC Master Mode**) in terms of:

- PC master software general overview
- detailed description of each control page

7.2 PC Master Software General Overview

The principle is briefly shown in **Section 2.4. Personal Computer**.

In **Figure 7-1** it is possible to see whole PC master software page, which is comprised of four main boxes.



Designed By
Motorola Czech System Laboratories, Roznov p.Radh., Czech Republic

LIN Bus Control

Control Loop 1			
Frame PosCmd	Frame PosStatus	Frame AppStatus	Frame P
☑ Send Error: None	☑ Receive Error: None	☑ Receive Error: None	☑ S Error: None
Select Axis:			
Axis 1	Axis 1_1	Axis 1_1	Axis 2
Periode:			
periodeSendMin[ms]: 15			

Name	Value	Unit	Period
LIN_Status	RUN	ENUM	500
LIN_SleepWakeReq	Wake	ENUM	5000
enableTxPosition_Loop1	Yes	ENUM	5000
axisTxPosition_Loop1	Axis1_1, Axis1_2	ENUM	5000
TxPositionError_Loop1	None	ENUM	1000

Figure 7-1. PC Master Software Main Page

7.2.1 Project Tree

Clicking the mouse on the legend selects what is displayed in the remaining boxes.

7.2.2 Variable Watch

Here some selected variables from the Master Board are located, and the Value (Number or Expression) and Period of reloading is assigned to them.

7.2.3 View Tabs

The View tabs select what is displayed in the View area.

7.2.4 View Area

Shows one of following items:

- HTML Control page (LIN-bus Control page or Programming and Configuration page)
- Oscilloscope page
- Recorder page

Because some variables on the HTML control page have write status, it is necessary to reload them to Variable Watch by pressing the F5 key on the keyboard (recommended after a change). Reloading of the read variables is done automatically.

More information about the PC master software can be gained from the PC Master Software User Manual (**Section 9. References**, 1).

7.3 LIN-bus Control

The pages dedicated to LIN slave control are (see Project Tree):

- Slave control - LIN-bus control page
- Oscilloscope - real time variables watching
- Recorder - recorded variables watching

7.4 Slave Control

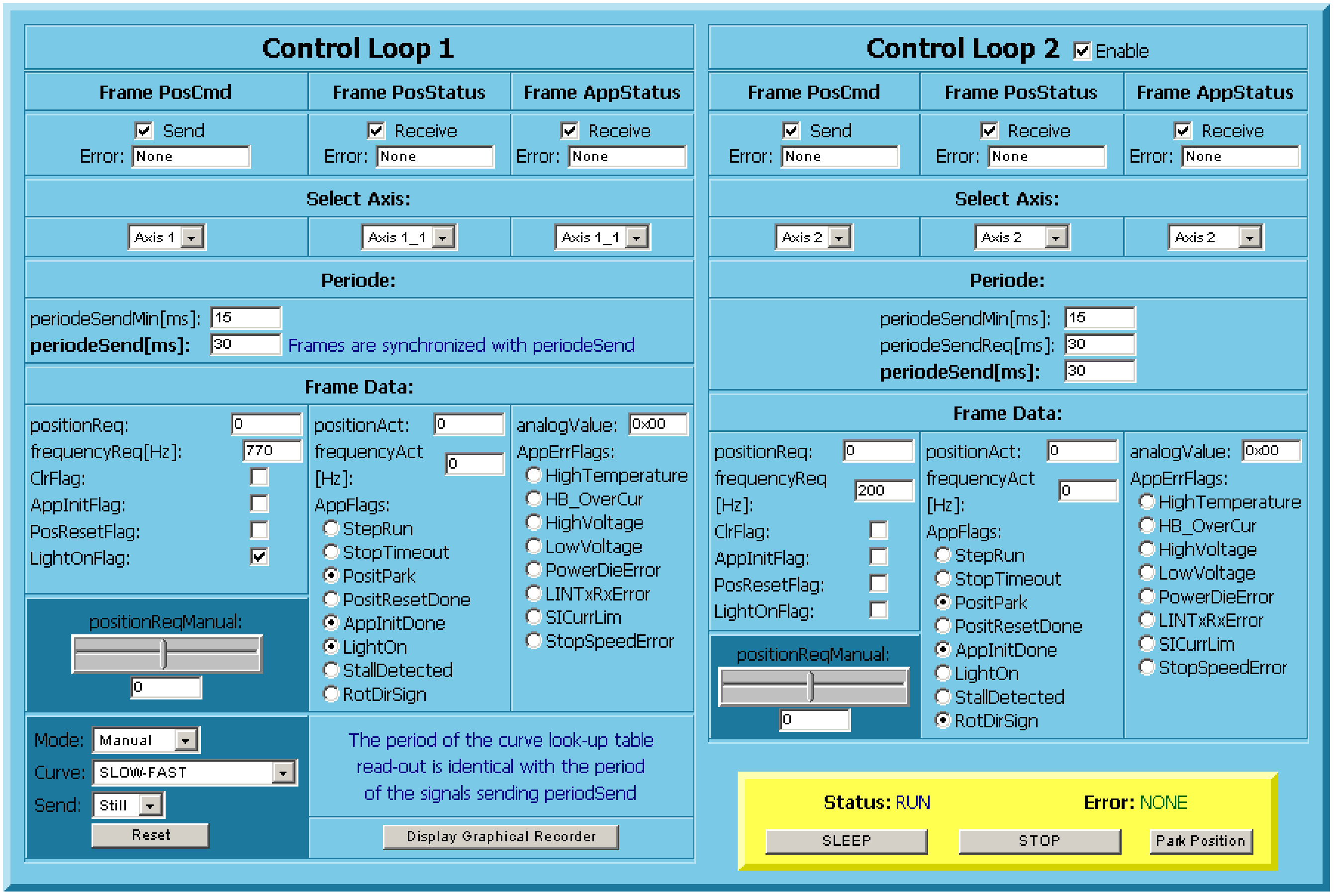
Via this HTML page (**Figure 7-2**), it is possible to fully control up to two LIN Stepper Controllers. The page comprises three main areas: Control Loop1, Control Loop2, and an area containing Status notes and State buttons on a yellow background.

7.4.1 Control Loop1

The steps for setting this page are as follows:

1. Chose LIN frames (see **Section 5. LIN Master Software Description**) for communication via checking of Send or Receive boxes.
2. Select Axis (LIN Stepper Controller)
3. The *periodeSendMin* box displays the automatically calculated minimum time necessary for transmitting and receiving selected LIN frames. Set time distances in *periodeSend* box between two communication events greater than or equal to that minimum time (displayed in milliseconds; maximum value = 255).
4. All LIN frames data are displayed in the *Frame Data* box. In *FramePosCmd* it is necessary to choose the source for *positionReq* 16-bit data. Either it will be driven manually or will be automatically generated from predefined curves in LIN Master memory. In the first case, choose Manual in the *Mode* combo box, and *positionReq* is driven by *positionReqManual.* The actual value can be seen either in the box below the slide bar or in the *positionReq* box. In the second case, choose Automatic in the *Mode* combo box, select curve and set “if generate data still” or “read out curve data only once and then stop” (*Send* combo box). The *Reset* button causes shifting in curve data to beginning.

There are four predefined curves and one special choice called SQUARE.



Control Loop 1

Frame PosCmd: [✓] Send, Error: None
Frame PosStatus: [✓] Receive, Error: None
Frame AppStatus: [✓] Receive, Error: None

Select Axis: Axis 1 | Axis 1_1 | Axis 1_1

periodeSendMin[ms]: 15
periodeSend[ms]: 30 Frames are synchronized with periodeSend

positionReq: 0
frequencyReq[Hz]: 770
ClrFlag: ☐
AppInitFlag: ☐
PosResetFlag: ☐
LightOnFlag: ✓

positionAct: 0
frequencyAct [Hz]: 0
AppFlags: StepRun, StopTimeout, [✓ PositPark], PositResetDone, [✓ AppInitDone], [✓ LightOn], StallDetected, RotDirSign

analogValue: 0x00
AppErrFlags: HighTemperature, HB_OverCur, HighVoltage, LowVoltage, PowerDieError, LINTxRxError, SICurrLim, StopSpeedError

positionReqManual: 0

Mode: Manual
Curve: SLOW-FAST
Send: Still

The period of the curve look-up table read-out is identical with the period of the signals sending periodSend

Control Loop 2 [✓] Enable

Frame PosCmd: [✓] Send, Error: None
Frame PosStatus: [✓] Receive, Error: None
Frame AppStatus: [✓] Receive, Error: None

Select Axis: Axis 2 | Axis 2 | Axis 2

periodeSendMin[ms]: 15
periodeSendReq[ms]: 30
periodeSend[ms]: 30

positionReq: 0
frequencyReq [Hz]: 200
ClrFlag: ☐
AppInitFlag: ☐
PosResetFlag: ☐
LightOnFlag: ☐

positionAct: 0
frequencyAct [Hz]: 0
AppFlags: StepRun, StopTimeout, [✓ PositPark], PositResetDone, [✓ AppInitDone], LightOn, StallDetected, [✓ RotDirSign]

analogValue: 0x00
AppErrFlags: HighTemperature, HB_OverCur, HighVoltage, LowVoltage, PowerDieError, LINTxRxError, SICurrLim, StopSpeedError

positionReqManual: 0

Status: RUN **Error:** NONE

Figure 7-2. LIN-bus Control Page

7.4.2 Control Loop2

The setting is the same as for Control Loop1 except:

- If there are communication requests of both loops at the same time, Loop1 always has the main priority. This means that a request from Loop2 is shifted until the request of Loop1 is satisfied. That is why there is a delay between two following satisfied communication requests from Loop2, measured and displayed via *periodeSend*. The *periodeSendReq* box is used to set the desired time periods between two communication events of Loop2.

FramePosCmd 16-bit data source (*positionReq*) is set only manually, by means of the *positionReqManual* box.

7.4.3 Status Notes and State Buttons

On the yellow background can be seen two Status notes and three State buttons:

- Status - displays current status of LIN communication and can be:
  - Idle - LIN is activated, but communication is not started
  - Run - communication is running
  - Sleep - LIN-bus is in Sleep state
- Error - global error label, which accumulates all error warnings.
- Sleep/Wake up - sends sleep or wake-up frame.
- Run/Stop - runs or stops communication via LIN-bus. Park Position - after button click is *positionReq* variable of both loops forced to zero and the mode of Loop1 is set to Manual.

7.4.4 Error Handling

The Error notes can be:

- None - communication without errors
- No Response - Slave is not responding (wrong selected Axis or Slave is missing). In case, when is Slave responding, this note reflects checksum error.
- Transmitter Issue - more than one LIN device is transmitting or LIN SIO wire is shorted to the supply source wires.

7.4.5 Low Time Cases

If the time dedicated to Loop1 communication requests is always shorter than the time necessary for Loop2 communication, the priority rule of Loop1 is dismissed and loop requests are satisfied in order following each other. This state is signalled by the note *Frames are not* *synchronized with periodeSend*, that is situated beside Loop1 *periodeSend* variable.

7.5 Recorder

This page is shown in **Figure 7-3**. It is possible to reach it in two ways. Either click on the Display Graphical Recorder button (Slave Control page) or click on the legend Recorder in the Project tree.

The Recorder is working as a standalone oscilloscope running directly on the Master Board. Each cycle is started by the Run button on the Recorder page. Then data in predefined time distances are sampled and stored to Master Board RAM. After ten seconds from start, data are reloaded to the PC master tool and displayed via the graphical interface. The dvantage of this way of watching variables is the recording of fast changing events.

In this case, three variables of Loop1 are watched and displayed (because of finite size of RAM):

1. positonReq - desired position of HID lamp (LIN Stepper Controller) before *FramePosCmd* transmitting
2. positionReqSent - desired position of HID lamp (LIN Stepper Controller) immediately after *FramePosCmd* transmitting
3. positionAct - actual position of HID lamp (LIN Stepper Controller) after receiving *FramePosStatus*

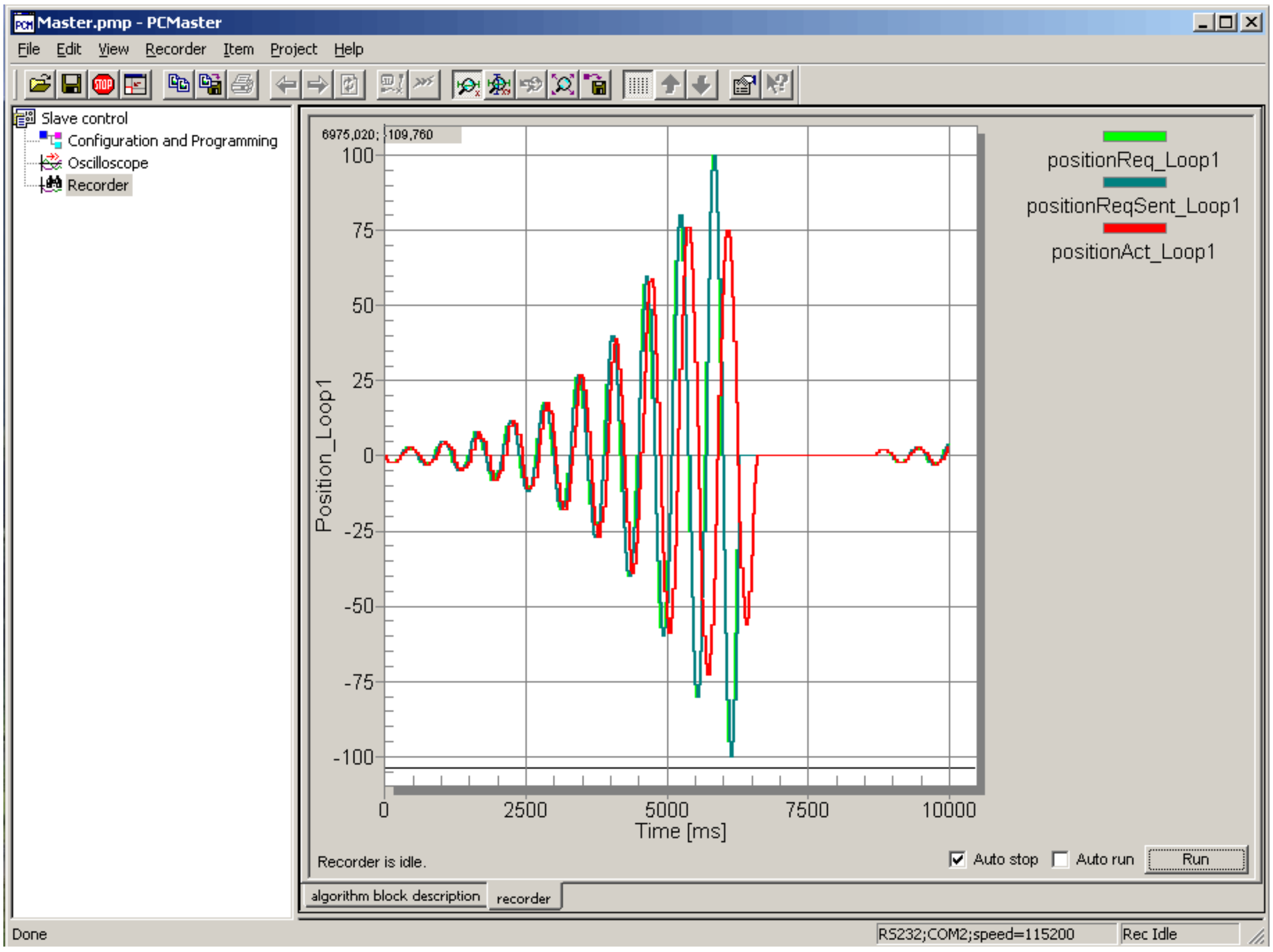


Figure 7-3. Recorder Page

By comparing 1) and 2) it is possible to watch the delay caused by transmitting *FramePosCmd* on LIN-bus; by comparing 1) and 3) it is possible to watch the delay caused by the mechanical parts of the LIN Stepper Controller that are driving the lamp position (LIN communication delays can be in the most of this cases neglected).

7.6 Oscilloscope

Figure 7-4 shows the Oscilloscope page.

The Oscilloscope works as a real time recorder and displays the current state of the variables. In comparison to the Recorder, the data for Oscilloscope are loaded to PC master software immediately and individually, whereas, in the case of the Recorder, this is done by group. This causes the Oscilloscope to be slower than the Recorder, so when the event is faster than the Oscilloscope data flow, data triggering that event are missing. The advantage of this tool is that it does not use the Master Board RAM as Recorder, so triggered events can be infinite.

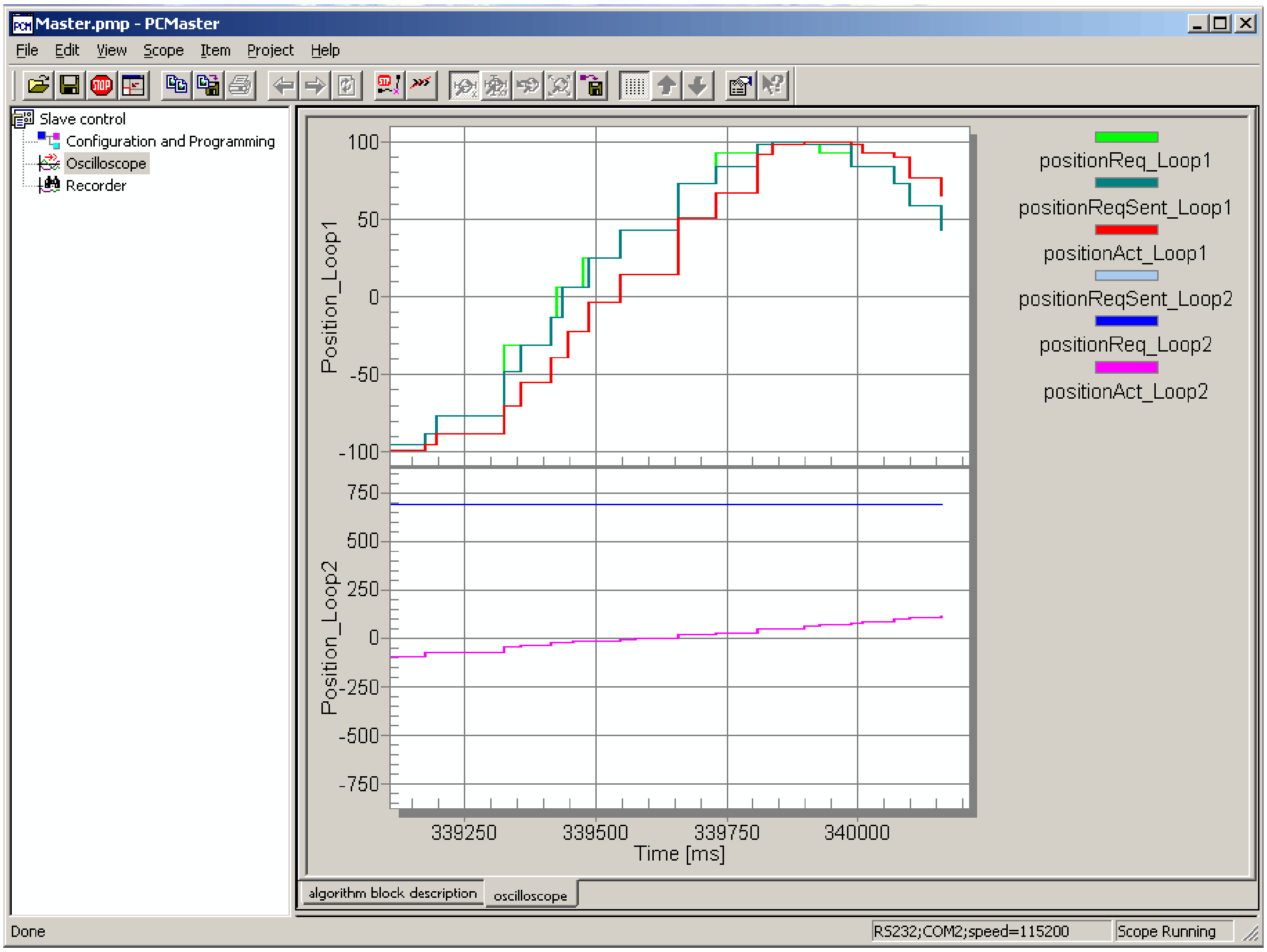


Figure 7-4. Oscilloscope Page

The same variables as in Recorder are displayed on the Oscilloscope page, but for both loops.

This section is finished by the comparison tables of variables between LIN-bus Control page and Variable Watch:

- Loop1 (**Table 7-1**)
- Loop2 (**Table 7-2**)
- Status notes and State buttons (**Table 7-3**)

Table 7-1. LIN-bus Control Page and Variable Watch Variables Comparison - Loop1

LIN-bus Control page variable name	Name of representative from Watch variable	Note
Send	enableTxPosition_Loop1	Enable transmit frame PosCmd
Receive	enableRxPosStatus_Loop1	Enable receive frame PosStatus
	enableRxStatus_Loop1	Enable receive frame AppStatus
Error	TxPositionError_Loop1	Error during frame PosCmd transmitting
	RxPosStatusError_Loop1	Error during frame PosStatus receiving
	RxStatusError_Loop1	Error during frame AppStatus receiving
Select Axis	axisTxPosition_Loop1	Target device of PosCmd frame
	axisRxPosStatus_Loop1	Target device of PosStatus frame
	axisRxStatus_Loop1	Target device of AppStatus frame
periodeSendMin	periodeSendMin_Loop1	Calculated period, range <0 - 255>, value in milliseconds
periodeSend	periodeSend_Loop1	Real period, range <0 - 255>, value in milliseconds
Mode	modeAutMan_Loop1	Select Manual or Automatic mode
Curve	autCurveSelect_Loop1	Select one predefined curve
Send	autSendOnesStill_Loop1	Send curve Ones or Still

Table 7-1. LIN-bus Control Page and Variable Watch Variables Comparison - Loop1

LIN-bus Control page variable name	Name of representative from Watch variable	Note
Reset	autReset_Loop1	Shift pointer in selected Curve to begin
positionReq	positionReq_Loop1	Included in frame PosCmd, data field, length 16 bits
frequencyReq	frequencyReq_Loop1	Included in frame PosCmd, data field, length 8 bits, page range <0Hz - 2500 Hz>
ClrFlag	ClrFlag_Loop1	Included in frame PosCmd, data field, length 1 bit each
AppInitFlag	AppInitFlag_Loop1	
PosResetFlag	PosResetFlag_Loop1	
LightOnFlag	LightOnFlag_Loop1	
positionAct	positionAct_Loop1	Included in frame PosStatus, data field, length 16 bits
frequencyAct	frequencyAct_Loop1	Included in frame PosStatus, data field, length 8 bits
AppFlags	uAppFlags_Loop1	Included in frame PosStatus, data field, length 8 bits
analogValue	analogValue_Loop1	Included in frame AppStatus, data field, length 8 bits
AppErrFlags	uAppErrFlags_Loop1	Included in frame AppStatus, data field, length 8 bits
positionReqManual	positionReqManual_Loop1	If is Mode manual, sets value of positionReq variable

Table 7-2. LIN-bus Control Page and Variable Watch Variables Comparison - Loop2

LIN-bus Control page variable name	Name of representative from Watch variable	Note
Send	enableTxPosition_Loop2	Enable transmit frame PosCmd
Receive	enableRxPosStatus_Loop2	Enable receive frame PosStatus
	enableRxStatus_Loop2	Enable receive frame AppStatus

Table 7-2. LIN-bus Control Page and Variable Watch Variables Comparison - Loop2

LIN-bus Control page variable name	Name of representative from Watch variable	Note
Error	TxPositionError_Loop2	Error during frame PosCmd transmitting
	RxPosStatusError_Loop2	Error during frame PosStatus receiving
	RxStatusError_Loop2	Error during frame AppStatus receiving
Select Axis	axisTxPosition_Loop2	Target device of PosCmd frame
	axisRxPosStatus_Loop2	Target device of PosStatus frame
	axisRxStatus_Loop2	Target device of AppStatus frame
periodeSendMin	periodeSendMin_Loop2	Calculated period, range <0 - 255>, value in milliseconds
periodeSendReq	periodeSendReq_Loop2	Desired period, range <0 - 255>, value in milliseconds
periodeSend	periodeSend_Loop2	Real period, range <0 - 255>, value in milliseconds
positionReq	positionReq_Loop2	Included in frame PosCmd, data field, length 16 bits
frequencyReq	frequencyReq_Loop2	Included in frame PosCmd, data field, length 8 bits, page range <0Hz - 2500 Hz>
ClrFlag	ClrFlag_Loop2	Included in frame PosCmd, data field, length 1 bit each
AppInitFlag	AppInitFlag_Loop2	
PosResetFlag	PosResetFlag_Loop2	
LightOnFlag	LightOnFlag_Loop2	
positionAct	positionAct_Loop2	Included in frame PosStatus, data field, length 16 bits
frequencyAct	frequencyAct_Loop2	Included in frame PosStatus, data field, length 8 bits
AppFlags	uAppFlags_Loop2	Included in frame PosStatus, data field, length 8 bits

Table 7-2. LIN-bus Control Page and Variable Watch Variables Comparison - Loop2

LIN-bus Control page variable name	Name of representative from Watch variable	Note
analogValue	analogValue_Loop2	Included in frame AppStatus, data field, length 8 bits
AppErrFlags	uAppErrFlags_Loop2	Included in frame AppStatus, data field, length 8 bits
positionReqManual	positionReqManual_Loop2	Value for positionReq variable

Table 7-3. LIN-bus Control Page and Variable Watch Variables Comparison - Status Notes and State Buttons

LIN-bus Control page variable name	Name of representative from Watch variable	Note
Status	LIN_Status	Display LIN status: IDLE/RUN/SLEEP
Error	Created as OR function of all error messages in HTML page code	
Sleep/Wake-up	LIN_SleepWakeReq	Change current LIN Status Sleep/Wake-up
Run/Stop	LIN_RunStopReq	Change current LIN Status Run/Stop
Park Position	Park	Sets lamp system to initial position
Note: *Frames are (not) synchronized with periodeSend*	SynchMode_Loop1	Display relationship between real and desired communication timing

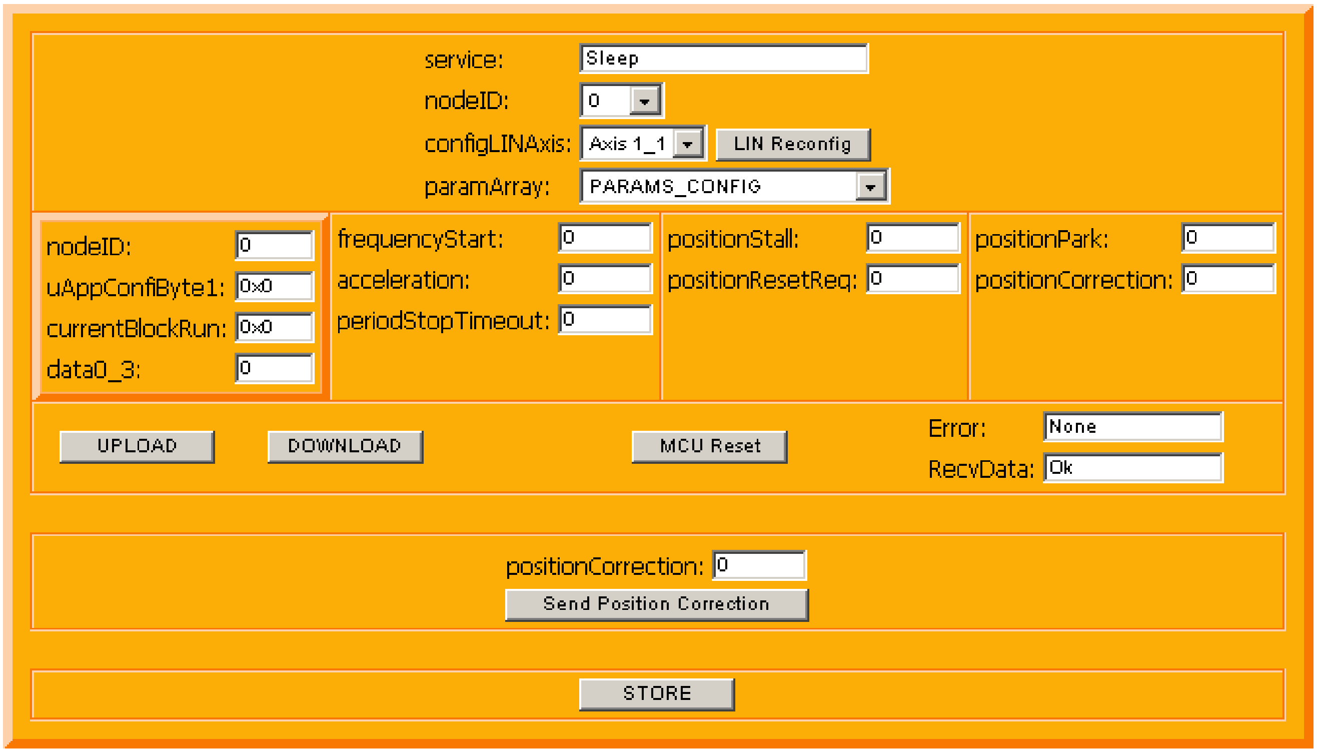
7.7 Programming and Configuration

By the means of this page (see **Figure 7-5**) it is possible to program and configure the LIN Stepper Controller via LIN-bus. The services are following:

- LIN Reconfiguration - assign Axis number
- Upload Parameters - read parameters
- Download Parameters - write parameters
- Store Parameters - store parameters to program MCU memory

- MCU Reset - reset MCU
- Send Position Correction - set new position

All variables included in the parameters array are described in **Section 5. LIN Master Software Description**.



service:	Sleep
nodeID:	0
configLINAxis:	Axis 1_1
paramArray:	PARAMS_CONFIG

nodeID:	0	frequencyStart:	0	positionStall:	0	positionPark:	0
uAppConfiByte1:	0x0	acceleration:	0	positionResetReq:	0	positionCorrection:	0
currentBlockRun:	0x0	periodStopTimeout:	0				
data0_3:	0						

Error:	None
RecvData:	Ok
positionCorrection:	0

Figure 7-5. Programming and Configuration Page

7.7.1 LIN Reconfiguration

If is necessary to change or program new Axis number of LIN Stepper Controller, select the desired Axis number in *configLINAxis* combo box and click on *LIN Reconfig* button. The target device will now have the chosen Axis number.

7.7.2 Upload Parameters

Steps are as follows:

1. Select nodeID (node identity) - for an uninitialized device (by the nodeID item in parameters array) the nodeID is 255
2. In the paramArray combo box, set which parameters are to be uploaded (a dedicated box will appear just like the box selected by paramArray PARAMS_CONFIG vote on **Figure 7-5** - in the middle of left side).
3. Click on the UPLOAD button
4. Parameters are now reloaded from the selected Node, and in service box is displayed the name of action that was provided by Node (in this case it must be UPLOAD, otherwise the Node reaction was wrong).

7.7.3 Download Parameters

For this choice:

1. Select nodeID (node identification) - for an uninitialized device (by the nodeID item in parameters array) the nodeIDis 255
2. In the *paramArray* combo box, set which parameters are to be downloaded (a dedicated box will appear just like box selected by *paramArray* PARAMS_CONFIG vote on **Figure 7-5** - in the middle of left side). Then change those parameters.
3. After clicking on the DOWNLOAD button, the parameters are written and immediately read back for verifying. If the parameter values are the same as were determined, in the *RecvData* box will be “Ok”. In the opposite case, “Different” will be displayed. In the *service* box must be “DOWNLOAD”, otherwise the Node reaction was wrong.

7.7.4 Store Parameters

Choose Node (*nodeID*) and click on the STORE button. All parameters are stored in the Node program memory.

For all Nodes, *nodeID* is zero; for uninitialized devices, it is 255.

7.7.5 MCU Reset

Select Node via *nodeID* combo box and click on MCU Reset button.

For all Nodes, *nodeID* is zero; for uninitialized devices, it is 255.

7.7.6 Send Position Correction

Set *nodeID* and variable *positionCorrection*. Then click on the Send Position Correction button.

For uninitialized device, nodeID is 255.

7.7.7 Error Handling

Error (corresponding variable in Variable Watch - *configProgramError*) box notes can be:

- None - communication without errors
- No Response - Slave is not responding (wrong selected Axis or Slave is missing). In case, when Slave is responding, this note reflects checksum error.
- Transmitter Issue - more than one LIN device is transmitting or LIN SIO wire is shorted to the supply source wires.

Section 8. Conclusion

One of the aims of this reference design was to show that the LIN-bus is suitable for HID headlamp levelling control and its communication speed is fully sufficient for this application.

The dynamic behavior of the HID lamp levelling system has some limitations due to mechanical parts. The bus communication system (LIN-bus) should not be the limitation for the system dynamic.

This is demonstrated below using the PC master recorder (see **Section 7.5. Recorder**). The figures below are measured using the PC master software. Control Loop1 controls automatically the levelling of a standard HID lamp according to signals pre-programmed in LIN Master. The horizontal levelling is set for Axis1.

All Axis1 frames are being sent. Control Loop2 is also enabled and controls Axis2. All Axis2 frames are being sent (see **Section 7.4. Slave Control**). All frames are sent with constant period **periodeSend** = 30ms.

According with the stepper motor and mechanical parts of the HID lamp, the following setting is provided:

- start speed = 200rpm
- max. speed **frequencyReq** = 700rpm
- and position range **positionReq** +/-128steps

Figure 8-1 shows that a slow sinusoidal signal of required position **positionReq** can be followed by the **actualPosition** signal. If the signal frequency increases, the HID lamp mechanics are not able to copy the required position. The LIN-bus is able to provide enough samples.

The system mechanics are the limiting factor.

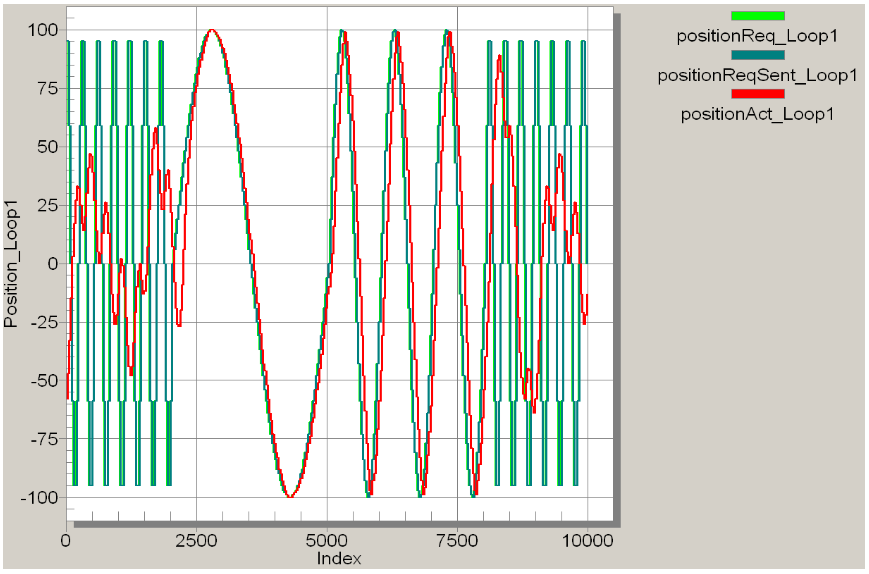


Figure 8-1. Slow-Fast Signal

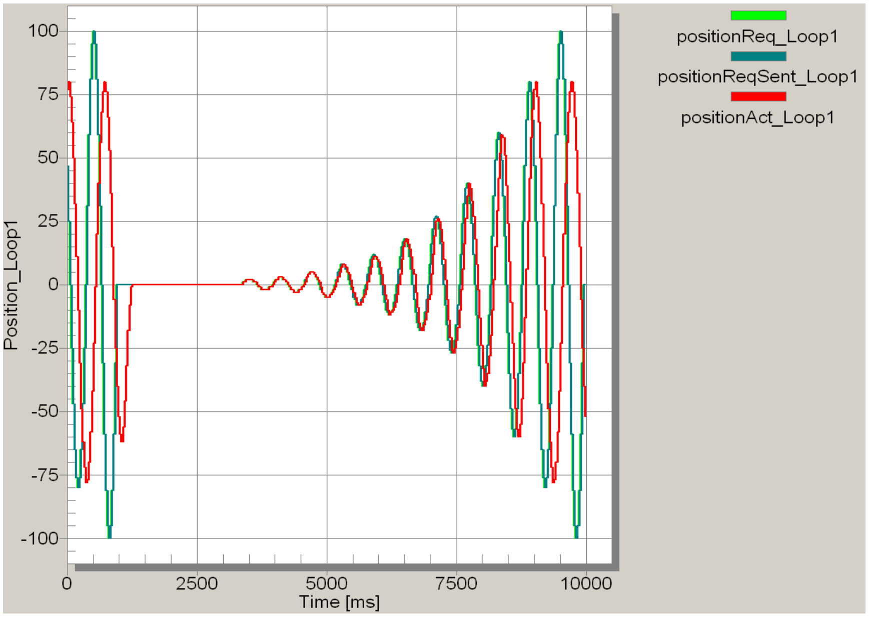


Figure 8-2. Low-High (Amplitude) Signal

The Figure 8-2 shows that the required position **positionReq** with a sinusoidal signal of small amplitude can be followed by **actualPosition** signal. If the signal amplitude increases, the stepper motor with its maximum speed is not able to follow the required position.

The LIN-bus is able to provide enough samples.

The system mechanics are the limiting factor.

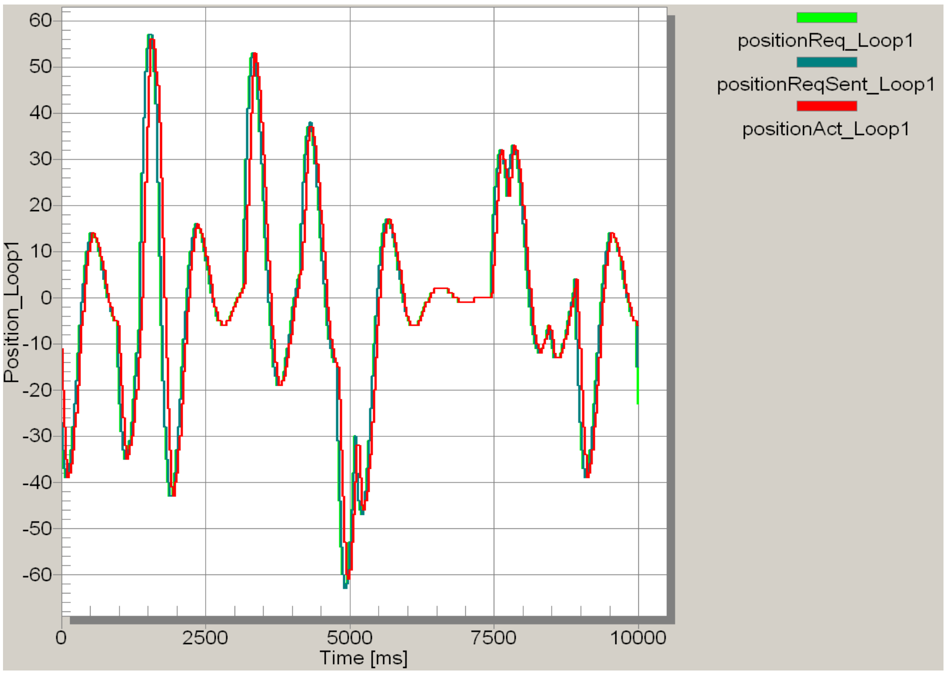


Figure 8-3. Road1 Signal

Figure 8-3 shows a non-sinusoidal signal of required position **positionReq** be followed by the **actualPosition** signal. It can simulate a HID levelling system on a road.

Therefore, we can say that the LIN Leveller described in this reference design could be an advanced solution for HID headlamp levelling control with a very competitive system cost thanks to:

- LIN-bus communication protocol

which is a cost-effective bus system based on standard SCI (UART) communication and:

- 908E625 device

as an integrated solution, which makes the slave nodes easy with a low number of components.

Section 9. References

1. PC Master Software User Manual
2. LIN Specification Package, Revision 1.2
3. System Integration in Automotive Lighting - Improvements in Visibility at Night, Rainer Neumann, Visteon Deutschland GmbH, SAE 2002-01-1989
4. Bending Light, Kevin Jost, SAE 1-110-12-26
5. Adaptive front lightning, Stuart Birch, SAE 1-109-12-39
6. Bifunction HID Headlamp Systems - Reflection and Projection Type, Doris Boebel, Heike Eichler and Verena Hebler, Automotive Lighting GmbH, Automotive Lighting Research (SP–1531)
7. HID System: Function Integration, Christophe Cros, Valeo Lighting System
8. Innovations in Lighting with Adaptive Headlamp Technology, Michael Hamm and Ernst-Olaf Rosenhahn, Automotive Lighting Reutlingen, SAE 2001-01-3392
9. LIN General Purpose IC 908E625ACDWB/D
10. MC68HC908EY16 Data Sheet MC68HC908EY16/D
11. LIN Physical Interface MC33399
12. 16-bit MCU MC9S12DP256B

Appendix A. Hardware Schematics

A.1 LIN Master Board Schematic

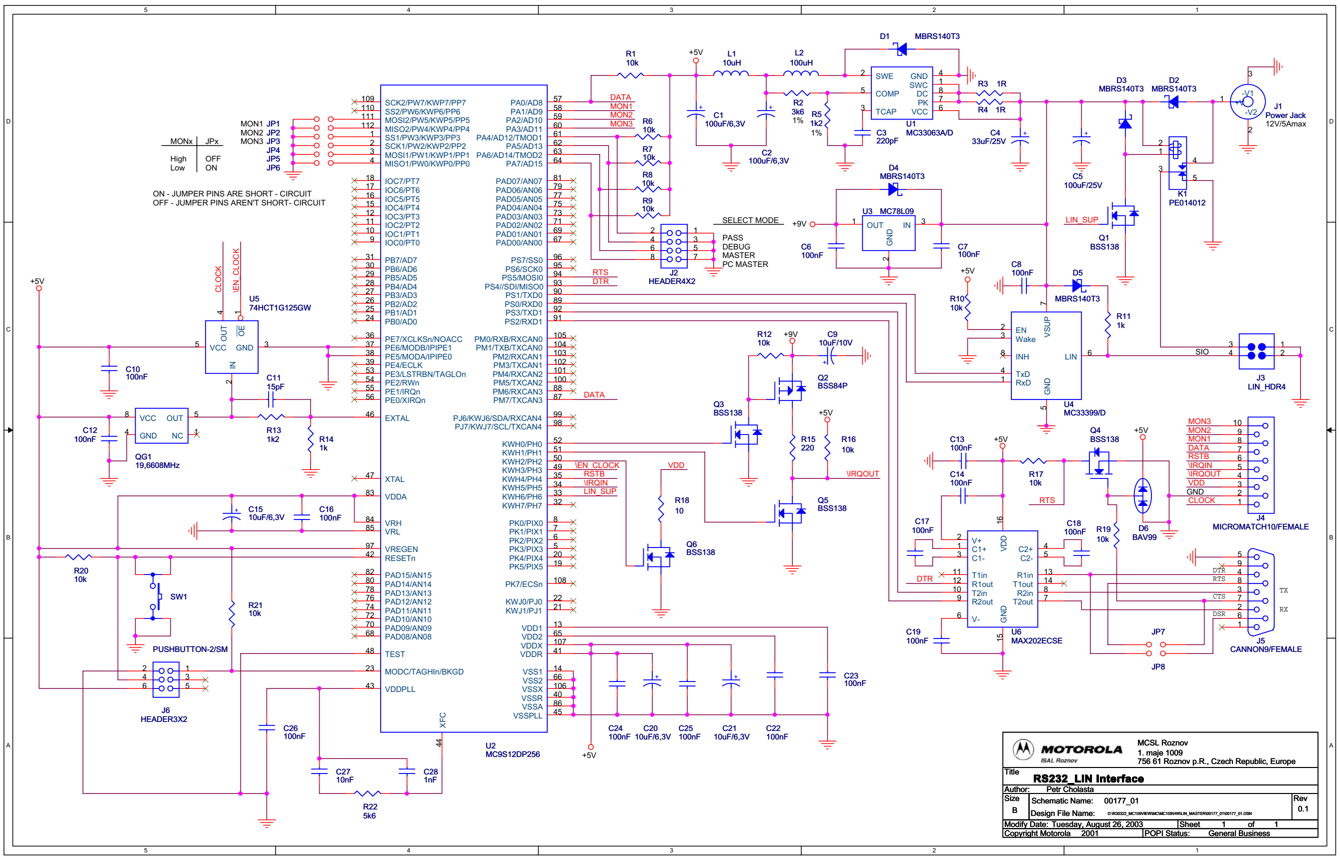


Figure A-1. LIN Master Board Schematic

A.2 LIN Stepper Board Schematic

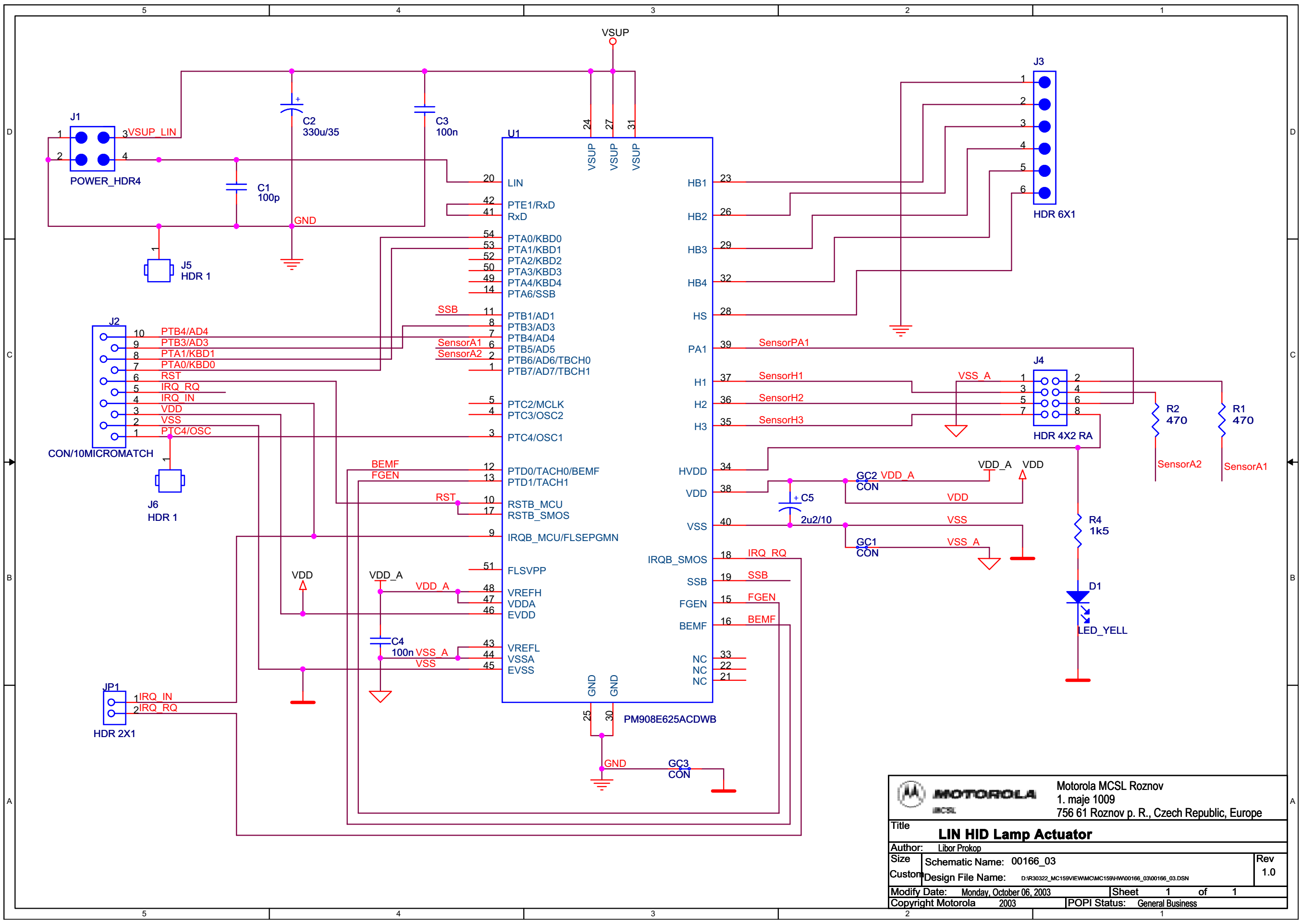


Figure A-2. LIN Enhanced Stepper Board Schematic

Appendix B. 908E625 Advantages and Features

This general purpose IC from Motorola has been developed as a highly integrated and cost-effective solution for load driving within intelligent LIN distributed architectures. It is especially suited to the control of automotive mirror, door-lock, and light-levelling applications.

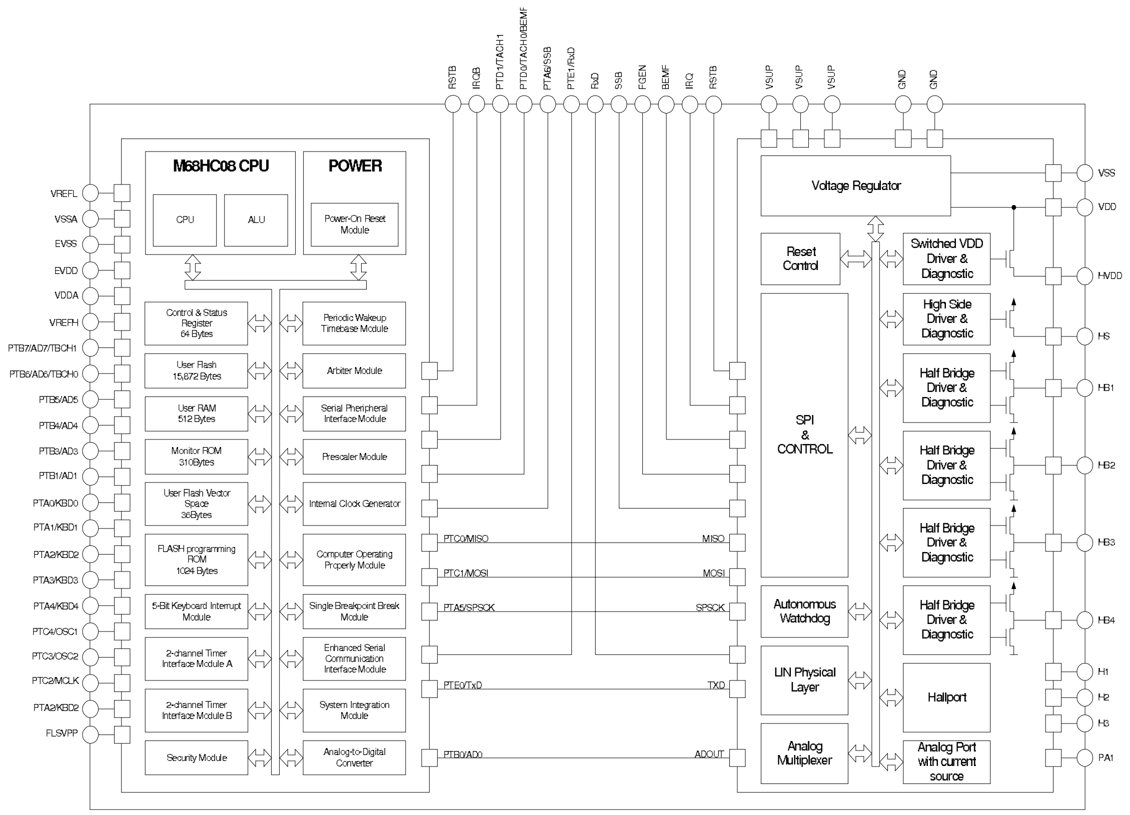

M68HC08 CPU
CPU
ALU
POWER
Power-On Reset Module
Control & Status Register 64 Bytes
User Flash 15,872 Bytes
User RAM 512 Bytes
Monitor ROM 310Bytes
User Flash Vector Space 36Bytes
FLASH programming ROM 1024 Bytes
5-Bit Keyboard Interrupt Module
2-channel Timer Interface Module A
2-channel Timer Interface Module B
Security Module
Periodic Wakeup Timebase Module
Arbiter Module
Serial Pheripheral Interface Module
Prescaler Module
Internal Clock Generator
Computer Operating Properly Module
Single Breakpoint Break Module
Enhanced Serial Communication Interface Module
System Integration Module
Analog-to-Digital Converter
VREFL
VSSA
EVSS
EVDD
VDDA
VREFH
PTB7/AD7/TBCH1
PTB6/AD6/TBCH0
PTB5/AD5
PTB4/AD4
PTB3/AD3
PTB1/AD1
PTA0/KBD0
PTA1/KBD1
PTA2/KBD2
PTA3/KBD3
PTA4/KBD4
PTC4/OSC1
PTC3/OSC2
PTC2/MCLK
PTA2/KBD2
FLSVPP
RSTB
IRQB
PTD1/TACH1
PTD0/TACH0/BEMF
PTA6/SSB
PTE1/RxD
RxD
SSB
FGEN
BEMF
IRQ
RSTB
VSUP
VSUP
VSUP
GND
GND
PTC0/MISO
MISO
PTC1/MOSI
MOSI
PTA5/SPSCK
SPSCK
PTE0/TxD
TXD
PTB0/AD0
ADOUT
Voltage Regulator
Reset Control
SPI & CONTROL
Autonomous Watchdog
LIN Physical Layer
Analog Multiplexer
Switched VDD Driver & Diagnostic
High Side Driver & Diagnostic
Half Bridge Driver & Diagnostic
Half Bridge Driver & Diagnostic
Half Bridge Driver & Diagnostic
Half Bridge Driver & Diagnostic
Hallport
Analog Port with current source
VSS
VDD
HVDD
HS
HB1
HB2
HB3
HB4
H1
H2
H3
PA1

Figure B-1. 908E625 Simplified Block Diagram

This device is a multi-chip combination within a 54-lead Small Outline Integrated Circuit (SOIC) package. It consists of a standard HC08 MCU chip with SCI, SPI, internal oscillator, and FLASH memory, plus a Power

Die with four half-bridges and one high-side switch with diagnostic functions combined with Hall sensor and analog inputs, a LIN physical layer, and a voltage regulator.

908E625 Features:

- Multi-chip combination within a 54-lead SOIC package
- High-performance M68HC08 core
- 16K bytes of on-chip FLASH memory
- 512 bytes of RAM
- Internal clock generation module
- 16-bit, 2-channel timet
- 10-bit analog-to-digital converter (ADC)
- LIN physical layer interface
- Three 2-pin Hall sensor inputs
- One analog input with switchable current source
- Four low-resistive half-bridge outputs with current limitation
- One low-resistive high-side output
- 14 microcontroller I/Os

Appendix C. LIN Frames and Signals

This section describes LIN messaging with the frames, signals and the LIN Stepper Controller functionality.

C.1 LIN Leveller Basic Frames

Figure C-1 describes the LIN Leveller signals and frames.

NOTE: *The LIN messaging scheme, with exact signal description and updated according to latest software modifications, is described in the file Messaging_LIN_Levellew.xls which is provided with the application software files.*

The signal provider is the node that sends the response fields (**Section 9. References**, **2**) of the described frame. The signal acceptor is the node that is programmed to act on the received frame (see **Section 4.1. Axis and Signal Providers and Acceptors**).

The column **Signal Functionality Description** describes the LIN Stepper Controller functionality according to the described signal. The raw value range is the range of the signal in system units. Although the LIN API specifies the unsigned signals, some signals (e.g. position) have signed representation. The normalized value is a physical representation of the signal (variable).

NOTE: *The normalized value range is determined by the scaling factor. The scaling constants RESOLUTION_FREQUENCY_HZ, RESOLUTION_PERIOD_NS are defined and can be changed in the LIN Stepper software header files.*

See Sections **6.3.1**, **6.3.2**, **6.3.3**, and **6.3.4** for information about scaling.

Table C-1. LIN Leveller Messaging

Signal Provi der	Signal Accep tor(s)/ Axis	Frame Name (ID)	Signal Name	Signal Functionality Description	Raw Value Range	Normalized Value Range
Master	A1_1 A1_2	frmPosCmdA1 (0x20)	l_u16_rd_positionReqA1	Required absolute Position. The LIN Stepper Controller acceptor provides automatic control of the actual position to this value	<0xC000, 0x3FFF>	<-16384, 16383>
			l_u8_rd_frequencyReqA1	Required Frequency. The LIN Stepper Controller acceptor provides automatic position control with motor actual stepping frequency trying to ramp up (or down) to the Required Frequency.	x = <0x00, 0xFF>	f(x) = RESOLUTION_FREQUENCY_HZ *x [Hz]
			l_bool_rd_flagClrA1	Clear Error Flag. The LIN Stepper Controller acceptor provides all system error clear	0 False 1 True	
			l_bool_rd_ApplInitFlagA1	Application Initialization Flag. The LIN Stepper Controller acceptor puts all processes to the application initialization state. The actual position is not initialized in this state. For details see **Section 6.1.2. Position and Speed Control** *Caution: The application init. state is unconditionally entered even if the motor is not stopped. This can cause the actual position to be lost.*	0 False 1 True	
			l_bool_rd_PosResetFlagA1	Position Reset Flag. The LIN Stepper Controller acceptor provides motor position reset. This position reset is described below: - software controls the motor stepping to the positionResetRqValue. - mechanical stall keeps the motor at defined position (even the electrical stepping is in progress) - after actual position counter reaches the positionResetRqValue, the software sets the counter to positionStall For details see **Section 6.1.2. Position and Speed Control** Note: There can be other requirements to control the motor to a reset position, e.g. using stall detection or Hall sensor. These were not used in current software but can be easily implemented on LIN Stepper Controller	0 1	0-False 1-True

Table C-1. LIN Leveller Messaging (Continued)

Signal Provider	Signal Acceptor(s)/ Axis	Frame Name (ID)	Signal Name	Signal Functionality Description	Raw Value Range	Normalized Value Range
Master	A1_1 A1_2	frmPosCmdA1 (0x20)	l_bool_rd_LightOnFlagA1	Light On Flag. The LIN Stepper Controller acceptor controls the High Side switch of the power output. So the connector J3 pin 6 can be used for the light on/off control	0 1	0-HS OFF 1-HS ON
Master	A2	frmPosCmdA2 (0x23)	same as frmPosCmdA1 but the acceptor is A2			
Master	A3	frmPosCmdA3 (0x25)	same as frmPosCmdA1 but the acceptor is A3			
A1_1	Master	frmPosStatusA1_1 (0x21)	l_u16_wr_positionActA1_1	Actual absolute Position. The LIN Stepper Controller provider sends the actual position of the position counter	<0xC000, 0x3FFF>	<-16384, 16383>
			l_u8_wr_frequencyActA1_1	Actual Frequency. The LIN Stepper Controller provider sends the actual motor stepping frequency	x = <0x00, 0xFF>	f(x) = RESOLUTION_FREQUENCY_HZ * x [Hz]
			l_u8_wr_uAppFlags1A1_1	Application status Flags #1 The LIN Stepper Controller provider sends the status byte with the flags bit0 **StepRun** - motor is stepping = run bit1 **StopTimeout** - stop timeout (after stepping before setting motor block) bit2 **PositPark** - motor is actually stopped with actual position = positionPark bi3 **PositResetDone** - position reset was provided after last MCU reset bit4 **ApplnitDone** - application initializationflag 0 - initialization started 1 - initialization done bit5 **LightOn** - light (Hight side switch on the connector J3 pin 6) set to on bit6 **StallDetected** - NOT IMPLEMENTED stall detected bit7**RotDirSign** - motor rotation direction signature 1 - actual position decremented 0 - actual position incrementing	0 1	0 - False 1 - True
A1_2	Master	frmPosStatusA1_2 (0x22)	same as frmPosStatusA1_1 but the signal provider is A1_2			
A2	Master	frmPosStatusA2 (0x24)	same as frmPosStatusA1_1 but the signal provider is A2			
A3	Master	frmPosStatusA3 (0x26)	same as frmPosStatusA1_1 but the signal provider is A3			

Table C-1. LIN Leveller Messaging (Continued)

Signal Provider	Signal Acceptor(s)/ Axis	Frame Name (ID)	Signal Name	Signal Functionality Description	Raw Value Range	Normalized Value Range
A1_1	Master	frmAppStatusA1_1 (0x1C)	l_u8_wr_AppErrFlagsA1_1	Application Error Flags. The LIN Stepper Controller provider sends the error byte with the flags bit0 **HighTemperature** - Power Die over temperature bit1 **HB_OverCur** - H-bridge overcurrent flag bit2 **HighVoltage** - H-bridge high voltage bit3 **LowVoltage** - H-bridge low voltage bit4 **PowerDieError** - Power Die error bit5 **LINTxRxError** - **NOT IMPLEMENTED** bit6 **SICurrLim** - serial input current limitation bit7 **StopSpeedError** - speed was too enough when motor stopped (possible loss of the step - used for software debugging)	0 1	0 - False 1 - True
			l_u8_wr_analogValueA1_1	**NOT IMPLEMENTED** - any analog value like temperature, DC bus voltage, etc. can be send out from the LIN Stepper Controller provider		
A1_2	Master	frmAppStatusA1_2 (0x1D)	same as frmAppStatusA1_1 but the signal provider is A1_2			
A2	Master	frmAppStatusA2 (0x1E)	same as frmAppStatusA1_1 but the signal provider is A2			
A3	Master	frmAppStatusA3 (0x1F)	same as frmAppStatusA1_1 but the signal provider is A3			

C.2 Node ID

The controlled axis specifies the relation to the LIN basic messages; each LIN Stepper Controller node relation to LIN messaging is specified by two parameters

- the configured axis
- exclusive Node ID

The Node ID is used for configuration. The LIN Master Request and Slave Response Frames (Command frames) used for configuration are broadcast frames (each slave node acts upon them). We must distinguish the nodes when using **Appendix C.3. LIN Leveller Configuration Frames**.

NOTE: *The default node ID settings of the LIN Stepper software reflect the configured axis; so, Axis3 has node ID = 4, Axis2 has node ID = 3, Axis1_2 has node ID = 2, Axis1_1 has node ID = 1. However, both the configuration axis and the node ID can be independently changed in the software or during configuration. The user must guarantee that there will be no other nodes with the same node ID connected to one LIN-bus.*

C.3 LIN Leveller Configuration Frames

The Master Request and Slave Response frames were used for the LIN Stepper Controller configuration. The configuration allows, on LIN, adaptation of the LIN Stepper software. Each configuration frame is used to configure the LIN Stepper Controller with node ID equal to the l_u8_rd_nodeID signal (see **Appendix C.3. LIN Leveller Configuration Frames**).

The configuration process covers two functions:

1. Parameters Configuration

   Provides upload and download of the control parameters from and to **paramRAM** structure. The paramArray variable with a dedicated signal l_u8_rd_paramArray defines the section of the parameters sent in four data signals l_u8_rd_datax. It is also described in **Section 6.1.8. Config Param**. Before the Store service, the parameters updated in the configuration are stored in **paramRAM** volatile structure.

   The service l_u8_rd_service = Store (FLASH) provides storing of all RAM parameters arrays = paramRAM to paramROM. The paramROM is in the FLASH memory and is copied to the paramRAM after any MCU reset.

   The service l_u8_rd_service = MCU Reset forces the reset of dedicated LIN Stepper Controller

2. LIN Reconfiguration

   Changes the dedicated LIN Stepper Controller configuration. It sets its LIN driver to select the frames and signals according to the defined axis. The axis are described in **Section 4.1. Axis and Signal Providers and Acceptors**. The LIN driver filters out the messages dedicated for other controlled axis. The LIN reconfiguration is also described in the **Section 6.1.9. Reconfig LIN**.

Table C-2. LIN Leveller Configuration Frames

Signal Provider	Signal Acceptor(s)/ Axis	Frame Name (ID)	Signal Name	Signal Functionality Description	Raw Value Range	Normalized Value Range
Master	broadcast	frmMaster Request (0x3C)	l_u8_rd_service	Service Byte. Specifies the Master Request Frame service. The application uses this Command Frame user defined service as specified in **Section 9. References**, 2. The service byte determines the meaning of the next signals in the frmMasterRequest frame	0x00 <0x01, 0x7F> 0x80 0x81 0x82 0x83 0x84	0x00 - Sleep <0x01 - 0x7F> Reserved 0x80 Upload parameters (S->M) (prior frmSlaveResponse) 0x81 Download parameters (M->S) 0x82 Store (FLASH) 0x83 MCU Reset 0x84 LIN Reconfig
			l_u8_rd_nodeID	Node ID. Configuration of Each LIN Stepper Controller node is determined by the node ID (see **Appendix C.3. LIN Leveller Configuration Frames**). *With the exception of the Sleep Service the node reacts only if the l_u8_rd_nodeID is equal with the defined nodeID parameter.*	<0, 0xFF>	<0,255>
			l_u8_rd_configLINAxis	config. Axis. This signal is used only for LIN Reconfig. If any node Node ID parameter = l_u8_rd_nodeID and l_u8_rd_service = LIN Reconfig the node is reconfigured to the defined axis (see **Section 6.1.9. Reconfig LIN** and **Section 4.1. Axis and Signal Providers and Acceptors**).	0x00 0x01 0x02 0x03 <0x04, 0xFF>	0x00 - A1_1 0x01 - A1_2 0x02 - A2 0x03 - A3 <0x04, 0xFF> - Reserved
			l_u8_rd_paramArray	If any node Node ID parameter = l_u8_rd_nodeID and l_u8_rd_service = Download respectively and l_u8_rd_service = upload the following data signals l_u8_rd_datax addresses the space defined by l_u8_rd_paramArray as shown in **Figure C-1**	0x00 0x01 0x02 0x03	0-PARAMS_CONFIG 1-PARAM_SPEED 2-PARAM_RESET_POS 3PARAM_PARK_POSITION

Table C-2. LIN Leveller Configuration Frames (Continued)

Signal Provider	Signal Acceptor(s)/ Axis	Frame Name (ID)	Signal Name	Signal Functionality Description	Raw Value Range	Normalized Value Range
			in case paramArray = 0 [PARAMS_CONFIG] the data signals has the below described meaning			
Master	broad cast	frmMaster Request (0x3C)	l_u8_rd_data0	the signal is interpreted according to the l_u8_rd_paramArray signal (paramArray = 0) **node ID**. This signal changes current Stepper Controller node ID to this new node ID. *Caution: Beginning the next configuration frames. the configured node will react only when the signal l_u8_rd_nodeID = this new node ID*	<0,0xFF>	<0,255>
			l_u8_rd_data1	the signal is interpreted according to the l_u8_rd_paramArray signal (paramArray = 0) **uAppConfiByte1** bit3 - **RotDir24AtPosit** - motor rotation direction at motor positive direction signature (flag **RotDirSign**) is phase HB2 -> HB4 bit6 - **AccelEnbl** - motor speed Acceleration Enabled bit7 - **FullStep** - full step stepper operation (1 - full stepping 0 - half stepping)	0 1	0 - False 1 - True
			l_u8_rd_data2	the signal is interpreted according to the l_u8_rd_paramArray signal (paramArray = 0) **currentBlockRun** defines the current limitation when motor is running or stopped	0x1Y 0xZ2 0x3Y 0x4Y 0x5Y 0x6Y 0x7Y 0xZ0 0xZ1 0xZ2 0xZ3 0xZ4 0xZ5 0xZ6 0xZ7	Block current Limitation: 0x0Y off (no current) 0x1Y no limitation 0xZ2 no limitation 0x3Y 60mA 0x4Y 250mA 0x5Y 350mA 0x6Y 500mA 0x7Y 700mA Run current Limitation: 0xZ0 no limitation 0xZ1 no limitation 0xZ2 no limitation 0xZ3 60mA 0xZ4 250mA 0xZ5 350mA 0xZ6 500mA 0xZ7 700mA
			l_u8_rd_data3	the signal is interpreted according to the l_u8_rd_paramArray signal (paramArray = 0) **data0_3** - RESERVED		

Table C-2. LIN Leveller Configuration Frames (Continued)

Signal Provider	Signal Acceptor(s)/ Axis	Frame Name (ID)	Signal Name	Signal Functionality Description	Raw Value Range	Normalized Value Range
Master	broad cast	frmMaster Request (0x3C)	in case paramArray = 1 [PARAM_SPEED] the data signals has the below described meaning			
			l_u8_rd_data0	the signal is interpreted according to the l_u8_rd_paramArray signal (paramArray = 1) **frequencyStart** is the minimum frequency of the motor. The motor starts stepping and ramps down to this frequency before stop	x = <0,255 >	f(x) = FREQUENCY_ RESOLUTION_HZ * x [Hz]
			l_u8_rd_data1	(paramArray = 1) **acceleration** is motor speed acceleration and deceleration ramp constant	x = <0,255 >	f(x) = RESOLUTION_ACCEL_ DECEL_S_S2* x [steps/s^2]
			l_u8_rd_data2	(paramArray = 1) **periodStopTimeoutL** time instant before the motor block state after the motor stops Low Byte	x.L = <0,255 >	f(x) = RESOLUTION_PERIOD _NS* x.HL [ns]
			l_u8_rd_data3	(paramArray = 1) **periodStopTimeoutH** time instant before the motor block state after the motor stops High Byte	x.H= <0,255 >	f(x) = RESOLUTION_PERIOD_ NS* x.HL [ns]
Master	broad cast	frmMaster Request (0x3C)	in case paramArray = 2 [PARAM_RESET_POS] the data signals has the below described meaning			
			l_u8_rd_data0	(paramArray = 2) **positionStallL** position of the low stall used for position reset (see **Table C-1**. **l_bool_rd_PosResetFlagA1**) High Byte	x.L= <0x0, 0xFF>	x.HL = <-16384, 16383>
			l_u8_rd_data1	(paramArray = 2) **positionStallH** position of the low stall used for position reset (see **Table C-1**. **l_bool_rd_PosResetFlagA1**) High Byte	x.H= <0xC0, 0x3F>	x.HL = <-16384, 16383>
			l_u8_rd_data2	(paramArray = 2) **positionResetRqValueL** position reset value is used for position reset (see **Table C-1**. **l_bool_rd_PosResetFlagA1**) Low Byte	x.L= <0x00, 0xFF>	x.HL = <-16384, 16383>

Table C-2. LIN Leveller Configuration Frames (Continued)

Signal Provider	Signal Acceptor(s)/ Axis	Frame Name (ID)	Signal Name	Signal Functionality Description	Raw Value Range	Normalized Value Range
Master	broad cast	frmMaster Request (0x3C)	l_u8_rd_data3	(paramArray = 2) **positionResetRqValueH** position reset value is used for position reset (see **Table C-1**. **l_bool_rd_PosResetFlagA1**) Low Byte	x.L= <0xC0, 0x3F>	x.HL = <-16384, 16383>
Master	broad cast	frmMaster Request (0x3C)	in case paramArray = 3 [PARAM_PARK_POSITION] the data signals has the below described meaning			
			l_u8_rd_data0	(paramArray = 3) **positionParkL** position park is used for actual position setting after MCU reset. Therefore the master should set the positionReq of each Stepper Controller to this dedicated positionPark before reset (and power down). The Stepper Controller indicates that positionActual = positionPark with **PositPark** flag (see **Table C-1**. **PositPark**) Low Byte	x.L= <0x0, 0xFF>	x.HL = <-16384, 16383>
			l_u8_rd_data1	(paramArray = 3) **positionParkH** position park is used for actual position setting after MCU reset. Therefore the master should set the positionReq of each Stepper Controller to this dedicated positionPark before reset (and power down). The Stepper Controller indicates that positionActual = positionPark with **PositPark** flag (see **Table C-1**. **PositPark**) High Byte	x.H= <0xC0, 0x3F>	x.HL = <-16384, 16383>
			l_u8_rd_data2	(paramArray = 3) **positionCorrectionL** the relative position Correction is used to correct the actual position by sending this signals Low Byte	x.L= <0x00, 0xFF>	x.HL = <-16384, 16383>
			l_u8_rd_data3	(paramArray = 3) **positionCorrectionH** the relative position Correction is used to correct the actual position by sending this signals High Byte	x.H= <0xC0, 0x3F>	x.HL = <-16384, 16383>

Table C-2. LIN Leveller Configuration Frames (Continued)

Signal Provider	Signal Acceptor(s)/ Axis	Frame Name (ID)	Signal Name	Signal Functionality Description	Raw Value Range	Normalized Value Range
slave with node ID = l_u8_rd_nodeID	Master	frmSlave Response (0x3D)	The frame frmSlaveResponse provides the LIN Stepper Controller previously addressed (initiated) with frmMasterRequest of l_u8_rd_service = 0x80 Upload l_u8_rd_nodeID = node ID parameter			
			l_u8_wr_service	**Service Byte**. Specifies the Master Request Frame service. The application uses this Command Frame user defined service as specified in **Section 9. References**, **2**. The service byte the meaning of the next signals in the frmMasterRequest frame	0x00, <0x01, 0x7F>, 0x80	0x00 - Sleep <0x01, 0x7F> - Reserved 0x80 Upload parameters (S->M)
			l_u8_wr_nodeID	**Node ID**. Current LIN Stepper Controller node ID (see **l_u8_rd_nodeID** and **Appendix C.3. LIN Leveller Configuration Frames**).	<0, 0xFF>	<0,255>
			l_u8_wr_configLINAxis	**config Axis**. Current LIN Stepper Controller signal axis (see **l_u8_rd_configLINAxis**, **Section 4.1. Axis and Signal Providers and Acceptors**, **Section 6.1.9. Reconfig LIN**).	0x00 0x01 0x02 0x03 <0x04, 0xFF>	0x00 - A1_1 0x01 - A1_2 0x02 - A2 0x03 - A3 <0x04, 0xFF> - Reserved
			l_u8_wr_paramArray	**l_u8_wr_paramArray = l_u8_rd_paramArray** of the initiated frmMasterRequest the following data signals l_u8_wr_dataX addresses the space defined by that l_u8_rd_paramArray according the **Figure C-1**.	0x00 0x01 0x02 0x03	0-PARAMS_CONFIG 1-PARAM_SPEED 2-PARAM_RESET_POS 3PARAM_PARK_POSITION
			l_u8_wr_dataX	same as **frmMasterRequest** but l_u8_rd_dataX data upload see **l_u8_rd_data0**...		see **frmMasterRequest**, **l_u8_rd_data0**...

NOTE: *the normalized value range is determined by scaling factor. The scaling constants RESOLUTION_FREQUENCY_HZ and RESOLUTION_PERIOD_NS are defined and can be changed in the LIN Stepper software header files.*

The parameters configuration uses addressing of the parameters according to **Figure C-1**. There are four data signals in the Master

Request and Slave response frames. The parameters space of these data signals is addressed the parameters RAM according to paramArray pointer.

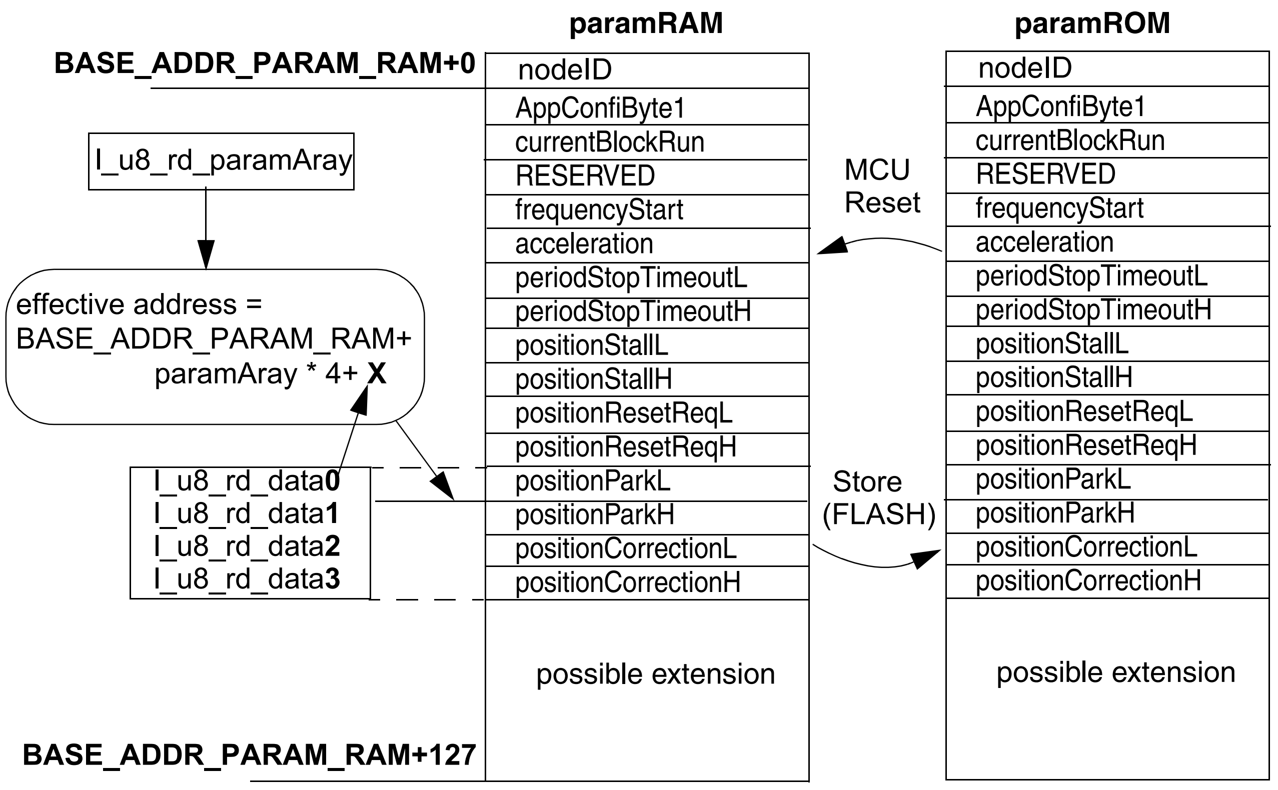


Figure C-1. Configuration Parameters Addressing

C.4 Possible Software Extension Programming via LIN

The principle of the parameters configuration could be possibly enhanced via LIN-bus software programming. The software FLASH memory could be split into two segments.

- resident software segment with LIN driver
- re-programmable software segment

The resident constant software segment cannot be reprogrammed. Since the LIN-bus communication and some control features are necessary for the software download.

The re-programmable software segment could be reprogrammed using the same principle as the parameters configuration. So there is a RAM space area up to 128 or even 256 bytes. This RAM area could be loaded step by step using a configuration frame (similar to the one used for parameters configuration). Each configuration frame would load four data bytes in the RAM area. The FLASH memory with the code could be split into areas (128 or 256). Each area could be step-by-step flashed after the RAM space is fully loaded. This way any size of re-programmable software segment in FLASH memory can be programmed.

Appendix D. LIN Stepper Software Data Variables

Table D-1 describes the LIN Stepper Controller data variables.

Table D-1. Stepper Controller Software Data Variables

Name	Components	Description
eAppState	enumeration constants: APP_INIT, APP_RUN, APP_POS_INIT, APP_PREPARE_CONFIG APP_CONFIG, APP_PREPARE_SLEEP, APP_SLEEP,	Application State enumeration
frequencyActLowHigh		Actual motor stepping Frequency union
	Word	
	Byte.High	
	Byte.Low	
frequencyReq		Required motor stepping Frequency
HBCTL, eHBCTL		H-Bridge Control Register in Power Die (see **Section 9. References**, 9)
HBOUT, eHBOUT		H-Bridge Output Register in Power Die (see **Section 9. References**, 9)
IFR, erIFR		Interrupt Flag Register in Power Die (see **Section 9. References**, 9)
IMR, erIMR		Interrupt Mask Register in Power Die (see **Section 9. References**, 9)
periodStep		motor stepping Period
positionAct		Actual motor Position
positionreq		Required motor Position
POUT, erPOUT		Power Output register in Power Die (see **Section 9. References**, 9)

Table D-1. Stepper Controller Software Data Variables (Continued)

Name	Components	Description
sParameterRAM		RAM structure with control parameters See **Table C-2**, **frmMasterRequest** for details on each component
	positionCorrection	Motor Position Correction
	positionPark	motor Parking/ position reset position for position reset
	positionResetRqValue	motor position Reset Request position for position reset
	positionStall	motor Stall position for position reset
	periodStopTimeou	Period Stop Timeout after motor deceleration
	acceleration	frequency acceleration constant
	frequencyStart	motor Start/Minimum stepping frequency
	data0_3	RESERVED
	curentBlockRun	current limitation for motor block/run state
	uAppConfigByte1	Application Configuration Byte #1
	nodeID	
uAppConfigByte1	RotDir24AtPosit	Flag motor rotation direction at positive speed HB2->HB4
	AccelEnbl	Flag motor speed Acceleration Enabled
	FullStep	Flag full step stepper operation 1 - full stepping 0 - half s
sParameterROM	same as sParameterRAM	
SYSCTL, erSYSCTL		System Control Register in Power Die (see **Section 9. References**, 9)
SYSSTAT, erSYSSTAT		System Status Register in Power Die (see **Section 9. References**, 9)

Table D-1. Stepper Controller Software Data Variables (Continued)

Name	Components	Description
uAppErrFlags		Application Error Flags register See **Table C-1**, **l_u8_wr_AppErrFlagsA1_1** for details on each components
	HighTemperature	HTF Over Temperature Status Bit
	HB_OverCur	HB_OCF H-Bridge Over Current Flag Bit\
	HighVoltage	HVF H-Bridge High Voltage Bit
	LowVoltage	LVF H-Bridge Low Voltage Bit
	PowerDieError	Power Die error
	LINTxRxError	NOT IMPLEMENTED
	SICurrLim	serial input current limitation
	StopSpeedError	speed was too enough when motor stopped
uAppFlags1		Application status Flags #1 register See **Table C-1**, **l_u8_wr_uAppFlags1A1_1** for details on each components
	StepRun	motor Running flag
	StopTimeout	motor Stop Timeout flag
	PositPark	motor is in Parking Position
	PositResetDone	motor Position Reset Done
	AppInitDone	Application Init Done
	LightOn	Light On status
	StallDetected	Not implemented
	RotDirSign	motor rotation direction signature
uMotStepControlFlags		Motor stepping Control Flags
	RotDir24	rotation direction HB2->HB4
timeMotStep		Motor Controller Time

Appendix E. System Setup

E.1 Hardware Setup

The hardware setup depends on desired functionality of the whole system. There are two main possible setups. The first is for a LIN HID demo application (see **Figure E-1**), where the HID lamp position is driven by a personal computer. The second setup is for programming and debugging slave nodes (see **Figure E-2**). It is also controlled by a PC. Both setups incorporate the following modules:

- HID lamp system with two slave boards
- Master board
- Personal computer
- Power supply +12 V, 5 A (the total power current is dependent on the lamp; in this case it is less than 0,5 A)

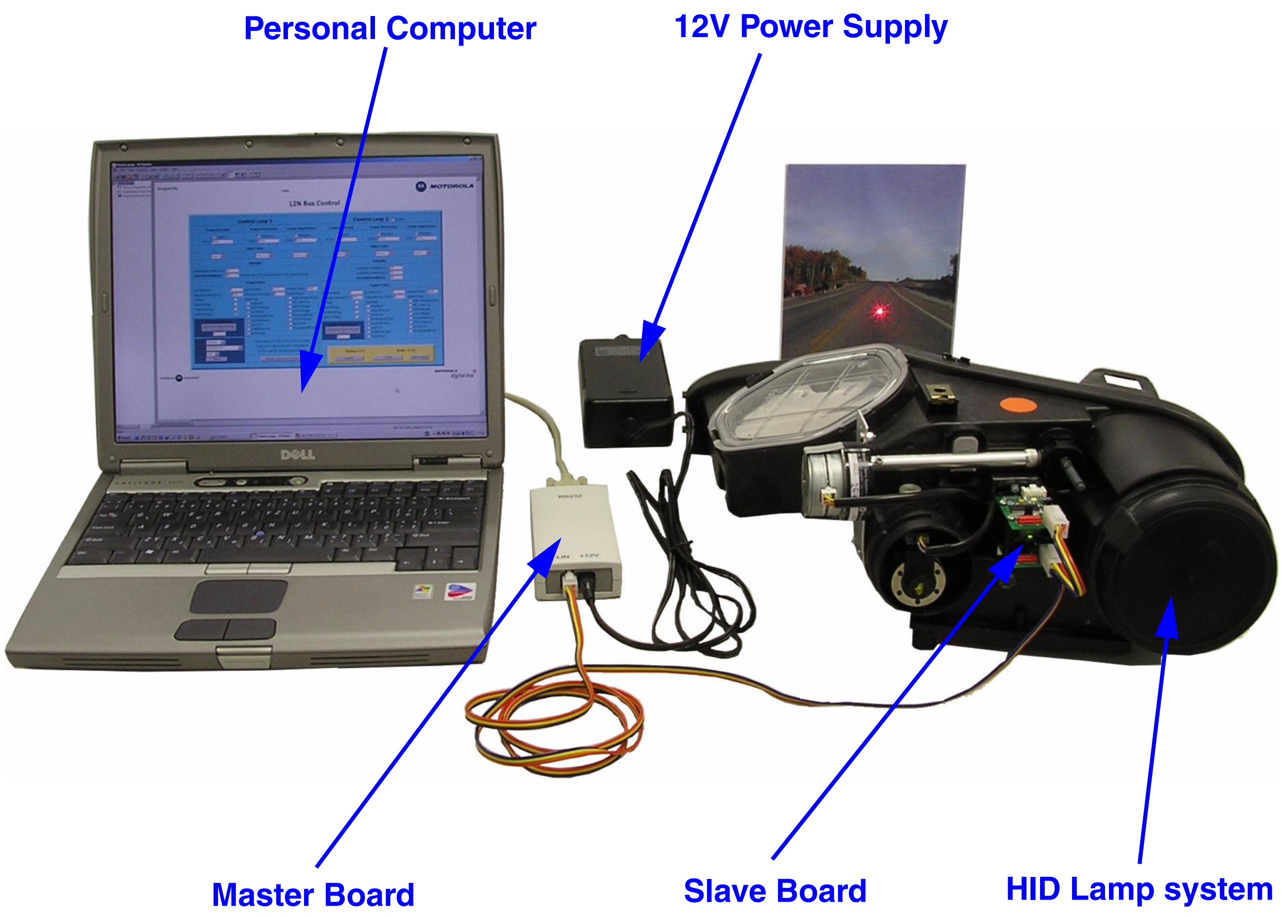


Figure E-1. LIN HID Demo Application

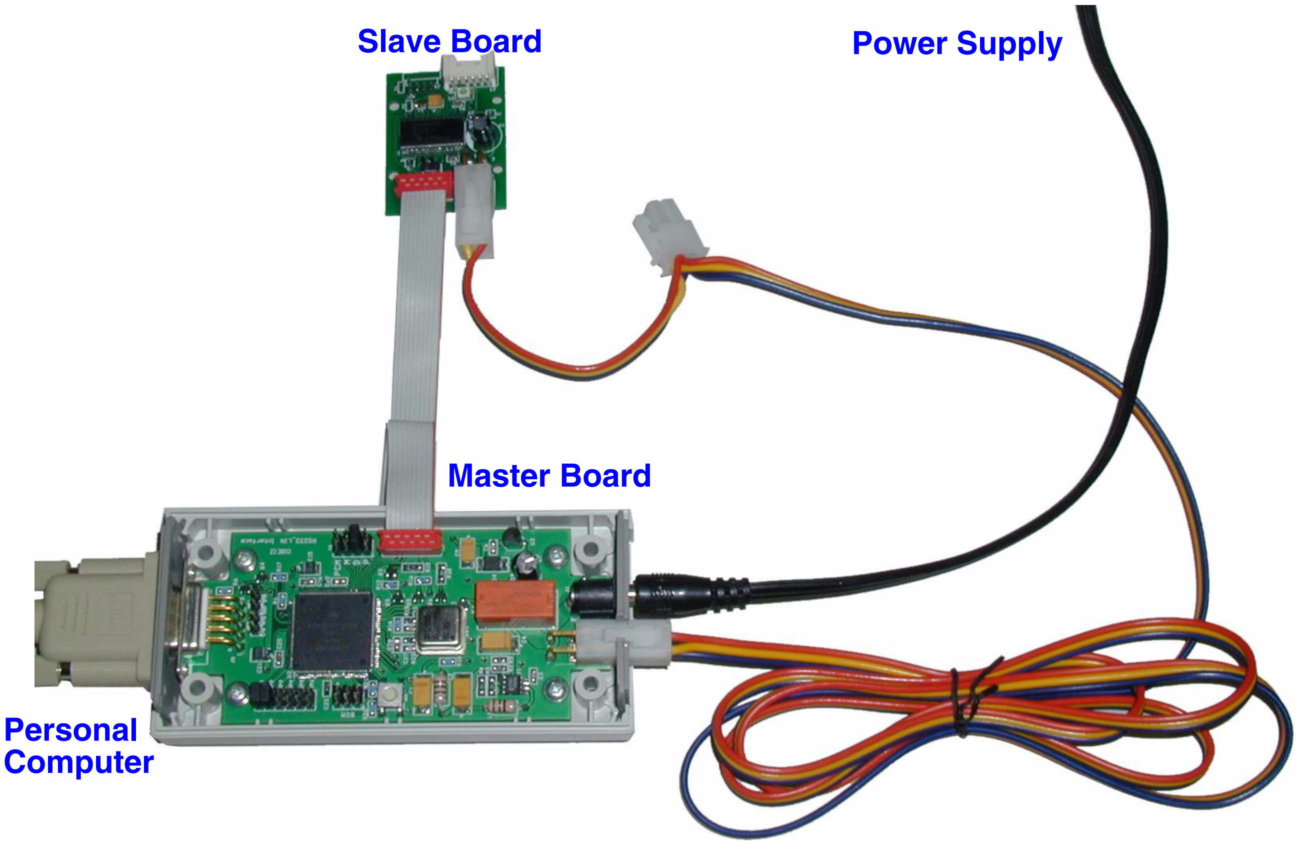


Figure E-2. Programming and Debugging Application - Detail

E.2 Jumper Settings of Master and Slave Boards

The jumper settings depend on the desired device functionality and are specified by **Table E-1.**

Table E-1. Master and Slave Boards Jumper Settings

	System Setup	
	LIN HID demo application	**Programming and Configuration**
Master Board jumper header J2	Jumper to position marked as PCM, click on the Reset button SW1	Jumper to position marked as D, click on the Reset bottom SW1
Slave Board jumper header JP1	Short jumper pins	Open jumper pins

E.3 Required Software Tools

The application requires the following software development tools:

- Metrowerks CodeWarrior for HC08 microcontrollers, version 2.1 or later.
- PEMICRO PROG08SZ Flash/EEprom Programmer HC08 devices using MON08, version 1.68
- Metrowerks CodeWarrior for HC12 microcontrollers with BDM support, version 2.0 or later.
- Microsoft Internet Explorer
- PC Master software tool

E.4 Building and Uploading the Application

the application software is delivered in the folder **lin_leveller**. The master software is located in the sub folder **lin_master**. The slave software is located in the sub folder **lin_stepper**.

E.4.1 LIN Master

The application software is delivered as the **master.mcp** project file with main C-source *master.c* and main header *master.h*. Using Metrowerks CodeWarrior for HC12, the executable file can be created. The executable file is then downloaded into the MCU through the BDM multilink hardware connected to the parallel port on the PC.

E.4.2 LIN Stepper (Slave) Controller

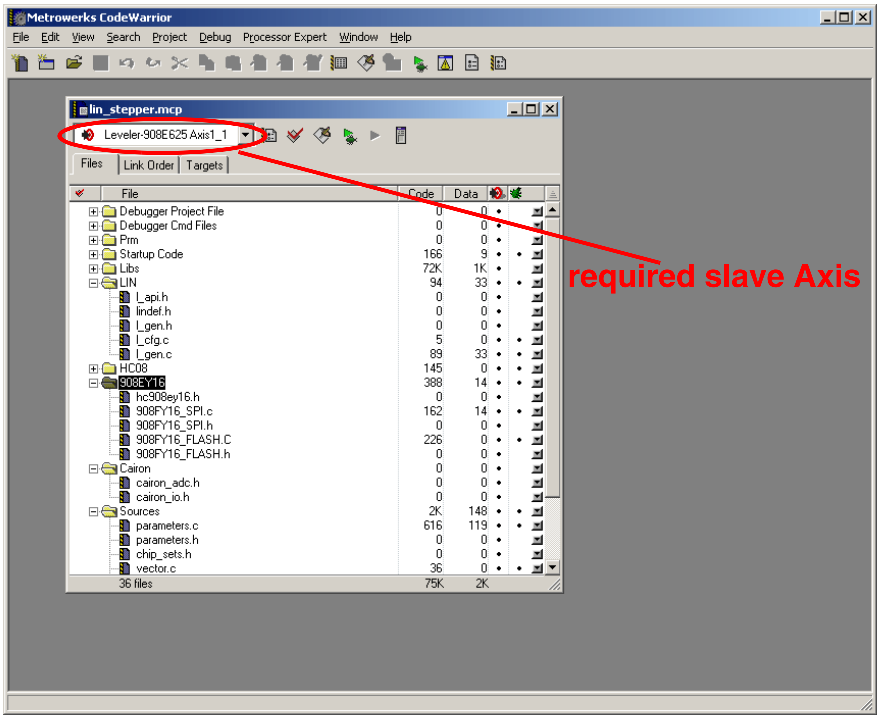
After successfully loading the master software, as described in the previous section, configure the system as shown in **Figure E-2** and described in **Section E.1. Hardware Setup**.

The application software is delivered as the **lin_stepper.mcp** project file with C-source and header files in the sub folder **lin_stepper**. Using Metrowerks CodeWarrior, the executable S19 file **lin_stepper.sx** can be created in the folder **lin_leveller\lin_stepper\bin**. Prior to the compile, the target must be set according to required the axis (see **Figure E-3**). This sets the LIN signal drivers to receive the required signals.

The executable file is then downloaded into the MCU from the PC with the support of LIN Master. All the jumpers must be connected according to **Table E-1. Master and Slave Boards Jumper Settings**, under **Programming and Configuration**).

The software can be loaded using the PEMICRO PROG08SZ Flash programmer. After the programmer is started, the page from **Figure E-5** appears and the parameters must be set as shown.

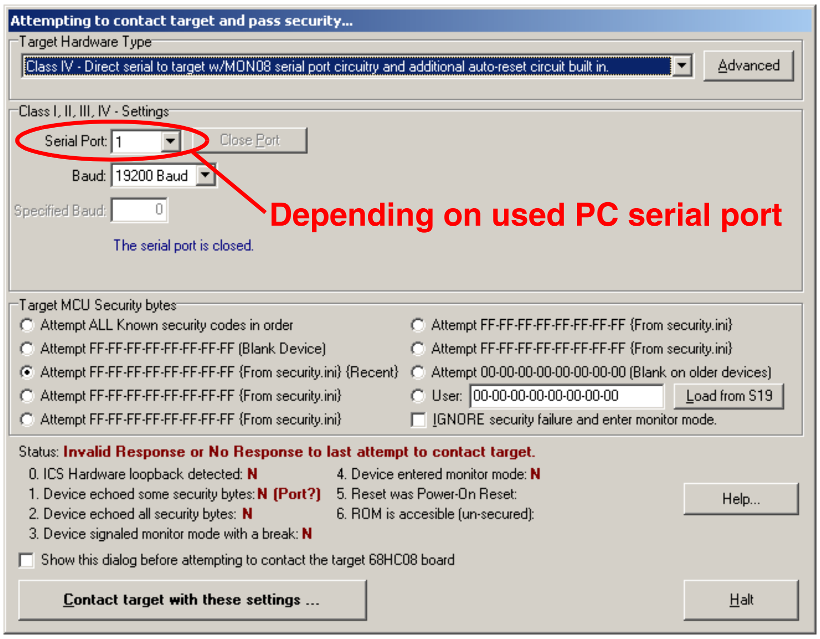
NOTE: *The bootloader communication speed must be set to 19200 baud.*



Leveler-908E625 Axis1_1

File	Code	Data		
Debugger Project File	0	0	•	
Debugger Cmd Files	0	0	•	
Prm	0	0	•	
Startup Code	166	9	•	•
Libs	72K	1K	•	
LIN	94	33	•	•
l_api.h	0	0	•	
lindef.h	0	0	•	
l_gen.h	0	0	•	
l_cfg.c	5	0	•	•
l_gen.c	89	33	•	•
HC08	145	0	•	•
908EY16	388	14	•	•
hc908ey16.h	0	0	•	
908FY16_SPI.c	162	14	•	•
908FY16_SPI.h	0	0	•	
908FY16_FLASH.C	226	0	•	•
908FY16_FLASH.h	0	0	•	
Cairon	0	0	•	
cairon_adc.h	0	0	•	
cairon_io.h	0	0	•	
Sources	2K	148	•	•
parameters.c	616	119	•	•
parameters.h	0	0	•	
chip_sets.h	0	0	•	
vector.c	36	0	•	•
36 files	75K	2K		

Figure E-3. Metrowerks Compiler with lin_stepper.mcp



Attempting to contact target and pass security...

Target Hardware Type

Class IV - Direct serial to target w/MON08 serial port circuitry and additional auto-reset circuit built in.

Class I, II, III, IV - Settings

Serial Port: 1

Baud: 19200 Baud

Specified Baud: 0

The serial port is closed.

Depending on used PC serial port

Target MCU Security bytes

- ( ) Attempt ALL Known security codes in order
- ( ) Attempt FF-FF-FF-FF-FF-FF-FF-FF (Blank Device)
- [✓ Attempt FF-FF-FF-FF-FF-FF-FF-FF (From security.ini) (Recent)]
- ( ) Attempt FF-FF-FF-FF-FF-FF-FF-FF (From security.ini)
- ( ) Attempt FF-FF-FF-FF-FF-FF-FF-FF (From security.ini)
- ( ) Attempt FF-FF-FF-FF-FF-FF-FF-FF (From security.ini)
- ( ) Attempt FF-FF-FF-FF-FF-FF-FF-FF (From security.ini)
- ( ) Attempt 00-00-00-00-00-00-00-00 (Blank on older devices)
- ( ) User: 00-00-00-00-00-00-00-00
- [ ] IGNORE security failure and enter monitor mode.

Status: Invalid Response or No Response to last attempt to contact target.

0. ICS Hardware loopback detected: N
1. Device echoed some security bytes: N (Port?)
2. Device echoed all security bytes: N
3. Device signaled monitor mode with a break: N
4. Device entered monitor mode: N
5. Reset was Power-On Reset:
6. ROM is accesible (un-secured):

[ ] Show this dialog before attempting to contact the target 68HC08 board

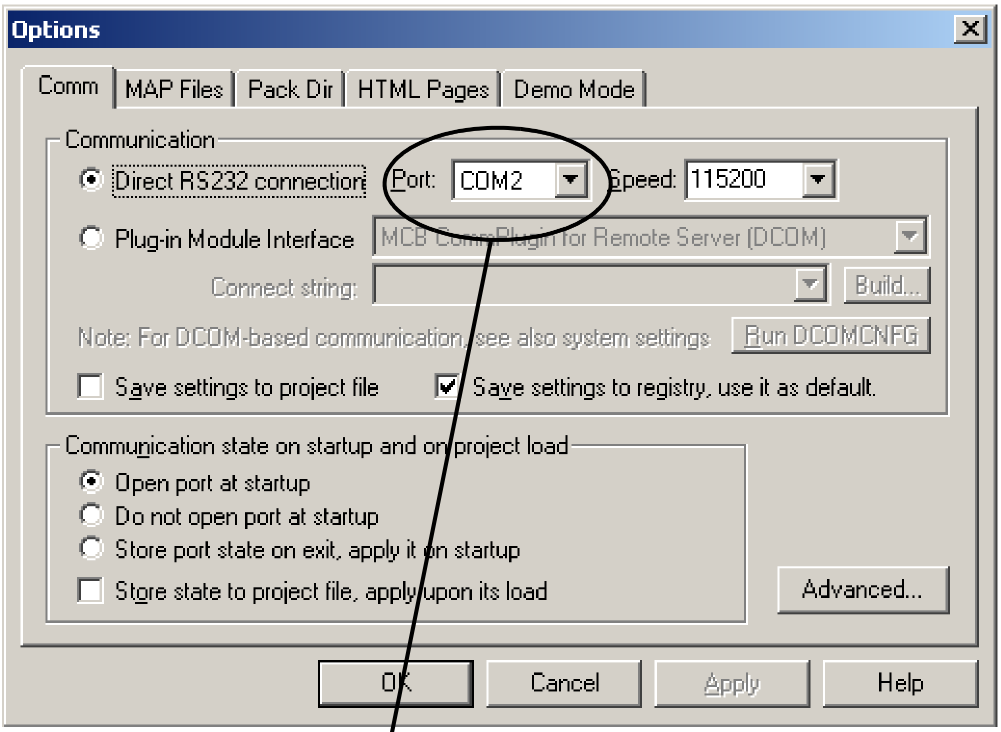
Figure E-4. Bootloader Setting

E.5 Executing the LIN HID Demo Application

The LIN HID demo application is prepared for operation when connected according to **Figure E-1** in **Appendix E.1. Hardware Setup**, with jumpers setting according to **Appendix E.2. Jumper Settings of Master and Slave Boards**.

Run the PC Master tool on the PC via **Master.pmp** in the *...lin_leveller\lin_master\pc_master* folder. Then set two present Project/Options windows as shown **Figure E-5** and **Figure E-6**.

NOTE: *The PC master software and Internet Explorer must be installed to be able to run the **Master.pmp**.*


Options

Comm | MAP Files | Pack Dir | HTML Pages | Demo Mode

Communication

(●) Direct RS232 connection — Port: COM2 — Speed: 115200

( ) Plug-in Module Interface — MCB CommPlugin for Remote Server (DCOM)

Connect string:

Note: For DCOM-based communication, see also system settings — Run DCOMCNFG

[ ] Save settings to project file — [✓] Save settings to registry, use it as default.

Communication state on startup and on project load

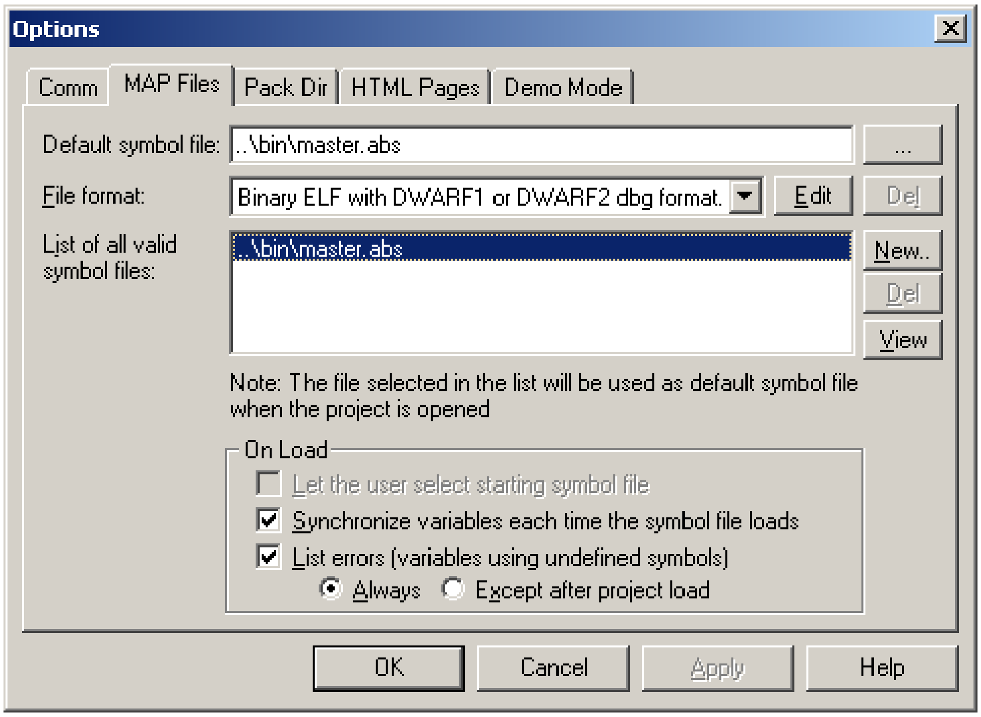
(●) Open port at startup

( ) Do not open port at startup

( ) Store port state on exit, apply it on startup

[ ] Store state to project file, apply upon its load

Figure E-5. Communication Page



Options

Comm | MAP Files | Pack Dir | HTML Pages | Demo Mode

Default symbol file: ..\bin\master.abs

File format: Binary ELF with DWARF1 or DWARF2 dbg format.

List of all valid symbol files: ..\bin\master.abs

Note: The file selected in the list will be used as default symbol file when the project is opened

On Load
- [ ] Let the user select starting symbol file
- [x] Synchronize variables each time the symbol file loads
- [x] List errors (variables using undefined symbols)
  - [✓ Always] Except after project load

Figure E-6. Variables Source page

Now the system is prepared. Its control is described in the User Interface Description.

NOTE: *Do not forget to click on the button RUN at the bottom right corner of the control page, to start sending signals on the bus.*

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://motorola.com/semiconductors

MOTOROLA

DRM047