AN3055
Application note

Getting started with STM8A LIN package (STSW-STM8A-LIN)

Introduction

This document explains how to get started with the STM8A LIN package (STSW-STM8A-LIN).

The STM8A LIN package is a software package containing an implementation of a software driver compliant with the LIN (Local Interconnect Network) standard specifications LIN 1.3, LIN 2.0 and LIN 2.1. This driver implementation is targeted to the STM8A microcontrollers.

Contents

List of tables

List of figures

1 Purpose and scope

The purpose of this document is to give some guidance for the integration of the STM8A LIN package into a user application during code development.

This document is intended to be used in conjunction with the STM8A LIN software package (STSW-STM8A-LIN).

The first part of this document gives a brief introduction to the LIN bus system. Only the information required to configure properly the LIN driver for the application is provided (see *Section 2: Brief introduction to the LIN bus*).

The second part contains detailed information on the LIN driver configuration (*Section 3: LIN driver*).

The third part describes the steps required to integrate the LIN driver into a user application (*Section 4: LIN driver integration into a user application*).

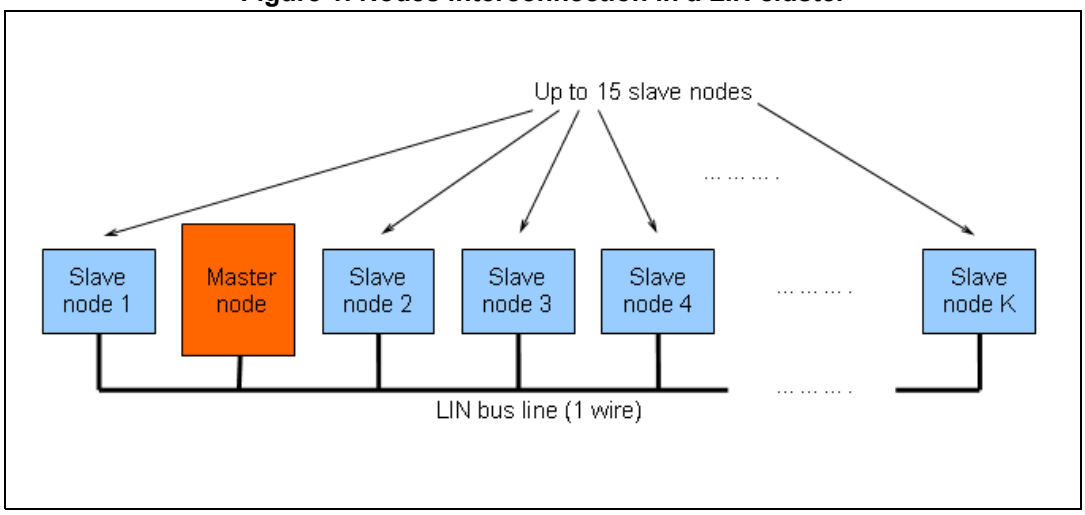
The fourth part explains how to integrate a LIN slave node into an existing LIN cluster when the LIN slave node runs a LIN version protocol which differs from the LIN version protocol used in the existing LIN cluster (*Section 5: LIN versions compatibility*).

2 Brief introduction to the LIN bus

This chapter gives an overview of the LIN bus system. It is mainly focused on the LIN bus structure and basic data exchange on the LIN network. The information is sufficient to get started with the LIN bus system and, with the help of the additional information provided herein, to be able to integrate the LIN driver into the user application and configure it. If more detailed information on the LIN bus are required, refer to a corresponding LIN standard specification document – documents [1], [2] and [3].

The LIN bus system uses low cost single-wire concept to interconnect all the “LIN cluster” members (LIN nodes). All nodes in the LIN cluster are connected to one common “LIN bus line”. The situation is depicted in *Figure 1*. The term “LIN cluster” means a set of LIN nodes (hardware units with LIN interface) which are interconnected via the LIN bus. *Figure 1* shows the structure of a LIN cluster.

Figure 1. Nodes interconnection in a LIN cluster



Two logical levels are defined on the LIN bus physical line – the recessive level and the dominant level. The recessive logical level corresponds to a high voltage level on the LIN bus line (measured against the ground potential, nominal value = 12 V). The dominant logical level corresponds to a low voltage level on the LIN bus line (nominal value = 0 V). The dominant logical level has higher priority. This means that when at least one node in the LIN cluster sends the dominant level, the resulting level on the LIN bus line is the dominant level.

The LIN bus system uses the “single master, multiple slaves” concept. LIN cluster consists of one master node and 1 or more (up to 15) slave nodes.

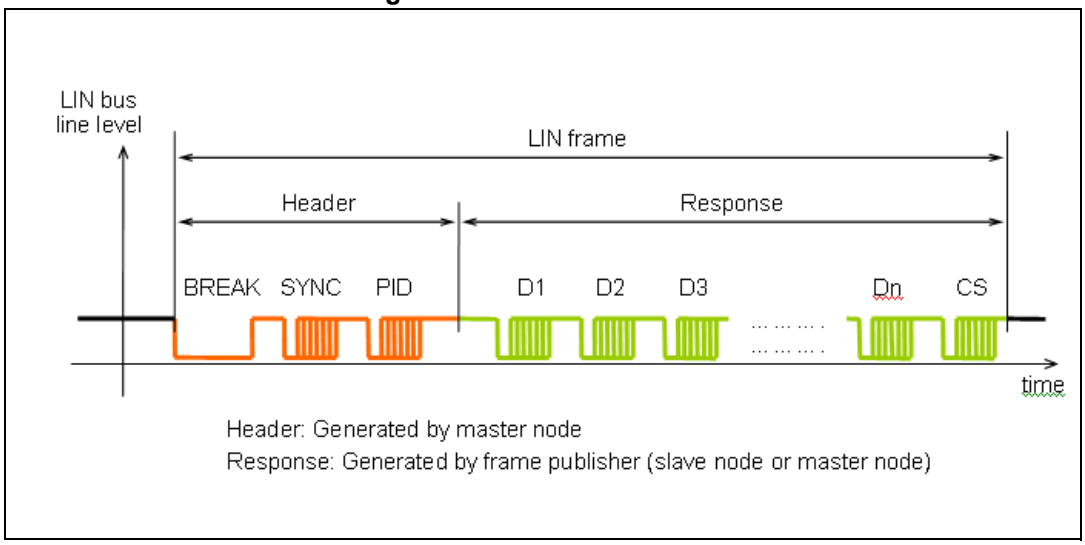
The LIN bus communication speed is up to 20 kbit/s. It is fixed for a particular LIN cluster and cannot be changed dynamically in an operating LIN cluster. It is common to all the nodes in that LIN cluster.

The data is transmitted serially as eight data bytes (LSB first) with one start bit, one stop bit and no parity (see “Detail B” in *Figure 5*).

All LIN nodes in a LIN cluster communicate using frames.

Every LIN frame consists of two main parts – “frame header” and “frame response”. The structure of the LIN frame is shown in *Figure 2*.

Figure 2. LIN frame structure



2.1 Frame header

The LIN frame header consists of three fields:

BREAK field: This special field is used to signal (in conjunction with the SYNC field) the beginning of a new LIN frame. It consists of at least 13 bit times of dominant level on the LIN bus line followed by a break delimiter. The break delimiter is made up of the recessive level on the LIN bus line with the duration of at least one nominal bit time. The details on this field can be seen in the *Figure 5*.

SYNC field: “Synchronization” field. It is a byte field with the data value fixed to 0x55. This field can be used by slave nodes (it is an optional feature of a slave node) to perform the baudrate synchronization procedure – to measure exactly the duration of the bit time period generated by the master node, and use this information for trimming the speed of its internal baudrate generator.

PID field: “Protected identifier” field. This byte field consists of two sub-fields - the “frame identifier” (ID) and the “parity”. Bits 0 to 5 are dedicated for the frame identifier (ID), bits 6 and 7 are dedicated for the frame identifier parity. The position of particular bits in any byte field is shown in the “Detail B” in the *Figure 5*.

The LIN frame header is generated (sent to the LIN bus line) by the master node only. It is received by slave nodes. A slave node detects the beginning of the frame header based on the BREAK field followed by the SYNC field.

2.2 Frame response

The LIN frame response consists of two parts:

DATA: This part consists of 1 up to 8 byte fields of “frame data” (denoted as D1, D2, D3, ….). The number of frame data fields (frame data bytes) is determined by the frame identifier (ID) – for every ID, which is a valid ID in a particular LIN cluster, the number of frame data fields is specified (given by the LIN cluster configuration).

CS field: “Checksum” field. It is a byte field which carries the checksum value. The checksum is computed as inverted eight bit sum with carry over all frame data bytes (so called “classic checksum”) or as inverted eight bit sum with carry over all frame data bytes and the protected identifier (so called “enhanced checksum”).

The LIN frame response is generated (sent to the LIN bus line) by the node (slave node or master node) which is the publisher of the frame (given by the LIN cluster configuration).

If the frame publisher is the master node, the master node generates both the frame parts – frame header and frame response.

If the frame publisher is a slave node, the master node generates the frame header and the slave node answers back by generating the frame response.

2.3 LIN frame slots and schedule tables

When a particular frame has to be transferred (be present at the LIN bus line and be received by LIN cluster nodes), it is identified by “schedule tables”. Schedule tables are defined during the LIN cluster configuration time (LIN cluster development phase). Every schedule table consists of a set of table items organized in a certain order. Every schedule table item contains information on the LIN frame identifier and the time quantum dedicated for a “LIN frame slot” corresponding to the particular LIN frame. The LIN frame slot is a time slot, measured in the “time base ticks”, in which a particular frame can be present on the LIN bus line. Each scheduled LIN frame allocates a LIN frame slot on the LIN bus. Relations among the master node schedule tables, master node time base and LIN frame slots can be seen in *Figure 3*.

Frame slot always starts at a time base tick. The duration of a frame slot must be long enough to accommodate the particular frame even in the worst case.

The order of items (entries) in the schedule table determines the order in which the LIN frames are scheduled.

Schedule tables are used by the master node only. The master can dynamically switch between different schedule tables during the LIN cluster operational time in order to meet various application requirements as well as to meet requirements given by the LIN bus communication protocol.

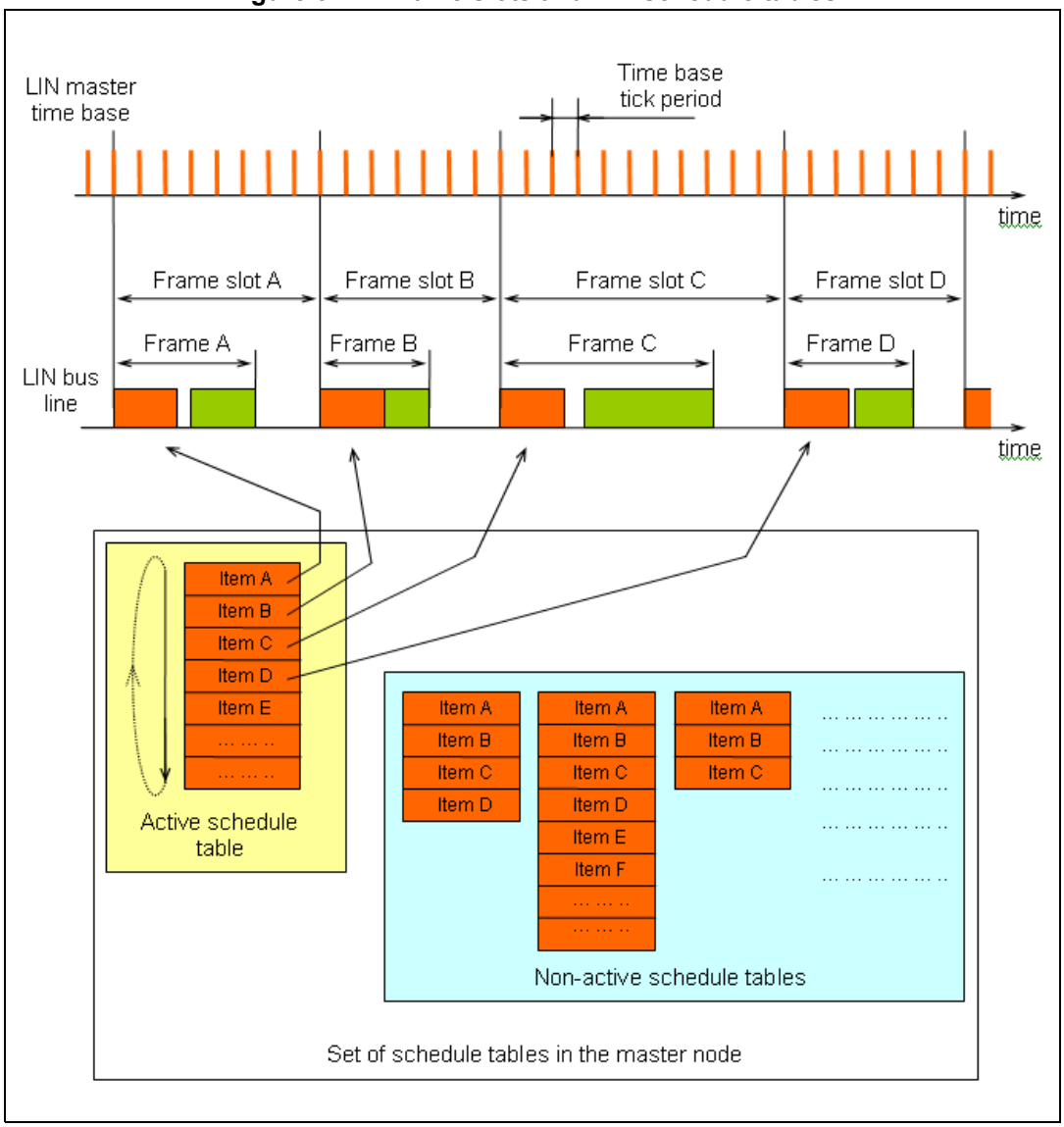
The “active schedule table” is one of the master node schedule tables currently used to schedule the LIN frames on the LIN bus. The “active schedule table” shall be processed until a switch to another schedule table is requested (by the application or by the LIN bus protocol). If the end of the active schedule table is reached, the schedule is started again from its beginning. The actual switch to the new schedule table is made at start of a frame

slot - this ensures that a schedule table switch request is not interrupted any ongoing transmission on the bus.

The minimum time unit used in a LIN cluster is the “time base tick” (see *Figure 3*). The time base is implemented in the master node and is used to control the timing of currently active schedule table. This means that the timing for the frames in a schedule table is based upon the time base of the master node.

The “inter-frame space” is defined as a time period between the end of the frame and the start of the next frame (see *Figure 5*). The inter-frame space must be non-negative.

Figure 3. LIN frame slots and LIN schedule tables



3 LIN driver

This chapter describes the structure of the LIN driver delivery package and explains how to configure the LIN driver configuration. This information is helpful for integrating the LIN driver into the user application.

3.1 LIN package structure

Table 1 shows the directory structure of the LIN driver delivery package together with a content description for every mentioned directory. The term “<LIN_Package_Root>” used in the table represents the path to the LIN driver delivery package root directory (the position of the package root directory in the file system directory structure as well as its name can vary and are chosen by the user during the LIN package installation procedure).

Table 1. LIN package content

Directory	Directory content description
<LIN_Package_Root>\demo	Root directory of LIN driver demo applications.
<LIN_Package_Root>\demo\generic	Platform independent source codes of LIN driver demo applications.
<LIN_Package_Root>\demo\stm8	Source codes of LIN driver demo applications, make files to compile demo applications, LIN driver related configuration files. All these components are to be used for the STM8A device.
<LIN_Package_Root>\doc	LIN driver related documentation (user guide manuals, test reports, etc.).
<LIN_Package_Root>\lingen\bin>	Binary of the “lingen” tool used to generate some source code files with content tightly coupled with user-defined LIN cluster configuration.
<LIN_Package_Root>\make	Make files used to compile the LIN driver source codes.
<LIN_Package_Root>\src	Source codes of the LIN driver.
<LIN_Package_Root>\src\arch	Hardware architecture dependent part of the LIN driver source codes. The STM8A microcontroller family is supported.
<LIN_Package_Root>\src\config	Source code template files containing the LIN driver configuration and some function definitions. All these files are intended to be used as a template for source code files (with the same file names) used in the user application. The integration of the LIN driver into the user application as well as the configuration of the LIN driver is done through these files. To get more information on the LIN driver integration into a user application, please refer to the *Section 4*.
<LIN_Package_Root>\src\diag	Source codes implementing the diagnostic functionality of the LIN driver. Platform independent.
<LIN_Package_Root>\src\general	Source codes implementing some general functionality of the LIN driver.

Table 1. LIN package content (continued)

Directory	Directory content description
<LIN_Package_Root>\src\master	Source code of the LIN driver with implementing LIN master node specific functionality. Platform independent.
<LIN_Package_Root>\src\slave	Source code of the LIN driver with implementing LIN slave node specific functionality. Platform independent.
<LIN_Package_Root>\src\timer	Source codes implementing the LIN driver time base functionality. Platform independent.

3.2 LIN driver configuration

The LIN driver, used in the user application, is configured through set of configuration/ definition files:

“lin_def.h” Contains some macro definitions which control general features and parameters of the LIN driver used by user application. The user defines a set of macros and their values according to the application needs. Content of this configuration file is described in the *Section 3.2.1*.

“lin_def.c” Contains definitions of some functions used by the LIN driver in order to support hardware related functionality as well as user-defined functionality of the LIN driver. Content of this file is described in the *Section 3.2.2*

“lin_def_stm8.h” Contains some macro definitions which control hardware related features and parameters of the LIN driver used by user application. The user defines a set of macros and their values according to the application needs. Content of this configuration file is described in the *Section 3.2.3*.

LIN Definition File (LDF file) A file with the file name extension “.ldf”. It contains description for LIN cluster(s) which the user application (generated LIN node application) operates in. The content of this file must conform to the LIN standard specification. Every LIN cluster is described by one separate LDF file. Therefore, in case the generated LIN node is connected to more than one LIN cluster (master node driving more than one LIN cluster), more then one LDF file exists for such a LIN node. Syntax of a content of this file is a part of the corresponding LIN standard specification document (document [*1*], [*2*] or [*3*]).

Lingen Control File (LCF file) A special control file used by the “Lingen” tool (part of the LIN package, see *Table 1*). The recommended file name extension for this file is “.lgn”. Content of this file describes which LIN interfaces (LIN communication capable peripherals) of the STM8A device are to be used and which LDF file is associated with a particular LIN interface. How to create this file for a user application is described in the *Section 4.1.2*. Syntax and semantic of the content of this file is specified in documents [*4*] and [*5*].

Figure 4 shows the flow to develop the executable binary code of a user application using the LIN driver. To start creating the application executable binary, the following components are needed:

- User application source files (including the file with information for the “linker” tool)
- LIN driver source files (included in the LIN package)
- LIN driver configuration files (“lin_def.h”, “lin_def.c” and “lin_def_stm8.h”)
- LIN cluster description files (LDF file, LCF file)

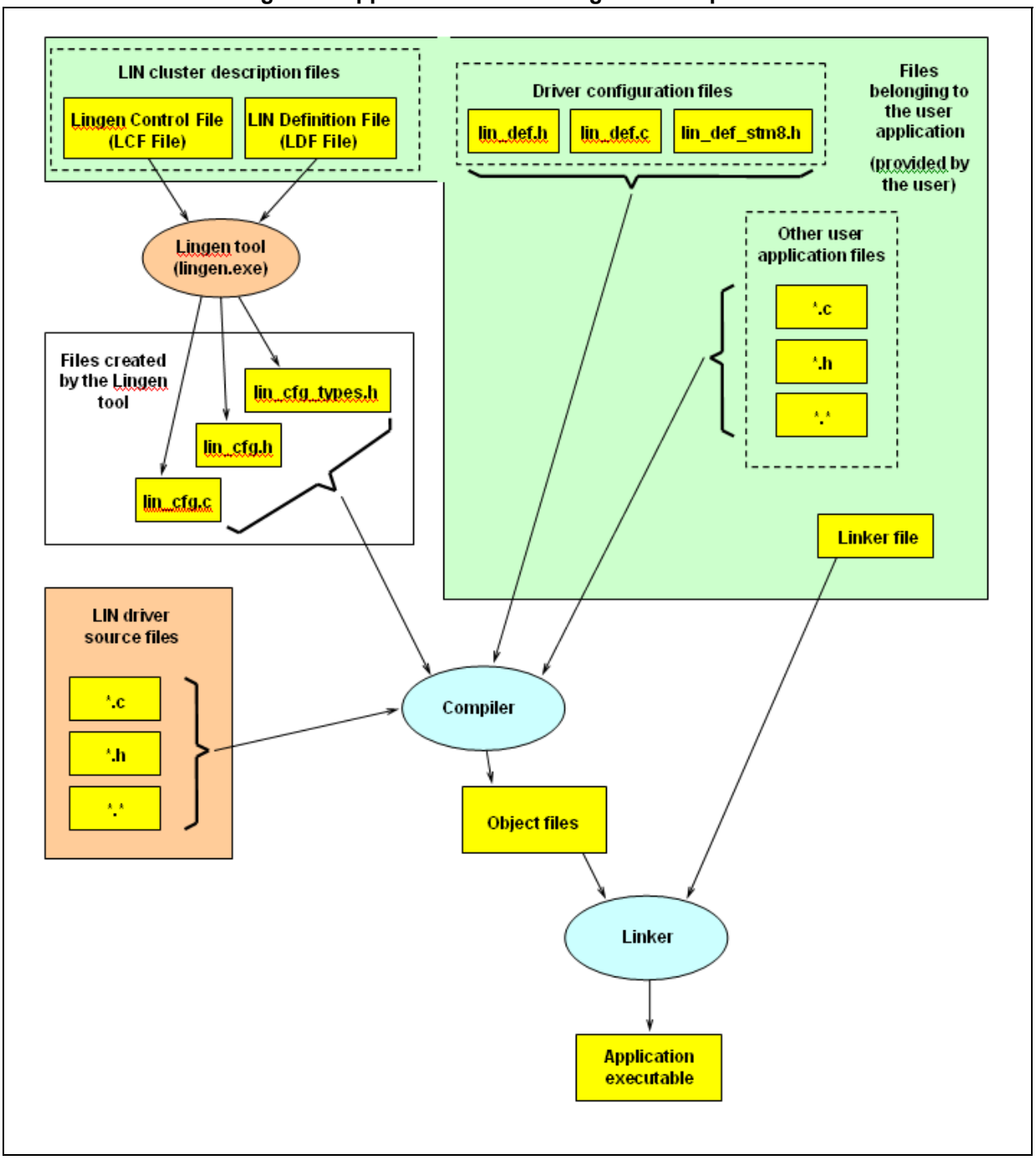
The LIN cluster description files (provided by the user) are processed by the “Lingen” utility and as a result of this processing three source files are created - “lin_cfg.h”, “lin_cfg.c” and “lin_cfg_types.h”. These three files contain some macro and function definitions which are tightly coupled with the LIN cluster configuration specified by the user and are used by the driver API as well as internally by the LIN driver.

The user must configure the features and parameters of the LIN driver through the content of the LIN configuration files.

The LIN configuration files, the files created by the “Lingen” tool, the LIN driver source files and user application source code files are then compiled to get object files of the application.

Finally, the application executable binary is created by a “linker” tool from the object files previously created by the compiling process. The application executable binary is created according to the linking instructions/parameters in the application “linker” file.

Figure 4. Application executable generation process



The following paragraphs take a closer look at the content of each LIN driver configuration file (“lin_def.h”, “lin_def.c” and “lin_def_stm8.h”). Parameters, macro definitions and function definitions are described as well as the relations between them and the scope of their validity. The information should be sufficient to allows a user (with the basic knowledge on the LIN bus communication and protocol) to properly configure the LIN driver according to the user application requirements.

3.2.1 lin_def.h

The “lin_def.h” file is LIN driver configuration file containing architecture independent settings for the LIN driver. This file is a part of the user application code. The LIN package contains a template for the “lin_def.h” file in the “<LIN_Package_Root>\src\config” directory. The user should make a copy of this file in his/her user application area and then change the content of this new file according to the LIN driver configuration needed for the user application.

The following tables provide detailed description for settings in the “lin_def.h” file.

Table 2. LIN_DEVELOPMENT

Macro	Description
Macro name	LIN_DEVELOPMENT
Validity restrictions	None
Description	This macro configures the driver for development version or production version. #define LIN_DEVELOPMENT In case the macro is defined, the development version of the driver is generated. The development version includes a more extensive check on parameters passed to LIN driver API function calls (it could be useful for debugging during the application development phase). #undef LIN_DEVELOPMENT In case the macro is not defined, the production version of the driver is generated. The production version contains just few necessary input parameter checks during LIN driver API function calls which implies. Use this for smaller and faster code.

Table 3. LIN_TIME_BASE_IN_MS

Macro	Description
Macro name	LIN_TIME_BASE_IN_MS
Validity restrictions	None
Description	The value of this macro should correspond to the time base period (in milliseconds) between two consecutive ticks provided by the driver time base timer. The time base period is denoted as the “time base tick period” in *Figure 5* and *Figure 3*. The timer is either a hardware timer or a software timer. The timer is configured by the user in the architecture specific configuration file “lin_def_stm8.h”. To configure the timer means to choose the type of the timer (HW or SW timer) and specify some more parameters needed to properly use the selected timer. The user can refer to the *Section 4.1.4* for more detailed information on driver time base configuration. If a hardware timer is configured to be used as a time base for the driver, the application must configure and enable the interrupts for the chosen hardware timer as well as to define an ISR (Interrupt Service Routine) for the timer in which the “l_timerISR” driver API function is called. The LIN driver sets automatically the timer expiration period so that the timer ISR (defined in the user application) is called with the time period determined by the value of this macro. If a software timer is configured to be used as a time base for the driver, the user application (or operating system) must provide the time base for the driver. For this case, this macro value gives the time period with which the driver API function “l_timer_tick” must be called by the user's application (or operating system). The value of this macro determines the highest possible time granularity in the timing provided by the driver. The recommended value for the time base tick period is either 1 or 2 ms. If a hardware timer is configured to be used as a time base for the driver, this macro can be set to value either 1 or 2. No other values are allowed and only limited number of frequencies of crystal oscillator can be used (see *Section 4.1.4*).

Figure 5. LIN frame structure in details

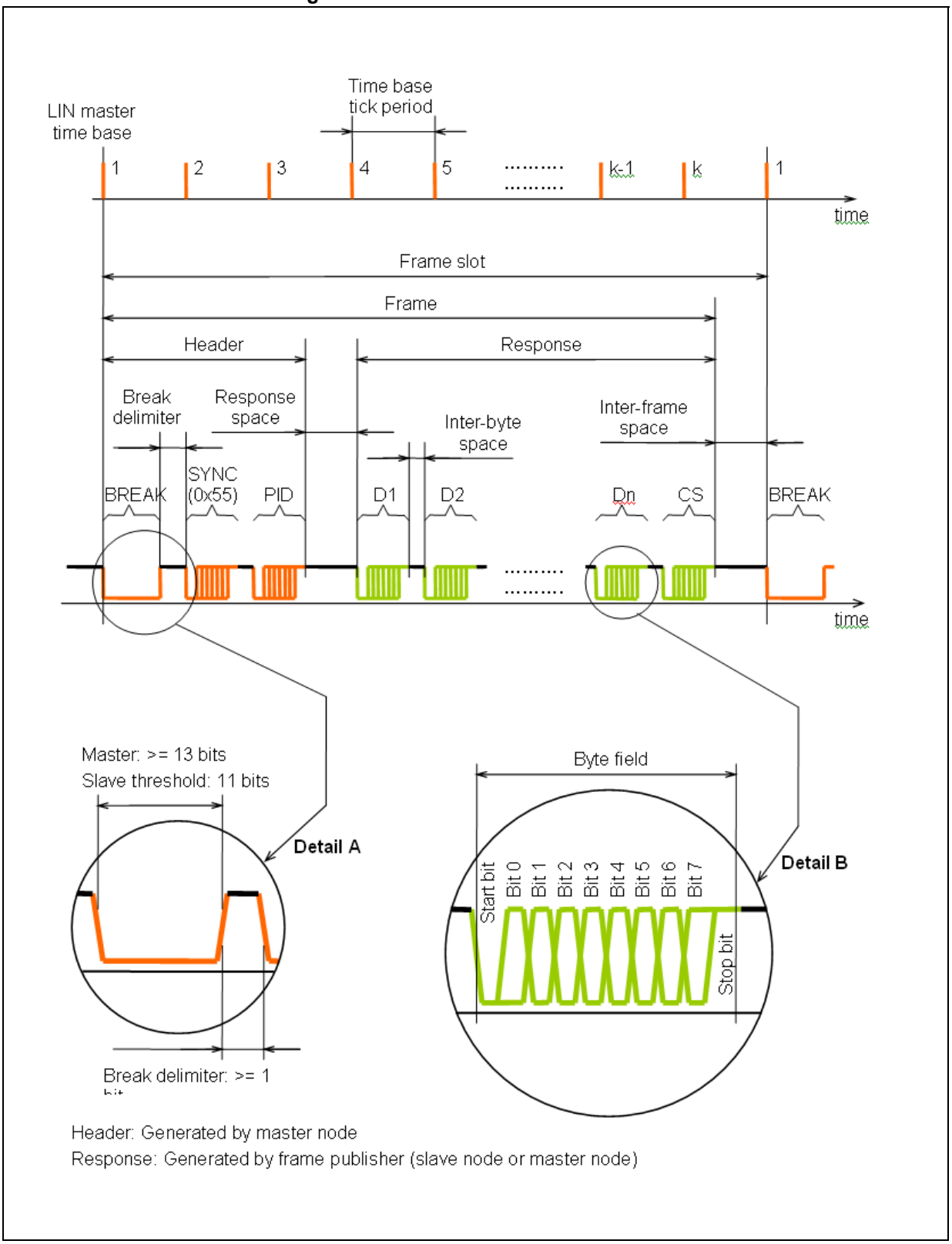


Table 4. LIN_START_BUSSLEEP_TIMER_ON_CONNECT

Macro	Description
Macro name	LIN_START_BUSSLEEP_TIMER_ON_CONNECT
Validity restrictions	Valid only for a slave node.
Description	This macro configures the start-up behavior of a slave node related to the “bussleep” mode. #define LIN_START_BUSSLEEP_TIMER_ON_CONNECT In case the macro is defined, the slave node driver starts the “bussleep timer” after it connects to the LIN network and the slave node enters the “sleep mode” if no activity on the LIN bus line is detected during the time between connecting to the network end the point when the “bussleep timer” elapses. The time after which the “bussleep timer” elapses is configured through the LIN_BUSSLEEP_TIMEOUT_VAL macro (see *Table 23*). For this configuration, it is recommended to set up the master node to send a “wakeup signal” on connecting to the network (described in the *Table 5*) in order to avoid a situation when the slave node enters the “sleep mode” and is not ready to receive frames immediately (possible loss of reception of some first frames). #undef LIN_START_BUSSLEEP_TIMER_ON_CONNECT In case the macro is not defined, the slave node driver does not start the “bussleep timer”. After it connects to the LIN network and therefore the slave node does not enter the “sleep mode” if there is no activity on the LIN bus after connecting to the network. The slave node stays in the “running mode” waiting for some activity on the LIN bus line.

Table 5. LIN_SEND_WAKEUP_SIG_ON_CONNECT

Macro	Description
Macro name	LIN_SEND_WAKEUP_SIG_ON_CONNECT
Validity restrictions	Valid only for a master node.
Description	This macro configures the start-up behavior of a master node related to the “bussleep mode”. #define LIN_SEND_WAKEUP_SIG_ON_CONNECT In case the macro is defined, the master node sends the “wakeup signal” (wakeup sequence) on connecting to the LIN network. It is recommended to use this setting in case that the LIN_START_BUSSLEEP_TIMER_ON_CONNECT macro (see *Table 4*) is defined for any slave node connected to the LIN cluster. #undef LIN_SEND_WAKEUP_SIG_ON_CONNECT In case the macro is not defined, the “wakeup signal” is not sent by the master node right after connecting to the LIN network.

Table 6. LIN_INCLUDE_PID_PARITY_CHECK

Macro	Description
Macro name	LIN_INCLUDE_PID_PARITY_CHECK
Validity restrictions	Valid only for LIN2.0 or LIN2.1 slave node.
Description	When receiving frames, LIN frame PIDs (protected identifiers) are validated against PIDs stored in the slave node (fixed values). However, for PIDs which are (dynamically) assigned by the master node or by the slave application, no validation of PIDs is performed. This macro enables the user to activate the PID parity checking. The parity checking is performed when the PID is (dynamically) assigned by the master or by the slave application to a particular LIN frame during the LIN cluster runtime. #define LIN_INCLUDE_PID_PARITY_CHECK In case the macro is defined, the slave node performs the parity checking of dynamically assigned PIDs (during the PID assigning process only). The new PID is accepted only when the PID parity check passes. #undef LIN_INCLUDE_PID_PARITY_CHECK In case the macro is not defined, the slave node does not perform parity checking of dynamically assigned PIDs during the PID assigning process. This implies that any PID value is accepted (assigned) to a particular LIN frame even if the parity of the PID field is not correct.

Table 7. LIN_INCLUDE_ASSIGN_FRAME_ID

Macro	Description
Macro name	LIN_INCLUDE_ASSIGN_FRAME_ID
Validity restrictions	Valid only for LIN2.0 or LIN2.1 node.
Description	This macro controls availability of the “ld_assign_frame_id” diagnostic function in the driver API in case of master node and the presence of the “Assign frame ID” diagnostic functionality in case of slave node. This function is mandatory for LIN 2.0 nodes and is obsolete for LIN 2.1 nodes. #define LIN_INCLUDE_ASSIGN_FRAME_ID In case the macro is defined, the “ld_assign_frame_id” diagnostic function is available for the application in case of master node and the “Assign frame ID” diagnostic functionality is activated in case of slave node. For LIN 2.0 node, this setting has to be used. #undef LIN_INCLUDE_ASSIGN_FRAME_ID In case the macro is not defined, the “ld_assign_frame_id” diagnostic function is not available for the application in case of master node and the “Assign frame ID” diagnostic functionality is not active in case of slave node.

Table 8. LIN_INCLUDE_ASSIGN_NAD

Macro	Description
Macro name	LIN_INCLUDE_ASSIGN_NAD
Validity restrictions	Valid only for LIN2.0 or LIN2.1 node.
Description	This macro controls the availability of the “ld_assign_NAD” diagnostic function in the driver API in case of master node and the presence of the “Assign NAD” diagnostic functionality in case of slave node. #undef LIN_INCLUDE_ASSIGN_NAD In case the macro is defined, the “ld_assign_NAD” diagnostic function is available for the application in case of master node and the “Assign NAD” diagnostic functionality is activated in case of slave node. #undef LIN_INCLUDE_ASSIGN_NAD In case the macro is not defined, the “ld_assign_NAD” diagnostic function is not available for the application in case of master node and the “Assign NAD” diagnostic functionality is deactivated in case of slave node.

Table 9. LIN_INCLUDE_READ_BY_ID

Macro	Description
Macro name	LIN_INCLUDE_READ_BY_ID
Validity restrictions	Valid only for LIN2.0 or LIN2.1 node.
Description	This macro controls the availability of the “ld_read_by_id” diagnostic function in the driver API in case of master node and the presence of the “Read by ID” diagnostic functionality in case of slave node. This function is mandatory for LIN 2.0 and LIN 2.1 nodes. #define LIN_INCLUDE_READ_BY_ID In case the macro is defined, the “ld_read_by_id” diagnostic function is available for the application in case of master node and the “Read by ID” diagnostic functionality is activated in case of slave node. This setting must be used for LIN 2.0 and LIN 2.1 nodes in order to comply with the LIN standard. In case of a slave node, the “ld_readByIdCallback” function is called during processing the “ld_read_by_id” function call. The functionality of the “ld_readByIdCallback” function is defined by the user. Details on this function can be found in the paragraph *ld_readByIdCallback*. #undef LIN_INCLUDE_READ_BY_ID In case the macro is not defined, the “ld_read_by_id” diagnostic function is not available for the application in case of master node and the “Read by ID” diagnostic functionality is deactivated in case of slave node.

Table 10. LIN_INCLUDE_COND_CHANGE_NAD

Macro	Description
Macro name	LIN_INCLUDE_COND_CHANGE_NAD
Validity restrictions	Valid only for LIN2.0 or LIN2.1 node.
Description	This macro controls the availability of the “ld_conditional_change_NAD” diagnostic function in the driver API in case of master node and the presence of the “Conditional change NAD” diagnostic functionality in case of slave node. #define LIN_INCLUDE_COND_CHANGE_NAD In case the macro is defined, the “ld_conditional_change_NAD” diagnostic function is available for the application in case of master node and the “Conditional change NAD” diagnostic functionality is activated in case of slave node. #undef LIN_INCLUDE_COND_CHANGE_NAD In case the macro is not defined, the “ld_conditional_change_NAD” diagnostic function is not available for the application in case of master node and the “Conditional change NAD” diagnostic functionality is deactivated in case of slave node.

Table 11. LIN_INCLUDE_DATA_DUMP

Macro	Description
Macro name	LIN_INCLUDE_DATA_DUMP
Validity restrictions	Valid only for LIN2.0 or LIN2.1 node.
Description	This macro controls the availability of the “ld_data_dump” diagnostic function in the driver API in case of master node and the presence of the “Data dump” diagnostic functionality in case of slave node. #define LIN_INCLUDE_DATA_DUMP In case the macro is defined, the “ld_data_dump” diagnostic function is available for the application in case of master node and the “Data dump” diagnostic functionality is activated in case of slave node. This setting must be used for LIN 2.0 and LIN 2.1 nodes in order to comply with the LIN standard. In case of a slave node, the “ld_dataDumpCallback” function is called during processing the “ld_dataDumpCallback” function call. The functionality of the “ld_dataDumpCallback” function is defined by the user. Details on this function can be found in the paragraph *ld_dataDumpCallback*. #undef LIN_INCLUDE_DATA_DUMP In case the macro is not defined, the “ld_data_dump” diagnostic function is not available for the application in case of master node and the “Data dump” diagnostic functionality is deactivated in case of slave node.

Table 12. SERIAL_NUMBER

Macro	Description
Macro name	SERIAL_NUMBER
Validity restrictions	Valid only for LIN2.0 or LIN2.1 slave node.
Description	The value of this macro determines the serial number of a particular slave node. Slave node may have a serial number to identify a specific instance of a slave node product. The serial number is 4 bytes long.

Table 13. LIN_INCLUDE_SAVE_CONFIGURATION

Macro	Description
Macro name	LIN_INCLUDE_SAVE_CONFIGURATION
Validity restrictions	Valid only for LIN 2.1 nodes.
Description	This macro controls the availability of the “ld_save_configuration” diagnostic function in the driver API in case of master node and the presence of the “Save configuration” diagnostic functionality in case of slave node. #define LIN_INCLUDE_SAVE_CONFIGURATION In case the macro is defined, the “ld_save_configuration” diagnostic function is available for the application in case of master node and the “Save configuration” diagnostic functionality is activated in case of slave node. #undef LIN_INCLUDE_SAVE_CONFIGURATION In case the macro is not defined, the “ld_save_configuration” diagnostic function is not available for the application in case of master node and the “Save configuration” diagnostic functionality is deactivated in case of slave node.

Table 14. LIN_INCLUDE_ASSIGN_FRAME_ID_RANGE

Macro	Description
Macro name	LIN_INCLUDE_ASSIGN_FRAME_ID_RANGE
Validity restrictions	Valid only for LIN 2.1 master node.
Description	This macro controls the availability of the “ld_assign_frame_id_range” diagnostic function in the driver API. #define LIN_INCLUDE_ASSIGN_FRAME_ID_RANGE In case the macro is defined, the “ld_assign_frame_id_range” diagnostic function is available for the application. This setting must be used for LIN 2.1 nodes in order to comply with the LIN standard. #undef LIN_INCLUDE_ASSIGN_FRAME_ID_RANGE In case the macro is not defined, the “ld_assign_frame_id_range” diagnostic function is not available for the application.

Table 15. LIN_DIAGNOSTIC_CLASS

Macro	Description
Macro name	LIN_DIAGNOSTIC_CLASS
Validity restrictions	Valid only for LIN 2.1 slave node.
Description	LIN 2.1 slave nodes must have a “diagnostic class” value defined through value of this macro. The acceptable macro values are 1, 2 or 3. The value corresponds to a diagnostic class defined by the LIN standard. The diagnostic classes are: DIAGNOSTIC CLASS 1: Only the node configuration services are supported. The slave does not support any other diagnostic services. Single frames (SF) transport protocol support is sufficient. Node Identification is limited to the mandatory “read by identifier” service. DIAGNOSTIC CLASS 2: “Node configuration and identification” services are supported. Full transport layer implementation is required to support multi-frame transmissions. “Node Identification” is extended to all the “read by id” services. Slave nodes supports a set of ISO 14229-1 diagnostic services like “node identification” (SID 0x22), “reading data parameter” (SID 0x22) if applicable, “writing parameters” (SID 0x2E) if applicable. DIAGNOSTIC CLASS 3: “Node configuration and identification” services are supported. Full transport layer implementation is required to support multi-frame transmissions. “Node identification” is extended to all the “read by id” services. Slave nodes shall support all services as of “Class 2”. Additionally, other services may be supported depending on the features which are implemented by the slave node: for example “session control” (SID 0x10), “I/O control by identifier” (0x2F), “read and clear DTC” (SID 0x19, 0x14). Only “Class 3” slave nodes can reprogram the application via the LIN bus.

Table 16. LIN_INCLUDE_COOKED_TP

Macro	Description
Macro name	LIN_INCLUDE_COOKED_TP
Validity restrictions	Valid only for LIN2.0 or LIN2.1 node.
Description	This macro controls the availability of a set of diagnostic functions which belongs to, so called, “Cooked” part of the “diagnostic transport layer API” of the driver. This set of diagnostic functions (Cooked TP diagnostics functions) includes “ld_send_message”, “ld_receive_message”, “ld_tx_status” and “ld_rx_status” functions. #define LIN_INCLUDE_COOKED_TP In case the macro is defined, the Cooked TP diagnostic functions are available for the application. #undef LIN_INCLUDE_COOKED_TP In case the macro is not defined, the Cooked TP diagnostic functions are not available for the application.

Table 17. LIN_INCLUDE_RAW_TP

Macro	Description
Macro name	LIN_INCLUDE_RAW_TP
Validity restrictions	Valid only for LIN2.0 or LIN2.1 node.
Description	This macro controls the availability of set of diagnostic functions which belongs to, so called, “Raw” part of the “diagnostic transport layer API” of the driver. This set of diagnostic functions (Raw TP diagnostics functions) includes “ld_put_raw”, “ld_get_raw”, “ld_raw_tx_status”, “ld_raw_rx_status” and “ld_raw_tx_delete” functions. The function “ld_raw_tx_delete” is not a function defined by the LIN standard. This function is a driver API extension implemented in order to enable the user to clear the driver internal Raw TP Transmit FIFO buffer. The size of the buffer is set through the LIN_DIAG3_FIFO_SIZE_MAX macro (described in the *Table 18*). #define LIN_INCLUDE_RAW_TP In case the macro is defined, the Raw TP (Transport Protocol) diagnostic functions are available for the application. #undef LIN_INCLUDE_RAW_TP In case the macro is not defined, the Raw TP diagnostic functions are not available for the application.

Table 18. LIN_DIAG3_FIFO_SIZE_MAX

Macro	Description
Macro name	LIN_DIAG3_FIFO_SIZE_MAX
Validity restrictions	Valid only for LIN2.0 or LIN2.1 node when the LIN_INCLUDE_RAW_TP macro (see *Table 17*) is defined.
Description	The value of this macro determines the maximum number of frames stored in the in the Raw TP (Transport Protocol) Transmit FIFO buffer as well as the maximum number of frames stored in the Raw TP Receive FIFO buffer. The macro value can be set to a number from 1 up to 32.

Table 19. LIN_BAUDRATE_DETECT

Macro	Description
Macro name	LIN_BAUDRATE_DETECT
Validity restrictions	Valid only for a slave node.
Description	This macro enables the baudrate detection feature in the driver. The baudrate detection works by the principle that the application starts to communicate with the highest possible baudrate and then repeatedly tries by lowering the baudrate until the communication is established. #define LIN_BAUDRATE_DETECT In case the macro is defined, the baudrate detection feature is activated in the driver. For that case, the “l_baudrate_callback_iii” callback function must be defined by the user. The “iii” in the callback function name corresponds to the LIN interface name – here it is “LIN_IFC_SCI1” or “LIN_IFC_SCI2”. Please refer to the *l_baudrate_callback_iii* to get more information on the “l_baudrate_callback_iii” callback function. #undef LIN_BAUDRATE_DETECT In case the macro is not defined, the baudrate detection feature is not available.

Table 20. LIN_PROTOCOL_SWITCH

Macro	Description
Macro name	LIN_PROTOCOL_SWITCH
Validity restrictions	Valid only for LIN2.0 or LIN2.1 node.
Description	This macro enables the protocol switching feature in the driver. The protocol switching enables the user to switch between the LIN protocol and an alternative communication protocol on the LIN bus. #define LIN_PROTOCOL_SWITCH In case the macro is defined, the baudrate detection feature is activated in the driver. For that case, the “l_protocolCallback_iii” callback function must be defined by the user. The “iii” in the callback function name corresponds to the LIN interface name “LIN_IFC_SCI1” or “LIN_IFC_SCI2”. Refer to the paragraph *l_baudrate_callback_iii* to get more information on the “l_protocolCallback_iii” callback function. #undef LIN_PROTOCOL_SWITCH In case the macro is not defined, the baudrate detection feature is not available.

Table 21. LIN_DELAY_INIT

Macro	Description
Macro name	LIN_DELAY_INIT
Validity restrictions	Valid for a master node only.
Description	When switching from L_NULL_SCHEDULE (no LIN schedule table is active) to a valid LIN schedule table after the startup, the initial delay is undefined. The value of this macro determines (in time base ticks) this initial delay.

Table 22. LIN_INCLUDE_2x_SLEEP_MODE_API

Macro	Description
Macro name	LIN_INCLUDE_2x_SLEEP_MODE_API
Validity restrictions	Valid only when LIN 1.3 protocol is configured.
Description	This macro controls the availability of the “l_ifc_goto_sleep” and “l_ifc_wake_up” sleep/wakeup driver API functions for a case when the LIN 1.3 protocol is used.

Table 23. LIN_BUSSLEEP_TIMEOUT_VAL

Macro	Description
Macro name	LIN_BUSSLEEP_TIMEOUT_VAL
Validity restrictions	Valid only for LIN2.0 or LIN2.1 slave node.
Description	The value of this macro determines the “bussleep timeout” time (in milliseconds). The bussleep timeout time is a time period without any activity on the LIN bus line after which the slave node switches to the “bussleep mode”. How the bussleep timeout is measured is depicted in the *Table 6*. The recommended default value given by the LIN standard is 4000 ms.

Table 24. LIN_WAKEUP_TIMEOUT_VAL_SHORT

Macro	Description
Macro name	LIN_WAKEUP_TIMEOUT_VAL_SHORT
Validity restrictions	Valid only for LIN2.0 or LIN2.1 slave node.
Description	The value of this macro determines the time (in milliseconds) after which the slave node sends first two retries of the wakeup signal (signal sent to the master node in case the LIN cluster is in the “sleep state” and the slave needs to communicate) in case that the master node do not react on “wakeup” signals. In the *Table 7*, this time is denoted as the “short wakeup request timeout”. There is a link between this macro value and the macro LIN_MASTER_WAKEUP_TIMER_VALUE. For more information on this dependency, please refer to the *Table 26*. The default value given by the LIN standard is 150 ms.

Table 25. LIN_WAKEUP_TIMEOUT_VAL_LONG

Macro	Description
Macro name	LIN_WAKEUP_TIMEOUT_VAL_LONG
Validity restrictions	Valid only for LIN2.0 or LIN2.1 slave node.
Description	The value of this macro determines the time (in milliseconds) after which the slave node sends another retry of the “wakeup” signal (signal sent to the master node in case the LIN cluster is in the “sleep state” and the slave needs to communicate) in case that 3 wakeup signals were already sent without any reaction from the master node. In the *Figure 7*, this time is denoted as “long wakeup request timeout”. The default value given by the LIN standard is 1500 ms.

Figure 6. Slave node switching to the sleep state

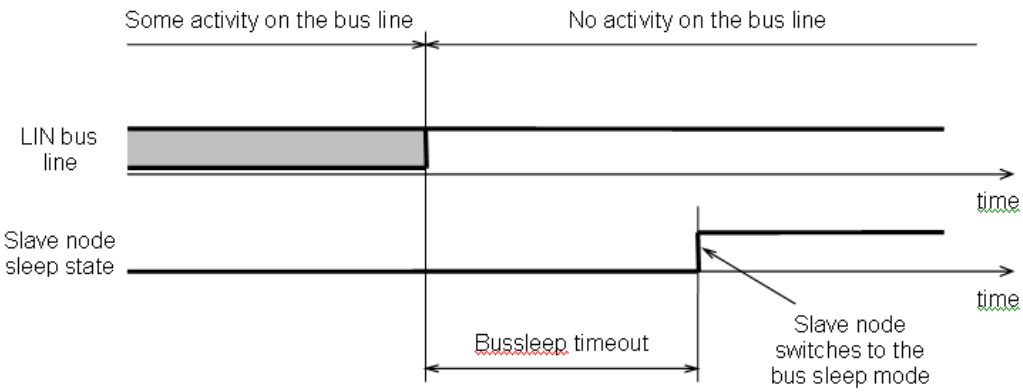


Figure 7. Slave node wakeup signaling

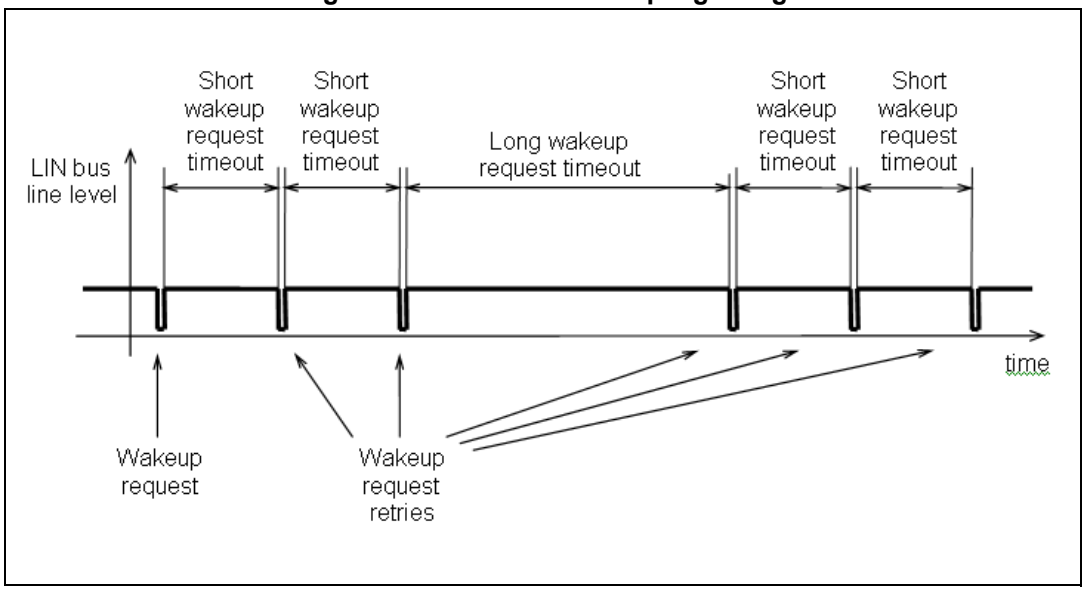


Table 26. LIN_MASTER_WAKEUP_TIMER_VALUE

Macro	Description
Macro name	LIN_MASTER_WAKEUP_TIMER_VALUE
Validity restrictions	Valid only for LIN2.0 or LIN2.1 master node.
Description	The value of this macro determines the time period (in milliseconds) after which the master node starts to send frames after it has received the “wakeup” signal from a slave node. The situation is documented in the *Figure 8* and the time period affected by this macro value is denoted as “maximum slave node wakeup time”. Nodes in a LIN cluster receive the wakeup request signal from a slave node. After receiving the wakeup request, all slave nodes in the LIN cluster must be ready to receive frames within a certain period of time (the LIN standard gives the value 100 ms). A master node can start to send frames after this time period has elapsed. Additionally, the first frame from the master node has to be sent within the time period given by the value of the LIN_WAKEUP_TIMEOUT_VAL_SHORT macro (default value 150 ms, described in the *Table 24*). The time period is measured from the end of the slave node wakeup request signal to the beginning of the BREAK field of the first frame header (generated by the master node). This means that the slave node has some time to get ready to receive frames after receiving the wakeup request signal. If it is known that slaves is ready to receive frames in a shorter time frame than the time specified by the LIN standard, then the value of this macro can be set to a lower value so that the master starts sending frames earlier after receiving the wakeup request signal. The default value given by the LIN standard is 100 ms.

Figure 8. Waking up from the LIN cluster sleep state

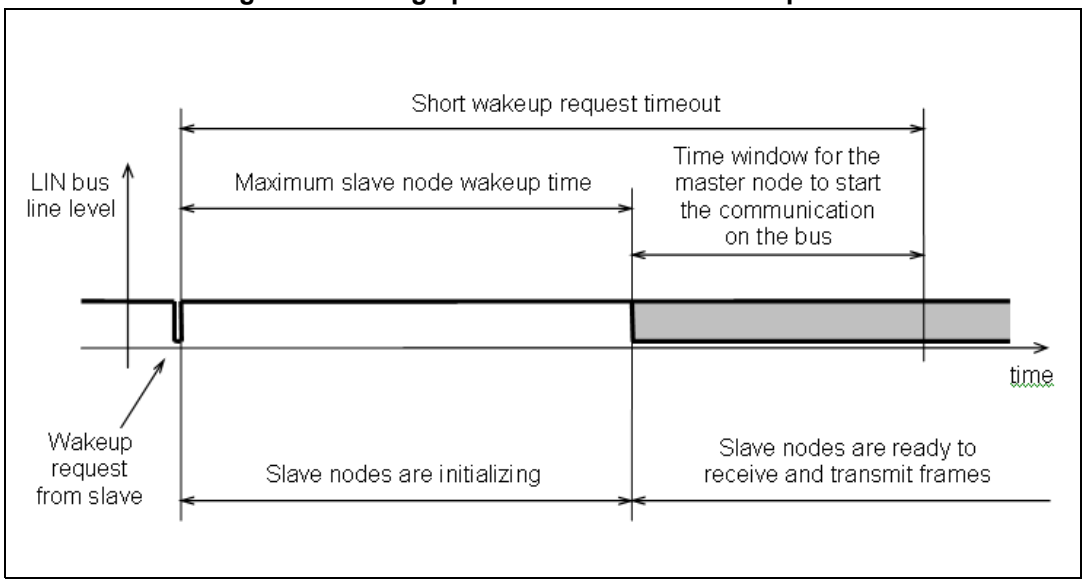


Table 27. LIN_WAKEUP_RETRIES_MAX

Macro	Description
Macro name	LIN_WAKEUP_RETRIES_MAX
Validity restrictions	Valid for a slave node only.
Description	The value of this macro determines the maximum number of retries to send the “wakeup” request signal to the master node in case the master mode do not react (master node doesn’t start to send frame headers) to wakeup request signals from a slave node.

Table 28. LIN_RESPONSE_ERROR_WHEN_INCOMPLETE_RX

Macro	Description
Macro name	LIN_RESPONSE_ERROR_WHEN_INCOMPLETE_RX
Validity restrictions	Valid only for LIN2.0 slave node.
Description	This macro controls the behavior of the “response_error” signal in a situation that the slave node has received any LIN frame in which the slave node is interested in and this LIN frame has been received as incomplete. The LIN2.0 standard does not define the behavior of the “response_error” signal on reception of an incomplete LIN frame. The LIN frame incompleteness means that any data byte field and/or the checksum field of the LIN frame is missing in the LIN frame response. #define LIN_RESPONSE_ERROR_WHEN_INCOMPLETE_RX In case the macro is defined, the “response_error” signal is set if the slave node receives an incomplete LIN frame. #undef LIN_RESPONSE_ERROR_WHEN_INCOMPLETE_RX In case the macro is not defined, the reception of an incomplete LIN frame is not affected the “response_error” signal state.

Table 29. N_AS_TIMEOUT

Macro	Description
Macro name	N_AS_TIMEOUT
Validity restrictions	Valid only for LIN2.1 master node when the LIN_INCLUDE_COOKED_TP macro (see *Table 16*) is defined.
Description	This macro controls and this macro value sets the value (in milliseconds) of the “N_As_timeout” parameter for the LIN2.1 master node. The “N_As_timeout” parameter is defined by the LIN standard (document [*3*]). #define N_AS_TIMEOUT numeric_value In case the macro is defined and its value is assigned with a numeric value, the value of this macro is used as the “N_As_timeout” parameter. #undef N_AS_TIMEOUT In case the macro is not defined, the value of the “N_As_timeout” parameter is set to the default timeout value. The default timeout value is 1000.

Table 30. N_CR_TIMEOUT

Macro	Description
Macro name	N_CR_TIMEOUT
Validity restrictions	Valid only for LIN2.1 node when the LIN_INCLUDE_COOKED_TP macro (see *Table 16*) is defined.
Description	This macro controls and this macro value sets the value (in milliseconds) of the “N_Cr_timeout” parameter for the LIN2.1 master node. The “N_Cr_timeout” parameter is defined by the LIN standard (document [*3*]). #define N_CR_TIMEOUT numeric_value In case the macro is defined and its value is assigned with a numeric value, the value of this macro is used as the “N_Cr_timeout” parameter. #undef N_CR_TIMEOUT In case the macro is not defined, the value of the “N_Cr_timeout” parameter is set to the default timeout value. The default timeout value is 1000.

3.2.2 lin_def.c

The “lin_def.c” file is a file containing source codes for few user-defined functions used by the LIN driver. This file is a part of the user application code. The LIN package contains a template for the “lin_def.c” file in the “<LIN_Package_Root>\src\config” directory. The user should make a copy of this file in his/her user application area and the implement bodies of callback/callout functions inside this new file.

There are few functions to be implemented in the “lin_def.c” file. The description on these functions follows.

l_sys_irq_disable and l_sys_irq_restore

These callback functions must be provided by the user as a part of the application code. They are used by the driver to disable and enable interrupts in the system if needed.

These two callback functions must be able to handle nested calls. This means that the “l_sys_irq_disable” function can be called more than once before calling the “l_sys_irq_restore” function and for such a case the disabled interrupts shall only be re-enabled at the outermost call to the “l_sys_irq_restore” function.

Table 31. l_sys_irq_disable

Callback function	Description
Function prototype	l_irqmask l_sys_irq_disable (void)
Validity restrictions	None
Parameters	None
Return value	Interrupt mask describing the state of the interrupts at time of call. If needed, it can be used to re-enable the interrupts.
Description	Disables interrupts in the system. It must be able to handle nested calls.

Table 32. l_sys_irq_restore

Callback function	Description
Function prototype	void l_sys_irq_restore (l_irqmask irqMask)
Validity restrictions	None
Parameters	irqMask – interrupt mask containing state of interrupts to be restored
Return value	None
Description	Restores the state of interrupts in the system as given by irqMask parameter. It must be able to handle nested calls.

ld_readByIdCallback

The user must provide implementation of this callback function in case of a LIN2.0 or LIN2.1 slave node with the LIN_INCLUDE_READ_BY_ID macro defined (described in *Table 9*) in the “lin_def.h” configuration file.

This function is called by the diagnostic module when the “read by identifier” service request with the “user defined” identifier is received by the slave node. This function implements slave node reactions (responses) on “read by identifier” service requests with “user defined” identifiers.

Table 33. ld_readByIdCallback

Callback function	Description
Function prototype	l_bool ld_readByIdCallback (l_u8 id, l_u8 * pBuffer)
Validity restrictions	Valid only for LIN2.0 or LIN2.1 slave node in case the LIN_INCLUDE_READ_BY_ID macro (see *Table 9*) is defined.
Parameters	id – request identifier pBuffer - buffer to compose the answer to
Return value	1 if application has filled response buffer 0 if no response has been provided
Description	Callback function for “read by identifier” service.

ld_read_by_id_callout

The user must provide implementation of this callout function in case of a LIN 2.1 slave node with the LIN_INCLUDE_READ_BY_ID macro defined (macro is described in *Table 9*) or the LIN_INCLUDE_COND_CHANGE_NAD macro defined (described in *Table 10*) in the “lin_def.h” configuration file.

This callout is used when the master node transmits a “read by identifier” request with an identifier in the “user defined” area. The slave node application is called from the driver when such request is received. This function implements slave node reactions (responses) on “read by identifier” service requests with “user defined” identifiers.

Table 34. ld_read_by_id_callout

Callback function	Description
Function prototype	l_u8 ld_read_by_id_callout(l_u8 id, l_u8* data)
Validity restrictions	Valid only for LIN 2.1 slave node in case the LIN_INCLUDE_READ_BY_ID macro (see *Table 9*) or/and the LIN_INCLUDE_COND_CHANGE_NAD macro (see *Table 10*) is/are defined.
Parameters	id - request identifier pBuffer - buffer to compose answer to
Return value	LD_NEGATIVE_RESPONSE - The slave node answers with a negative response. In this case the data area is not considered. LD_POSTIVE_RESPONSE - The slave node makes a positive response using the data provided by the application. LD_NO_RESPONSE - The slave node does not answer.
Description	Callout function for “read by identifier” service.

l_protocolCallback_iii

The user must provide implementation of this callback function in case of a LIN2.0 or LIN2.1 slave node with the LIN_PROTOCOL_SWITCH macro defined (described in *Table 20*) in the “lin_def.h” configuration file.

This function is called by the ISR (Interrupt Service Routine) when an interrupt from the corresponding LIN peripheral occurs after the application has called the “l_protocol_switch” API function to disable the LIN protocol. This user-defined function is then responsible to handle directly the LIN peripheral and to implement all the functionality of an alternative communication protocol.

Table 35. l_protocolCallback_iii

Callback function	Description
Function prototype	void l_protocolCallback_iii (void) where “iii” denotes LIN interface – “LIN_IFC_SCI1” or “LIN_IFC_SCI2”.
Validity restrictions	Valid only for LIN2.0 or LIN2.1 node in case the LIN_PROTOCOL_SWITCH macro (see *Table 20*) is defined.
Parameters	None
Return value	None
Description	Function is called by the LIN peripheral ISR when the alternative communication protocol is enabled and an interrupt from corresponding LIN peripheral occurs.

l_baudrate_callback_iii

The user must provide implementation of this callback function in case of a LIN slave node with the LIN_BAUDRATE_DETECT macro defined (macro is described in *Table 19*) in the “lin_def.h” configuration file.

This callback function is called by the driver when an incorrect baudrate is detected during the baudrate detection procedure. From this callback, the application can then call the “l_change_baudrate” function to reduce the current baudrate.

The baudrate detection works by the principle that the application starts to communicate with the highest possible baudrate and then repeatedly tries by lowering the baudrate until the communication is established.

Table 36. l_baudrate_callback_iii

Callback function	Description
Function prototype	void l_baudrate_callback_iii (l_u16 baudrate) where “iii” denotes LIN interface – “LIN_IFC_SCI1” or “LIN_IFC_SCI2”.
Validity restrictions	Valid only for slave node in case the LIN_BAUDRATE_DETECT macro (see *Table 19*) is defined.
Parameters	baudrate - the baudrate currently detected (i.e. the incorrect baudrate) on the interface.
Return value	None
Description	This callback is called if an incorrect (too high) baudrate is detected by the slave node during the baudrate detection procedure. It sets new baudrate (lower baudrate) for the given interface by making call to the “l_change_baudrate” driver API function.

ld_dataDumpCallback

The user must provide implementation of this callback function in case of a LIN slave node with the LIN_INCLUDE_DATA_DUMP macro defined (macro described in *Table 11*) in the “lin_def.h” configuration file.

This function is called by the diagnostic module when the “data dump” service request is received by the slave node. The function implements a user-defined reaction of the application on the “data dump” service request sent by the master node.

Table 37. ld_dataDumpCallback

Callback function	Description
Function prototype	l_bool ld_dataDumpCallback (const l_u8 * sendBuf, l_u8 * recBuf)
Validity restrictions	Valid only for slave node in case the LIN_INCLUDE_DATA_DUMP macro (see *Table 11*) is defined.
Parameters	sendBuf - buffer holding master request recBuf - buffer to compose the answer to
Return value	1 if application has filled response buffer 0 if no response has been provided
Description	The callback function for “data dump” service

3.2.3 lin_def_stm8.h

The “lin_def_stm8.h” file is a file containing STM8A architecture dependent settings for the LIN driver. This file is a part of the user application code. The LIN package contains a template for the “lin_def_stm8.h” file in the “<LIN_Package_Root>\src\config” directory. The user should make a copy of this file in his/her user application area and then change the

content of this new file according to the LIN driver configuration needed for the user application.

The following tables provide detailed description for settings in the “lin_def_stm8.h” file.

Table 38. LIN_BOARD_CPU_FREQ_HZ

Macro	Description
Macro name	LIN_BOARD_CPU_FREQ_HZ
Validity restrictions	None
Description	The value of this macro specifies the master clock frequency (in Hz) of the STM8A device. It is necessary to set this macro to the value which corresponds to the real master clock frequency of the STM8A in order to get the correct baudrate on the LIN bus as well as the correct driver internal timing in case the hardware timer is used to provide time basis for the LIN driver. If a hardware timer is configured to be used as a time base for the driver, this macro can be set to value either 8000000 or 16000000. For more information, refer to the section *Choosing and configuring hardware timer*.

Table 39. LIN_USE_HARDWARE_TIMER

Macro	Description
Macro name	LIN_USE_HARDWARE_TIMER
Validity restrictions	None
Description	This macro enables the user to choose either the software or the hardware timer to be used as the time basis for the LIN driver. #define LIN_USE_HARDWARE_TIMER In case the macro is defined, the LIN driver configures HW timer (specified by the macro LIN_TIMER, see *Table 40*) during driver initialization and uses it as the driver time basis with a period given by the macro LIN_TIME_BASE_IN_MS (described in *Table 3*). #undef LIN_USE_HARDWARE_TIMER In case the macro is not defined, the LIN driver time basis must be provided by the application (software timer) – the application must periodically call the “l_timer_tick” driver function with a period given by the macro LIN_TIME_BASE_IN_MS (described in *Table 3*).

Table 40. LIN_TIMER

Macro	Description
Macro name	LIN_TIMER
Validity restrictions	Valid only if the macro LIN_USE_HARDWARE_TIMER (see *Table 39*) is defined.
Description	The value of this macro specifies the hardware timer number used for timing the LIN driver. Only the TIM4 hardware timer of the STM8A device is used as a time basis for the LIN driver and therefore the value of this macro has no influence on the timer used.

Table 41. LIN_SLAVE_LINSCI_AUTOSYNC

Macro	Description
Macro name	LIN_SLAVE_LINSCI_AUTOSYNC
Validity restrictions	Valid only for a slave node and available for the SCI1 LIN interface only.
Description	This macro enables/disables the automatic LIN slave node baudrate synchronization on the SYNC field in LIN frame headers (generated by the master node). #define LIN_SLAVE_LINSCI_AUTOSYNC In case the macro is defined, the automatic LIN slave node baudrate synchronization is enabled. This allows a usage of a non-precise clock source (e.g. RC-based oscillator) to clock the slave node and still be able to meet the LIN bus communication timing requirements. #undef LIN_SLAVE_LINSCI_AUTOSYNC In case the macro is not defined, the automatic LIN slave node baudrate synchronization is disabled. The slave node LIN bus baudrate clock is then given by the source clock frequency for the LIN peripheral and a fixed division ratio set by the LIN driver to generate the baudrate clock.

Table 42. LIN_ZERO_PAGE_SIZE

Macro	Description
Macro name	LIN_ZERO_PAGE_SIZE
Validity restrictions	None
Description	The value of this macro specifies how many bytes located in zero page memory area of the STM8A device may be used for certain LIN driver internal variables in order to improve the speed and particularly the code size of the LIN driver. For more details on this macro, please refer to the document [*6*].

Table 43. LIN_xxx_IN_ZERO_PAGE

Macro	Description
Macro name	LIN_xxx_IN_ZERO_PAGE
Validity restrictions	None
Description	The name LIN_xxx_IN_ZERO_PAGE used here represents any of the following macro names: LIN_TX_FLAGS_IN_ZERO_PAGE LIN_RX_FLAGS_IN_ZERO_PAGE LIN_CHANGED_FLAGS_IN_ZERO_PAGE LIN_FRAME_BUFFER_IN_ZERO_PAGE LIN_FRAME_IDS_IN_ZERO_PAGE These macros control the possible location of some LIN driver internal data in the zero page memory area of the STM8A device in order to improve the speed and particularly the code size of the LIN driver. For more details on macros and their meanings, please refer to the documentation [*6*]. #define LIN_xxx_IN_ZERO_PAGE In case the macro is defined, the corresponding set of LIN driver internal data is located in the zero page memory area. #undef LIN_xxx_IN_ZERO_PAGE In case the macro is not defined, no explicit location of the corresponding set of LIN driver internal data in the zero page memory area is performed.

4 LIN driver integration into a user application

This chapter provides some guidelines to integrate the LIN driver into a user application.

4.1 LIN driver integration process

To integrate the LIN driver into a user application, follow the steps below:

1. Install the LIN package
2. Provide a description for LIN cluster(s) and LIN interface(s) used by the LIN node
3. Provide files with LIN driver configuration
4. Provide the application source codes together with a linker file
5. Setup the application development environment

After going through with all these steps, the application can be build to obtain the application executable binary. The steps listed above are described in the following paragraphs.

4.1.1 LIN package installation

The LIN package (a directory structure containing all the LIN driver related software stuff) can be installed into any location in the file system directory tree. The location where the LIN package must be installed is specified by the user during the package installation procedure. The name of the LIN package root directory is also specified by the user during the installation. After it is installed, the created LIN package content is as described in the *Section 3.1*.

4.1.2 Creating files for the lingen tool

The “Lingen” tool (a utility which is an integral part of the LIN package) is used to generate some LIN driver source codes with content according to the information stored in two description files taken as an input of the “Lingen” tool. These files are:

LIN Definition File (LDF file)	Contains description for the LIN cluster(s) in which the user application (generated LIN node application) operates. The content of this file must conform to the LIN standard specification. Every LIN cluster is described by one separate LDF file.
Lingen Control File (LCF file)	A special control file for the “Lingen” tool. This file specifies which LIN interfaces (LIN communication capable peripherals) of the STM8A device are to be used in the user application and which LIN Definition File (LDF file) is associated with a particular LIN interface.

Creating lingen control file (LCF file)

The LCF file contains a section which describes the interface(s) of the STM8A device used for the connection to the LIN cluster(s).

The LCF file name is optional as well as the file extension. However it is recommended to use the ".lgn" extension (used also for demo applications LCF files).

The following example shows the interface section of the Lingen control file when the node operates in one LIN cluster (the node is either a slave node connected to a LIN cluster or a master node driving one LIN cluster):

```
//
//  Lingen control file defining one interface
//
Interfaces
{
    SCI1: "lin_config/lin_basic_demo.ldf";
}
```

The example above illustrates the case of a LIN node connected to the LIN cluster via the SCI1 LIN interface of the STM8A. The LIN cluster configuration is described in the "lin_basic_demo.ldf" LDF file located under the path "lin_config/" (this is the relative path, the absolute path can be also used).

Note: Two LIN interfaces are available on an STM8A microcontroller:

SCI1 - corresponds to the LINUART peripheral of the STM8A device

SCI2 - corresponds to the USART peripheral of the STM8A device.

If the LIN node uses more than one LIN interface of the STM8A device (a master node driving more than one LIN cluster), a "tag identifier" should be specified for each LIN interface mentioned in the LCF file. The situation is illustrated in the following example which shows the interface section of a LCF control file for a master node connected to two LIN clusters:

```
//
//  Lingen control file defining two interfaces
//
Interfaces
{
    SCI1: "lin_config/lin_sci1.ldf", "IFC1";
    SCI2: "lin_config/lin_sci2.ldf", "IFC2";
}
```

In this example, the tag identifier "IFC1" is used for LIN interface "SCI1" (the LIN interface "SCI1" is connected to the LIN cluster described in the LDF file "lin_sci1.ldf") and the tag identifier "IFC2" is used for LIN interface "SCI2" (the LIN interface "SCI2" is connected to the LIN cluster described in the LDF file "lin_sci2.ldf").

Tag identifiers are essential for resolving the naming conflicts if two LDF files have common signal names. The tag identifier is concatenated with all frame and signal names when the "Lingen" tool processes LCF and LDF files. For example (considering the situation given by the LCF file content of the example mentioned above), if a signal with a name "slave1_sig1"

is defined in the LDF file “lin_sci1.ldf”, in a user application code this signal is referenced as “LIN_SIGNAL_IFC1_slave1_sig1”. In case that the same signal name is defined in the LDF file “lin_sci2.ldf”, there is not conflict in signal naming in the user application code as the this second signal is referenced as “LIN_SIGNAL_IFC2_slave1_sig1”.

The LCF file can contain also an optional keyword “LIN_use_default_frame_ids” which is intended to be used for slave nodes only. If present in a LCF file, the default frame identifiers given in the LDF file is used for all slave frames. It means that for that slave the initial configuration of IDs of used LIN frames, done by the master node, is not performed and for all the communication with the slave node only the default IDs defined in the LDF file is used. The following example shows the syntax of the keyword section if included in the LCF file:

```
//
// Specify that slave nodes starts with default frame IDs
//
LIN_use_default_frame_ids;
```

Creating LIN definition file (LDF file)

For every LIN cluster, to which the LIN node is connected, one separate LDF file must be provided. LDF file contains description of one LIN cluster (e.g. signals and frames used for communication be between nodes, type of frames used, schedule tables, cluster timing parameters).

The file name of a LDF file is optional. The file name extension used for LDF files is “.ldf”.

The syntax and semantic of a LDF file must conform to the LIN standard. The user should refer to the LIN standard specification (document [1], [2] or [3]) and create LDF files for his/her user application in accordance with the specification.

Some examples of LDF file content are included in LIN standard specification documents, some LDF file content examples can be found in the documents [4] and [5]. The demo application directory of the LIN package contains LDF files since every demo application includes a LDF file and these files can also be taken as LDF file examples.

Note: There are significant differences between LIN 2.0 and LIN 2.1 LDF file content syntax and semantic.

4.1.3 Creating LIN driver configuration files

The LIN driver configuration for a user application is defined by a few configuration files. The LIN package includes templates for these configuration files in the directory “<LIN_Package_Root>\src\config”. The user should make a copy of these files in the user application source file area and then modify the content of these new files according to the desired LIN driver configuration as well as according to the user application structure.

How to configure the driver and the explanation for the content of the LIN driver configuration files is described in the *Section 3.2*.

For some more specific information on the LIN driver configuration topic, refer to the documentation [4], [5] and [6].

4.1.4 Setting-up LIN driver time base

To provide correct timing for the LIN bus communication protocol, the LIN bus driver must be connected to a “time basis”. The time basis for the driver is delivered through a timer. The timer can be either a “hardware timer” or a “software timer”.

The term “hardware timer” means that all the STM8A timer peripherals are dedicated for the LIN driver and are directly used to generate LIN driver time base clocks. The term “software timer” means that the user application (or operating system) provides the LIN driver time base clocks by calling a specific LIN driver function with a period corresponding to the desired time base clock frequency.

Choosing and configuring hardware timer

The usage of hardware timer is configured through the macro LIN_USE_HARDWARE_TIMER (see *Table 39*) in the driver configuration file “lin_def_stm8.h” file. The macro must be defined, so the macro definition code line, as shown in the following code sample, must be present in the “lin_def_stm8.h” file:

```
#define LIN_USE_HARDWARE_TIMER
```

The user must specify which STM8A timer peripheral is used as the timer. This is done in the LIN driver configuration file “lin_def_stm8.h”. The following “lin_def_stm8.h” file line sample shows the macro definition which selects the hardware timer 1 as the hardware timer:

```
#define LIN_TIMER  1
```

Note: In current version of STM8A LIN driver, only the 8-bit timer TIM4 of the STM8A device can be used. Therefore the value of the macro LIN_TIMER has no influence on the timer used (see Table 40).

For a proper LIN driver timing operation (correct baudrate used on the STM8A LIN interface), it is essential to set the value of the macro LIN_BOARD_CPU_FREQ_HZ (see *Table 38*) to a value which corresponds to the master clock frequency of the STM8A device (e.g. external crystal oscillator frequency). This is done in the LIN driver configuration file “lin_def_stm8.h”. The following example shows the situation when the 8MHz oscillator frequency is used on the STM8A device:

```
#define LIN_BOARD_CPU_FREQ_HZ   8000000
```

Note: In current version of STM8A LIN driver, the numeric values 8000000 and 16000000 are the only values of the LIN_BOARD_CPU_FREQ_HZ macro which are accepted by the driver if the hardware timer is used. No other values can be used without certain modification in the LIN driver code.

The period between two clock ticks of the hardware timer, actually the time base period, must be specified in the configuration file “lin_def.h”. This time base period specifies the frequency at which the driver's timer routine must be called. The following code example demonstrates how to set the time base period to 2 milliseconds:

```
#define LIN_TIME_BASE_IN_MS  2
```

Note: In current version of STM8A LIN driver, the numeric values 1 and 2 are the only values of the LIN_TIME_BASE_IN_MS macro which are accepted by the driver if the hardware timer is used. No other values can be used without certain modification in the LIN driver code.

The LIN driver configures the timer peripheral (used as the hardware timer) according to the values of macros mentioned above. However the LIN driver does not configure and enable

the interrupts for the timer peripheral in the STM8A interrupt controller and does not enable the hardware timer peripheral. The user application must perform these tasks.

Therefore, the user must provide (somewhere in the user application code):

- a timer related code which configures and enables interrupts for the hardware timer
- a code which enables the timer peripheral before the driver is being used

The function “lin_osHWInit” defined in the demonstration application source codes can be used as an example. This function performs STM8A hardware initialization when the demonstration application starts. All the necessary hardware initialization steps, related to the hardware timer and performed by the user application, are included in this function.

A reference to the interrupt service routine (ISR) servicing interrupts from the hardware timer peripheral must be added into the STM8A “interrupt vector table” so that this ISR function is entered when an interrupt from the timer peripheral occurs. The LIN driver API function “l_timerISR” must be called in the ISR. The following code example shows the definition of such an ISR:

```
@interrupt void l_timer4ISR(void)
{
#ifdef LIN_USE_HARDWARE_TIMER
  l_timerISR();
#endif
}
```

Choosing and configuring a software timer

If a software timer is used, the application must periodically call the “l_timer_tick” driver API function and, to generate the time basis clocks. For example, this can be done by calling the function from the interrupt service routine of any hardware timer used by the user application or from a timing routine of the operating system (if used).

The usage of the software timer is configured through the macro LIN_USE_HARDWARE_TIMER (see *Table 39*) in the driver configuration file “lin_def_stm8.h” file. The macro must not be defined, so the macro definition code line, as shown in the following code sample, must be used:

```
#undef LIN_USE_HARDWARE_TIMER
```

For a proper LIN driver timing operation (correct baudrate used on the STM8A LIN interface), it is essential to set the value of the macro LIN_BOARD_CPU_FREQ_HZ (see *Table 38*) to the value which corresponds to the master clock of the STM8A device (e.g. external crystal oscillator frequency). This is done in the LIN driver configuration file “lin_def_stm8.h”. The following example shows the situation when the 8MHz oscillator frequency is used on the STM8A device:

```
#define LIN_BOARD_CPU_FREQ_HZ   8000000
```

The period between two consecutive calls to the LIN driver function “l_timer_tick”, actually the time base period, must be specified in the in the configuration file “lin_def.h”. The following code example demonstrates how to set the time base period to 2 milliseconds

```
#define LIN_TIME_BASE_IN_MS  2
```

4.1.5 Setting up interrupts, GPIOs, LIN interface enabling

The LIN driver does not enable the STM8A LIN interface peripheral. It does not configure the interrupts for LIN interface peripheral and does not configure GPIOs (General Purpose I/Os) used by the LIN interface peripheral. The user application perform these tasks.

Therefore, the user must provide (somewhere in the user application code):

- a code which configures GPIOs for the LIN interface peripheral used by the LIN driver
- a code which enables the LIN interface peripheral before the LIN driver is being used
- a code which enables the interrupts from a LIN interface peripheral
- an interrupt service routine servicing the interrupts from a LIN interface peripheral
- a code which enables processing of hardware interrupts in the STM8A core

The function “lin_osHWInit” defined in the demonstration application source codes can be used as an example on how to configure the GPIOs and enable the LIN interface peripheral. his function performs the STM8A hardware initialization when at the beginning of the demonstration application starts.

A reference to the interrupt service routine (ISR) servicing the interrupts from a LIN interface peripheral must be added into the STM8A “interrupt vector table” so that this ISR function is entered when any interrupt from the LIN interface peripheral occurs. In the ISR, the “l_ifc_rx_SCIx” LIN driver API function must be called (the “x” in the function name should be replaced with corresponding LIN interface number). The following code example shows the definition of such an ISR for the SCI1 interface:

```
@interrupt void l_serial1ISR(void)
{
#ifdef SCI1_USED
   l_ifc_rx_SCI1();
#endif
}
```

4.1.6 Lingen process, compiling and linking

The “Lingen” tool processing invocation and compilation for the LIN driver source codes and LIN configuration files are controlled by the “make” utility (included in the demo application directories). The “make’ utility processes the “Make_LIN” makefile located in the “<LIN_Package_Root>\make\stm8” directory.

The compilation of the user application part of source codes and the application linking process is then controlled by the “make” utility and a “top-level” makefile which is created by the user.

The top-level makefile, provided by the user, must include the settings for environment variables used by the “Make_LIN” makefile and invoke the “make” process of the “Make_LIN” makefile. The *Table 44* specifies environment variables used by the “Make_LIN” makefile.

The following top-level makefile line sample shows a way how to include the “Make_LIN” makefile into the top-level makefile (the term “<LIN_Package_Root>” specifies the LIN package root directory):

```
include <LIN_Package_Root>/make/stm8/Make_LIN
```

The invocation of the “make” process of the “Make_LIN” makefile from the top-level makefile results to the situation that the “make” process of the user application performs also the “Lingen” processing as well as the compilation of the LIN driver.

The generation of the LIN driver library can then be included as follows:

```
make $(LIN_OBJ_PATH)/lin.lib
```

or by including the following line into the target build instruction:

```
$(LIN_OBJ_PATH/lin.lib
```

Table 44. Top-level makefile variables description

Variable	Variable value description
LIN_NODE_IDENTITY	This must be set to MASTER_NODE for a master node driver or to SLAVE_NODE for a slave node driver.
LIN_CC	Compiler command.
LIN_CC_OPT	Compiler options.
LIN_CC_INC	Include directories for the compilation process.
LIN_MAKE_PATH	Path to the “Make_LIN” file.
LIN_OBJ_PATH	Path in which to generate object files.
LIN_LIB	Lib generation tool.
LIN_TMP_FILE	Name for temporary file.
LIN_SRC_PATH	Top-level directory of the LIN driver.
LIN_CFG_PATH	Directory containing the configuration information for the cluster and the driver - Lingen control file (LCF file), LIN definition file(s) (LDF files) and files “lin_def.h, “lin_def.c” and “lin_def_stm8.h” must be located in this directory.
LIN_GEN_PATH	Directory in which generated files are written to. This is used for the “–o” option for the “Lingen” tool in the file “Make_LIN”.
LIN_LINGEN_BIN	Command used to invoke the “Lingen” tool.
LIN_NODE	Name of the node as defined in the LDF file. If multiple interfaces are defined for a master node then the name given in the associated LDF files should be the same throughout.

Table 44. Top-level makefile variables description (continued)

Variable	Variable value description	
LIN_LINGEN_OPTS	Command line options to be used by the “Lingen” tool. Details for the options are as follows:	
	-c configuration	Specifies which of the possible configurations given in the LDF is to be used for the build.
	-o outputDirectory	Specifies the destination for the configuration files that are to be generated.
	-r receiveChecksum	Selects the checksum model to be used for receiving frames. Possible values are: ldf – the “Lingen” tool determines the checksum model from the information given in the LDF file. This is the default settings. both – the driver accepts either model for all frames.
	-s sendChecksum	Selects the checksum model to be used for sending frames. Possible values are: ldf – the “Lingen” tool determines the checksum model from the information given in the LDF file. This is the default settings. classic – the driver sends all frames using the classic checksum.
	-v	Verbose mode, details from the “Lingen” process in text form is sent to the shell.

In addition, the optional makefile variable “LIN_LDF_FILES” may be set by the user. This can be used to list the LDF filename(s) to be included in the dependency checks during the “make” process.

The LIN package contains two top-level makefiles in the demo application directories. These makefiles can be used by the user as a starting point to write his/her own user application makefiles.

5 LIN versions compatibility

This section explains how to integrate a LIN node implementing a given LIN protocol version into an existing LIN cluster using a different version.

5.1 LIN 2.0 node compatibility with LIN 1.3 nodes

LIN 2.0 is a superset of LIN 1.3. All the changes between LIN 1.3 and LIN 2.0 standard are listed in details in the document [2].

The LIN 2.0 physical layer is backwards compatible with the LIN 1.3 physical layer. This means that a LIN 2.0 node physical layer can operate in a LIN 1.3 cluster, but not vice versa.

A LIN 2.0 master node can handle clusters consisting of both LIN 1.3 slaves and/or LIN 2.0 slaves. The master does not request the new LIN 2.0 features from LIN 1.3 slaves:

- Enhanced checksum
- Reconfiguration
- Diagnostics
- Sporadic frames
- Automatic baudrate detection
- “Response_error” status monitoring

A LIN 2.0 slave nodes can not operate with a LIN 1.3 master node (e.g. the LIN 1.3 master does not support the enhanced checksum).

5.2 LIN 2.1 node compatibility with LIN 1.3 nodes

LIN 2.1 is a superset of LIN 1.3.

The LIN 2.1 physical layer is backwards compatible with the LIN 1.3 physical layer. This means that a LIN 2.1 node physical layer can operate in a LIN 1.3 cluster, but not vice versa.

A LIN 2.1 master node can handle clusters consisting of both LIN 1.3 slaves and/or LIN 2.1 slaves. The master does not request the new LIN 2.1 features from LIN 1.3 slaves:

- Enhanced checksum
- Reconfiguration
- Diagnostics
- Sporadic frames
- Automatic baudrate detection
- “Response_error” status monitoring

LIN 2.1 slave nodes can not operate with a LIN 1.3 master node (e.g. the LIN 1.3 master does not support the enhanced checksum).

5.3 LIN 2.1 node compatibility with LIN 2.0 nodes

All the changes between LIN 2.0 and LIN 2.1 standards are listed in details in the document [3]. The LIN 2.1 and the LIN 2.0 physical layers are compatible.

A LIN 2.1 master node may handle a LIN 2.0 slave node if the master node also contains all functionality of a LIN 2.0 master node (e.g. obsolete functions like “Assign Frame Id” configuration service, message identifiers for slave node frames). The master does not request the new LIN 2.1 features from LIN 2.0 slaves:

- “Assign frame ID range” configuration service
- “Save configuration” service
- Enhancements of status reporting to application
- Enhancements of transport layer
- Event-triggered frame collision handling modified
- New event-triggered frame collision handling
- The IDs 2 to 31 of the service “Read by Identifier” are reserved

A LIN 2.0 slave node shall not use NAD 0x7E since it is reserved as functional address for diagnostics in LIN 2.1. The LIN 2.1 slave node considers NAD 0x7E as a functional NAD and a LIN 2.0 slave node as a NAD.

A LIN 2.1 slave node can be used in a cluster with a LIN 2.0 master node if the LIN 2.1 slave node is pre-configured, i.e. the LIN 2.1 slave node has a valid configuration after reset.

One way how to ensure that the LIN 2.1 slave node is pre-configured is to create the LIN Description File (LDF file) with correct LIN cluster description. Following example shows settings for few important items in the content of LDF file which describes a LIN 2.0 cluster in which one newly created LIN 2.1 slave should operate:

```
LIN_description_file;
LIN_protocol_version = "2.0";
LIN_language_version = "2.1";
...
...
Nodes
{
  ...
  Slaves: slave1;
}
...
Node_attributes
{
  slave1
  {
    LIN_protocol = 2.1;
    ...
  }
 ...
}
...
```

If the LIN cluster description in the LDF file is as shown above, the “Lingen” tool checks that all the LIN 2.1 slave node properties (listed in the LDF file) are compatible with the LIN 2.0 standard (fault state signals must not be present, initial NAD must not be present, etc.) and ensures that the LIN 2.1 slave node has a valid configuration after reset (is pre-configured).

Another way to ensure that the LIN2.1 slave node is pre-configured, is to use the "LIN_use_default_frame_ids" keyword in the Lingen Control File (LCF file) as described in the *Creating lingen control file (LCF file)*. In this case, all frame IDs in the LIN cluster are initialized, included those of the other nodes which have been generated using the same LCF file. If this a side effect is not acceptable, this solution should not be implemented in the application.

6 Reference documents

1. LIN Specification Package, Revision 1.3, December 12, 2002
2. LIN Specification Package, Revision 2.0, September 23, 2003
3. LIN Specification Package, Revision 2.1, November 24, 2006
4. LIN 2.1 Driver Suite, Slave Node User's Guide, Version 2.0, March 27, 2008 (included in the STM8A LIN 2.1 package)
5. LIN 2.1 Driver Suite, Master Node User's Guide, Version 2.0, March 27, 2008 (included in the STM8A LIN 2.1 package)
6. LIN 2.1 Driver Suite, LIN driver for STM8, Architecture Notes, Version 2.0, May 5, 2009 (included in the STM8A LIN 2.1 package)

7 Revision history

Table 45. Document revision history

Date	Revision	Changes
13-Nov-2009	1	Initial release.
27-May-2014	2	Added reference to STSW-STM8A-LIN.