LIN_Protocol_ver 2.2A

Title: **SW Component LIN_Protocol**

Maturity:	Draft
Authors:	David Antonio Diaz Ramirez
	Francisco Javier Quirarte Pelayo
VERSION:	Public
Security Classification:	None

History				
Issue status (Index)	**Maturity/Date** (draft/invalid/valid) (dd-mmm-yyyy)	**Author** Department	**Check/Release** Department	**Description**
1.0	Draft 26-nov-15	Francisco Quirarte	Francisco Quirarte y David Díaz	Creation of the document.

History				
Issue status (Index)	**Maturity/Date** (draft/invalid/valid) (dd-mmm-yyyy)	**Author** Department	**Check/Release** Department	**Description**
2.0	Draft 27-nov-15	David Díaz	Francisco Quirarte	Added: Purpose, Definitions and abbreviations.

History				
Issue status (Index)	**Maturity/Date** (draft/invalid/valid) (dd-mmm-yyyy)	**Author** Department	**Check/Release** Department	**Description**
3.0	Draft 28-Nov-15	Francisco Quirarte y David Díaz	Francisco Quirarte	Added: Realization constraints and targets.

History				
Issue status (Index)	**Maturity/Date** (draft/invalid/valid) (dd-mmm-yyyy)	**Author** Department	**Check/Release** Department	**Description**
4.0	Draft 29- Nov -15	Francisco Quirarte y David Díaz	Oswaldo Garcia	Added: SW Conceptual design.

History				
Issue status (Index)	**Maturity/Date** (draft/invalid/valid) (dd-mmm-yyyy)	**Author** Department	**Check/Release** Department	**Description**
6.0	Draft 30-nov-15	Francisco Quirarte y David Díaz	Luis Alvarez	Review of SW Component internal breakdown and added general corrections. General corrections at traceability matrix

History				
Issue status (Index)	**Maturity/Date** (draft/invalid/valid) (dd-mmm-yyyy)	**Author** Department	**Check/Release** Department	**Description**
7.0	Release 30-Nov-15	Francisco Quirarte y David Díaz	Oswaldo Garcia e Isamar Gálvez	Review of SW Conceptual design and added general corrections. Review of the test plan.

Table of Contents

1   Introduction

This application note shows how to implement a LIN (Local Interconnect Network) slave task in a 32-bit RISC microcontroller without the need for any external components. The LIN protocol is a serial communications protocol which efficiently supports control of mechatronic nodes in a distributed network. This makes the protocol ideal for use in automotive applications. A LIN network consist of a single Master and a set of Slave nodes. This application note show how to implement the protocol for the physical layer and the data link layer according to the two lowest levels of the ISO/OSI reference model. This provides the foundation for message transmissions between nodes in the network. The physical layer of the ISO/OSI model only provides a raw bit-stream between two nodes in the network. The data link layer makes the physical layer reliable by adding error detection and control. It also adds the means to activate, maintain, and deactivate the link. The higher levels of the ISO/OSI reference model is beyond the scope of the LIN protocol and thus not explained here. For now, this is up to the user to specify and implement. The LIN protocol differs from the CAN protocol in the sense that it is below the performance and scope of the CAN. Its strongpoint is that it is provides a simple, low-cost solution for applications not requiring the power of the CAN protocol.

2   Purpose

Bus Local Interconnect Network, LIN usually abbreviated Bus is a system used in current data transmission networks Automotive System. SPECIFICATION meets the LIN consortium, in its first version 1.1, released in 1999. The specification has been developed from then to version 2.0 para solve the current needs of a red. The LIN bus is a little slow system that USING As a cheap bus para- red sub UN can integrate smart devices or actuators in today's cars.

Is a concept for low cost automotive networks, which complements the existing portfolio of automotive multiplex networks. LIN will be the enabling factor for the implementation of a hierarchical vehicle network in order to again further quality enhancement and cost reduction of vehicles. The standardization will reduce the manifold of existing low-end multiplex solutions and will cut the cost of development, production, service and logistics in vehicle electronics.

3 Comparison chart protocols

	SPI	I2C	RS232	RS485[1]	LIN	CAN
TIPO DE TRANSMISIÓN	Sincrono[2]*	Sincrono	Síncrona, asíncrona	Síncrona, asíncrona[3]	Asíncrono*	Asincrono[4]
SOPORTE MULTIMAESTRO	NO	SI	NO	SI	NO	SI[5]
PINES UTILIZADOS	SLCK, MOSI,MISO,SS	SDA,SCL	TxD,RxD,SG	TxD,RxD,SG	LIN BUS	CANH, CANL,
COMUNICACIÓN	Half dúplex, Full dúplex *	Halfdúplex, full dúplex*	simplex, half duplex o full dúplex[6]	HalfDuplex	Simplex*	Halfduplex
VELOCIDAD Y DISTANCIA DE TRANSMISION	320kbps-5Mbps[7],2 m	100kbps - 400kbps, y alta velocidad 3.4 Mbps	20 Kbps, hasta 15m[8]	10Mbps a 10 m, y 100kbps a 1200m[9]	20 kbps [10], 40 m y aumenta con un repetidor[11]	125 Kbps a 500m -1 Mbps a 40 m[12]
CONSUMO DE ENERGIA	No utiliza Pull-ups y ahorra energía, fuente +5V	Utiliza Pull-Ups, fuente +5V	+12 y -12 Volts	fuente +5V , 1.5v entre las salidas diferenciales.	Utiliza Pull-Ups	cable L (Low) = 0V y 2.25V, cable H= 2.75V. y 5V
DIRECCIONAMIENTO	utiliza líneas especificas para cada dispositivo	Cada dispositivo tiene una dirección única por software, 128 direcciones	Es punto a punto		basado en mensajes, no en direcciones	basado en mensajes, no en direcciones
CANTIDAD DE DISPOSITIVOS SOBRE EL MISMO BUS	Depende de los pines disponibles del PIC para SS	Depende de la capacitancia de las líneas 400 pf	Uno a uno	32 (Bus de 2 hilos), 64 (4 hilos bus)	16 unidades esclavas[13]	127
NOTAS			Trabaja con lógica negativa			Necesita de un transeiver x cada dispositivo
COMPAÑIA	Motorola	Philips Semiconductor	Electronic Industries Alliance	Electronic Industries Alliance		Robert Bosch GmbH

4 LIN Protocol Concepts

Single Master Multiple Slaves
The LIN protocol does not use bus arbitration. A single Master is responsible for initializing all message transfers. All slaves can respond to the Master or any other node in the network but only after being addressed and given permission by the Master.

Variable Length of Data Frames
Included in the identifier field are two bits indicating the length of the message field (see Table 1). This adds flexibility and reduces overhead bytes when only a limited amount of data is required.

Multi-cast Reception
When a message frame is transmitted from the Master or a Slave task, all connected nodes in the network can read the message. Depending on the identifier byte, the receiving nodes decides if any action is to be initiated or not. For example, a single "CLOSE ALL" command from the Master could be accepted from all nodes and could in the case of a car security system close all windows and doors.
Time Synchronization without Need for Quartz or Ceramic Resonators in the Slave Nodes
After the Synch Break, a Synch Field is transmitted from the Master, this field makes it possible for all the Slaves to synchronize to the master clock. Such synch fields are placed at the beginning of every message frame. The accuracy of the receiving Slaves need only be good enough to keep synchronization through the entire message frame. This feature allows the slave to run on an internal RC Oscillator thus reducing the overall system cost.

Data Checksum Security and Error Detection
The data in the message frame uses an inverted modulo256-checksum with the carry of the MSB added to the LSB for error detection. In addition, the identifier byte uses a XOR algorithm for error detection.

Detection of Defect Nodes in the Network
The Master task is responsible for initiating the transmission of message frames and thus has the responsibility of requesting information and checking that all nodes are alive and working correctly.
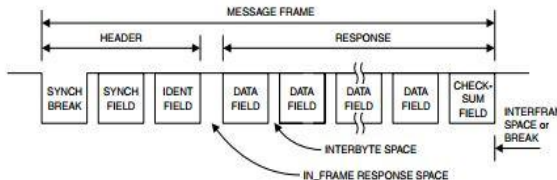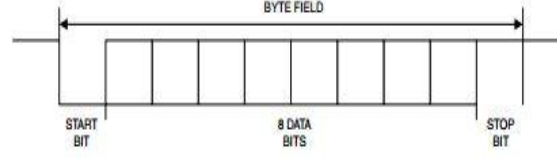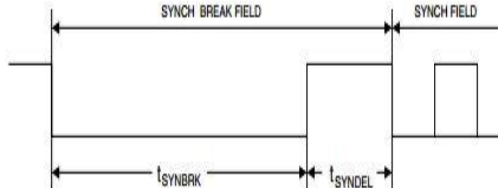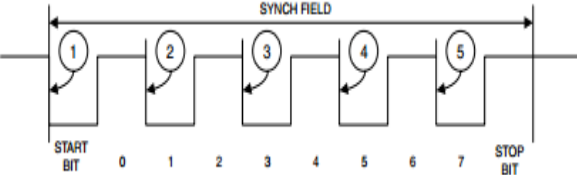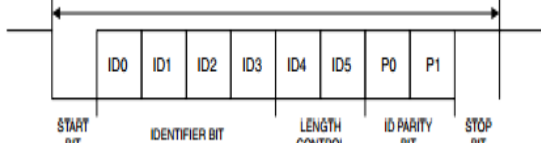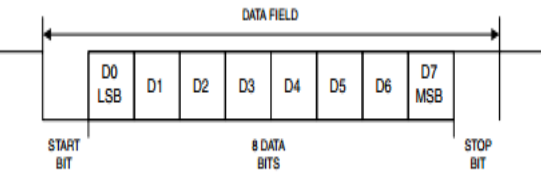
Minimum Cost Solution
Due to the simplicity of the protocol, a slave task complying to the LIN standard can be built using a minimum of external components and does not put heavy constraints on the accuracy of the Oscillator in the Slave nodes.
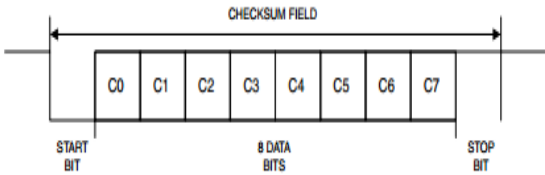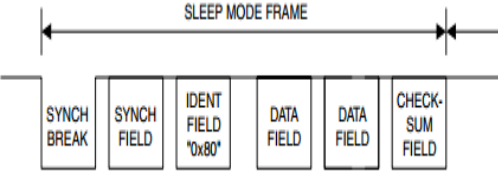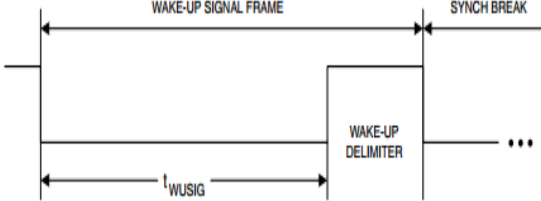
5 LIN PROTOCOL CONCEPTS

In the following chart will describe the concepts LIN protocol:

Signal Transmission	The LIN bus consists of a single channel that carries both data and synchronization information. The physical medium is a single wire connected to VCC via a pull-up resistor (see Figure 1). The idle state on the bus is high or "recessive" and the active state is low or "dominant". In automotive applications VCC will typically be the positive battery node. The LIN protocol does not define an acknowledgment procedure for the Slave tasks. The Master task uses its own slave task to verify that the sent message frame is identical with the one received by this slave task. If any discrepancy is detected the message frame can be retransmitted. The data rate for transmitted data is limited to 20 Kbits/s due to EMI (Electro Magnetic Interference) requirements for a single wire transmission medium.
Message Frames LIN Message Frame 	All information transmitted on the LIN bus is formatted as Message Frames. As shown in Figure 2, a Message Frame consists of the following fields: • Synch Break • Synch Field • Identifier Field • Data Field • Checksum Field
Byte Fields LIN Byte Field 	The Byte Field format, shown in Figure 3, is identical to the commonly used UART serial data format with 8N1 coding. This means that each Byte Field contains eight data bits, no parity bits, and one stop-bit. Every Byte Field has the length of 10 bit-times (Tbit). As shown in the figure, the start bit marks the beginning of the Byte Field and is "dominant" while the stop-bit is "recessive". The eight data bits can be either "dominant" or "recessive".

Synch Break	The Synchronization Break marks the beginning of a Message Frame. This field is always sent by the Master task, and provide a means for the Slave task to prepare for the synchronization field. The Synch Break consists of two different parts: a dominant (low) level that should be minimum 13 bit-times (Tbit) which is followed by a recessive (high) period that should be in the range one to four Tbit. The second field is necessary to detect the dominant (low) level of the start bit of the following Synch Field.
Synch Field	The Synch Field contains the signaling required for the slave to synchronize with the master clock. The Synch Field is a Byte Field containing the data "0x55" giving the waveform shown in Figure 4.
Identifier Field	The Identifier Field contains information about contents and length of a message. As shown in Figure 6, this field is divided into three sections: identifier bits (four bits), length control bits (two bits) and parity bits (two bits). This divides the set of 64 identifiers into four subsets of 16 identifiers each.
Data Field	The data frame contains from two to eight Data Fields containing eight bits of data each. Transmission is done with LSB first. The data fields are written by the responding Slave task. Since there is no bus arbitration, only one slave task should be allowed to respond to each identifier. All other Slave tasks are limited to reading the response and act accordingly.

Checksum Field	The last field in the Message Frame is the Checksum Field. This byte contains the inverted modulo-256 sum of all data bytes (data frame not including the identifier). This sum is calculated by doing an "ADD with carry" on all data bytes and then inverting the answer. The properties of the inverted modulo-256 sum are such that if this number is added to the sum of all data bytes the result will be "0xFF".
Sleep Mode Frame	The frame structure for the Sleep Mode Frame is identical to an ordinary Message Frame and is distinguished by the identifier byte "0x80". The contents of the data fields are not specified and could be used to distribute system parameters for the sleep mode.
Wake-up Signal	The sleep mode can be terminated by any connected slave task by sending a wake-up signal. The wake-up signal is only allowed when the bus is in sleep mode and a node internal request for wake-up is pending. The wake-up signal is the character "0x80".

6   Definitions and abbreviations

Definitions

LIN 2.2A	Serial network protocol used for communication between components in vehicles
Master (node)	Node chosen to control all communications. Bus arbiter.
Slave (node)	Node that responds to the master in three possible ways: Receive data, transmit data or ignore data.

Abbreviations

LIN	Local Interconnect Network
RX	Reception
TX	Transmission
MSTR	Master
SLV	Slave
MSG	Message
HAL	Hardware Abstraction Layer
LIN	Local Interconnect Network
CAN	Controller Area Network
ECU	Electronic Control Unit
API	Application User Interface
IDE	Integrated Development Enviroment
UART	Universal Asynchronous Receiver Transmitter
ASC	Async/Synchronous Serial Interface
ID	Identifier
EMI	Electro Magnetic Isolation
t_slv_stat	Signal to display slave node status.
t_cmdType	Signal to control LED states and slave node state.

t_LEDstat	Signal to display LED status.
t_boolean	Signal response that indicates slave node status.
array	Signal that contains the team member's initials for the Msg ID.
scalar	Signal that contains the team assigned number for the Msg ID

References

N°	Document name
1	*Traceability Matrix Template.xls*
2	MPC5604B/C Microcontroller Reference Manual.pdf
3	Test_template.xls
4	http://www.freescale.com/products/power-architecture-processors/mpc5xxx-5xxx-32-bit-mcus/mpc56xx-mcus/mpc5606b-startertrak-development-kit:TRK-MPC5606B
5	http://www.freescale.com/products/power-architecture-processors/mpc5xxx-5xxx-32-bit-mcus/mpc56xx-mcus/ultra-reliable-mpc56xb-mcu-for-automotive-industrial-general-purpose:MPC560xB#pspFeatures
6	SW_C_Code_Review_Template.docx
7	*Window-lifter-requirements.docx*
8	*LIN Protocol_Conti.ppt*
9	*LIN_Network_Database*
10	*LIN-Spec_2-2A.doc*
11	*LIN-Spec_2-2A.pdf*
12	*http://www.atmel.com/images/doc1637.pdf*

7   LIN Software Driver Services

The LIN driver provides several API-functions for LIN bus handling. The main services of the LIN software driver are:
• Message transmission
 • Message reception
 • Message filtering
• Connect/disconnect the LIN node to the LIN bus
• Sending "go to sleep mode" command
• Sending "wake up" command
• Bus timeout detection
• Frame monitoring
• ID field calculation
• Data length extraction
• Checksum calculation
• LIN message scheduler.

LIN Software Driver Requirements
The LIN software driver requires CPU processing time as well as hardware resources like peripherals, code/data memory, and interrupts nodes. The exact requirements are listed next.

Hardware Requirements
LIN utilizes the serial interface for message frame generation and additionally occupies one timer channel for LIN bus timing monitoring e.g. for detection of not responding slaves or no bus activity. Both peripherals require an interrupt handler, with main processes of the LIN software driver being executed from the interrupt handler of the serial communication channel peripheral.

Memory Requirements
The software driver occupies less than 3 kBytes of code memory and less than 100 bytes of data memory. These numbers include all the basic LIN software driver functions. Application specific user LIN message buffers require additional memory space.

Slave task:
1. Receive-ID-Field (message filtering), copy data to LIN transfer buffer and send first databyte
2. Receive first databyte and send second data byte
3. Receive second databyte and send third data byte
4. Receive third databyte and send fourth data byte
5. Receive fourth databyte and calculate / send checksum-field
6. Receive-Checksum

This bus configuration is running at a baudrate of 19.2 kBaud. The length of one message frame is about 4.5 ms. the required CPU time for a LIN frame generation depends strongly on the number of declared ID's. Within the example network, 16 identifiers are defined. During the states 1 to 9, the CPU of the master LIN node is busy processing LIN driver operations. The overall processing time for the LIN software driver is 110 µs (fCPU = 20 MHz). While sending and receiving a LIN message frame the total CPU load of the LIN master node is below 3%.

LIN_Protocol_ver_2.2A
SW COomponent LIN_Protocol

Division

I B&S

Automotive
Entry
Progam

8 The ISO/OSI reference model and LIN

8.1 Figure 10: reference model ISO/OSI LIN

Application Layer	→	**Applications, operating system**
Presentation Layer	→	**Conversion of data formats**
Session Layer	→	**Task synchronization, buffers, Connection setup and monitoring, Access rights control**
Transport Layer	→	**Address conversion, routing, segmentation**
Network Layer	→	**Setup of logical connection, Transport protocol**
Data Link Layer	→	**Transmission security, frame setup, Error manasegement**
Physical Layer	→	**Electrical/ mechanical characteristics: Transmission medium, wiring, Connectors, encoding, signals**

9 APPLICATION LAYER SLAVE

9.1 Layers
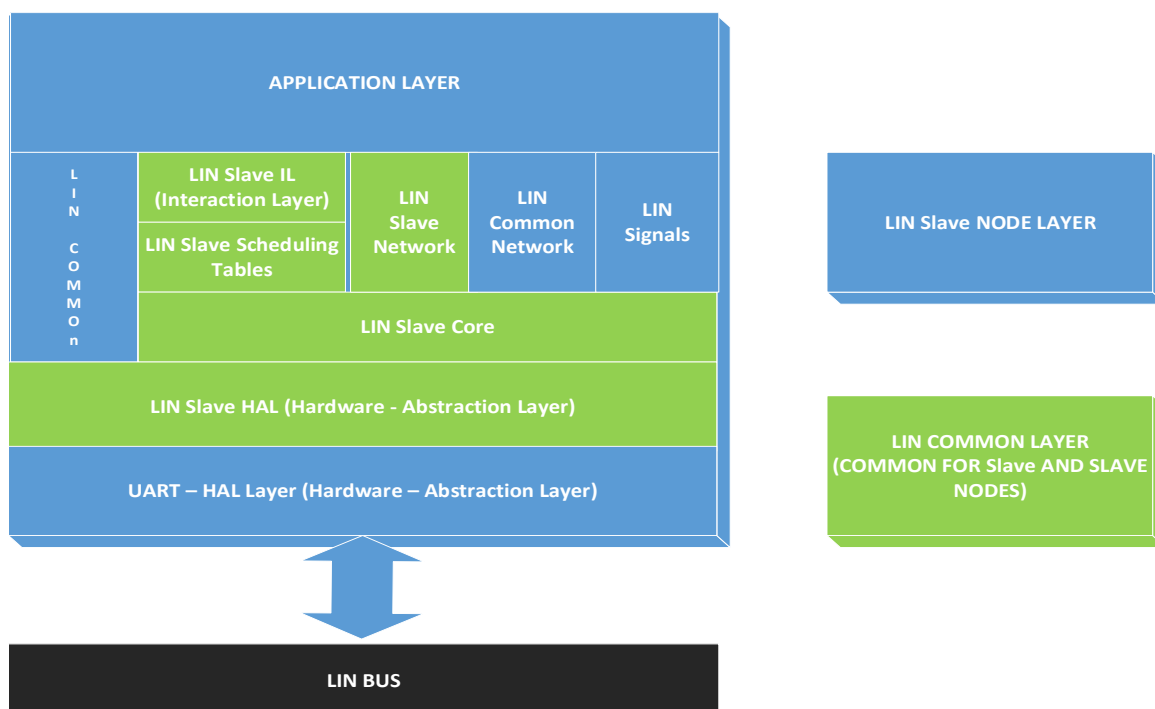
9.2 Layer descriptions

Layer	Description
Application Layer	This layer correspond to the application functionally which uses the LIN data being received by LIN and writes the data being transmitted by LIN.
LIN SLAVE Node Layer	This layer implements the functionally of the LIN Master node
LIN common layer	This layer provides implementation of functionally that is common either for a LIN master node or a LIN slave node.
UART Layer	This layer implements all UART related interfaces to be used by the LIN functionally

9.3 Functional Blocks Descriptions

Functional Block Name	Description
LIN SLAVE IL	Provides the "skeleton" of the callbacks being called by the Lin master node. Some of those callbacks are associated to messages defined in the LIN network so this file changes defined in the LIN network
LIN SLAVE Network	Defines and implements all functionality related to the LIN messages of the network where the master node is connected. This file changes each time any of the the LIN messages changes in the LIN network definition.
LIN SLAVE Scheduling Tables	Defines and implements all functionality related to the LIN master scheduling tables. This file changes each time a scheduling table or messafe changes in the LIN network
LIN SLAVE Core	Implements the core functionality of the LIN master node, i.e., the functionality for sending and receiving LIN frames. This functionality does not depend on the LIN network definition so it does not change upon changes of it

LIN SLAVE HAL	Implements the interfaces of the LIN master core functionality with the UART hardware.
LIN Common	Provides functionality that is common among any node of the LIN network, i.e., either the master node or any slave node. For example, the checksum calculation is implemented in this functional block.
LIN Common Network	Provides definitions of the LIN network which is common to all LIN nodes (master and slaves). For example, the ID and data length of the messages, etc.
LIN Signals	Provides data structures to ease the access of the individual signals of the LIN frames.
UART HAL	Implements the API's for sending and receiving bytes with the UART hardware.

10 Requeriments

Messages defined for this network.

Msg ID	Msg Name	Msg Data Length (byte)	Msg Publisher	Msg Subscribers	Msg Callback
CF	MASTER_CMD_ALL	1	MASTER	SLAVE1, SLAVE2, SLAVE3, SLAVE4	Auto
50	MASTER_CMD_SLV1	1	MASTER	SLAVE1	Auto
11	MASTER_CMD_SLV2	1	MASTER	SLAVE2	Auto
92	MASTER_CMD_SLV3	1	MASTER	SLAVE3	Auto
D3	MASTER_CMD_SLV4	1	MASTER	SLAVE4	Auto
20	SLAVE1_RSP	2	SLAVE1	MASTER	Auto
61	SLAVE2_RSP	2	SLAVE2	MASTER	Auto
E2	SLAVE3_RSP	2	SLAVE3	MASTER	Auto
A3	SLAVE4_RSP	2	SLAVE4	MASTER	Auto
F0	SLAVE1_ID	7	SLAVE1	MASTER	Auto
B1	SLAVE2_ID	7	SLAVE2	MASTER	Auto
32	SLAVE3_ID	7	SLAVE3	MASTER	Auto
73	SLAVE4_ID	7	SLAVE4	MASTER	Auto

Messages used for and by slave 2

Signal Types defined for this network.

Type Name	Elements...					
	0	1	2	3	4	5
t_slv_stat	Disabled	ENABLED				
t_cmdType	cmd_NONE	cmd_LED_on	cmd_LED_off	cmd_LED_toggling	cmd_disable_slv	cmd_enable_slv
t_LEDstat	OFF	ON	TOGGLING			
t_target_active	INACTIVE	ACTIVE				
t_boolean	FALSE	TRUE				

Signals defined for this network.

| Signal Name	Signal Msg	Signal Start Byte	Signal Start Bit	Signal Len (bits)	Signal Type	Signal Endianess
master_cmdForAll	MASTER_CMD_ALL	0	0	4	t_cmdType	LE
master_cmdForSlave1	MASTER_CMD_SLV1	0	0	4	t_cmdType	LE
master_cmdForSlave2	MASTER_CMD_SLV2	0	0	4	t_cmdType	LE
master_cmdForSlave3	MASTER_CMD_SLV3	0	0	4	t_cmdType	LE
master_cmdForSlave4	MASTER_CMD_SLV4	0	0	4	t_cmdType	LE
slave1_LEDstat	SLAVE1_RSP	0	0	2	t_LEDstat	LE
slave1_enabled	SLAVE1_RSP	1	0	1	t_boolean	LE
slave1_supplier	SLAVE1_ID	0	0	8	scalar	LE
slave1_serial	SLAVE1_ID	1	0	48	array	LE
slave2_LEDstat	SLAVE2_RSP	0	0	2	t_LEDstat	LE
slave2_enabled	SLAVE2_RSP	1	0	1	t_boolean	LE
slave2_supplier	SLAVE2_ID	0	0	8	scalar	LE
slave2_serial	SLAVE2_ID	1	0	48	array	LE
slave3_LEDstat	SLAVE3_RSP	0	0	2	t_LEDstat	LE
slave3_enabled	SLAVE3_RSP	1	0	1	t_boolean	LE
slave3_supplier	SLAVE3_ID	0	0	8	scalar	LE
slave3_serial	SLAVE3_ID	1	0	48	array	LE
slave4_LEDstat	SLAVE4_RSP	0	0	2	t_LEDstat	LE
slave4_enabled	SLAVE4_RSP	1	0	1	t_boolean	LE
slave4_supplier	SLAVE4_ID	0	0	8	scalar	LE
slave4_serial	SLAVE4_ID	1	0	48	array	LE

11 Realization constraints and targets

System must ensure reliability on window behavior and security with the anti-pinch function, push buttons functions shall be much delimited in any unusual situation from ambiguous scenarios.
The constrains to accomplish this targets are the use of microcontroller freescale TRK-MPC5606B, emulation just for 10 led bar and anti-pinch function managed only by push button with scheduler control.

The system will be running in the MPC 5606B. With the following specifications:

- MPC5606B MCU in a 144LQFP package.
- On-board JTAG connection via open source OSBDM circuit using the MPC9S08JM MCU
- MCZ3390S5EK system basis chip with advanced power management and integrated CAN transceiver.
- CAN and LIN interface.
- Analog interface with potentiometer.
- High-efficiency LEDs.
- SCI serial communication interface.
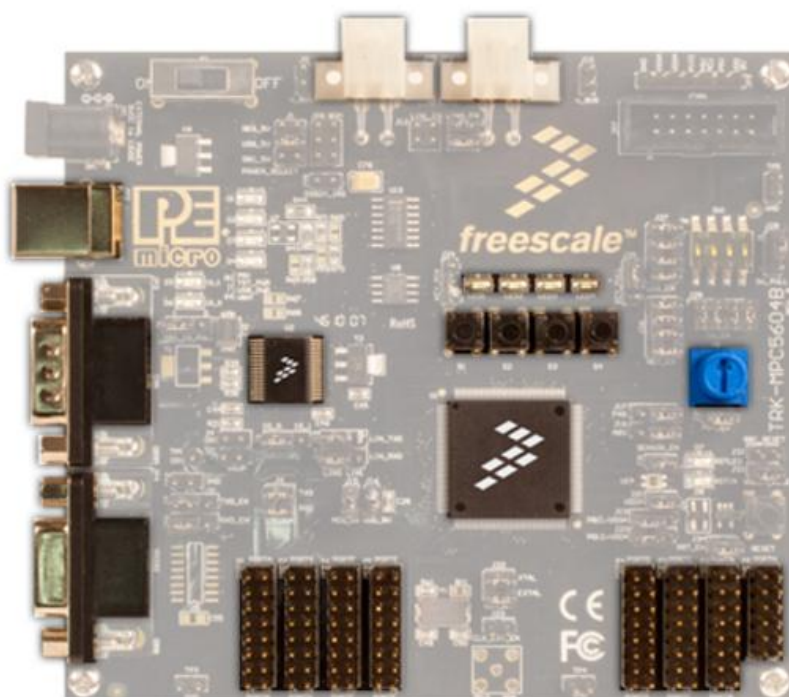
Figure 2: MPC5606B board.

Operating Frequency (Max): 64 MHz.
Total DMA Channels 16.
Internal Flash (KB): 512
GPIOs: 149.
EEPROM: 64 KB DataFlash®
RAM: Up to 96 KB
Timer: 16 bits up to 64 channels
ADC:
10 bits up to 36 channels
12 bits up to 16 channels
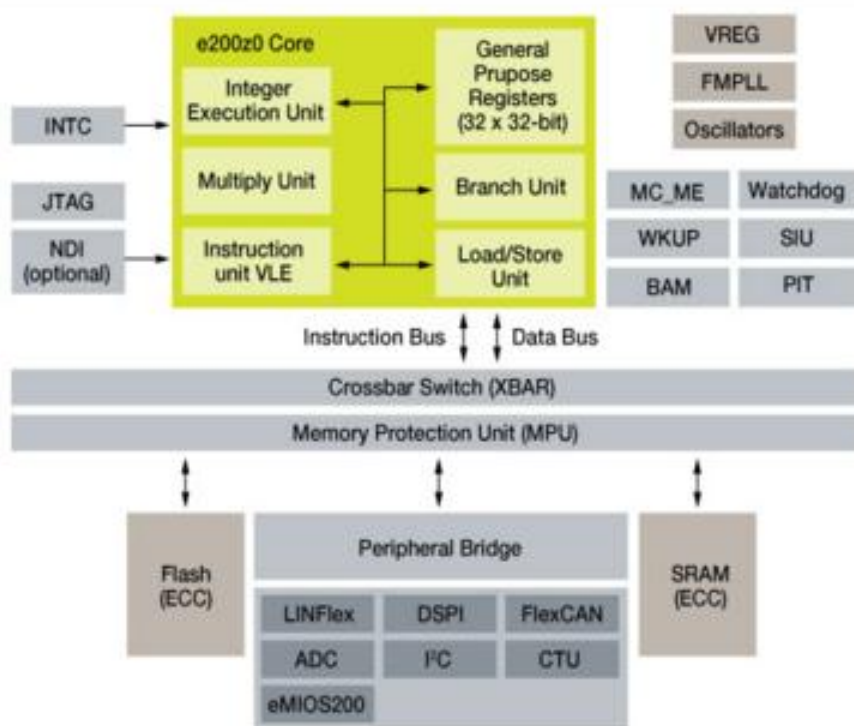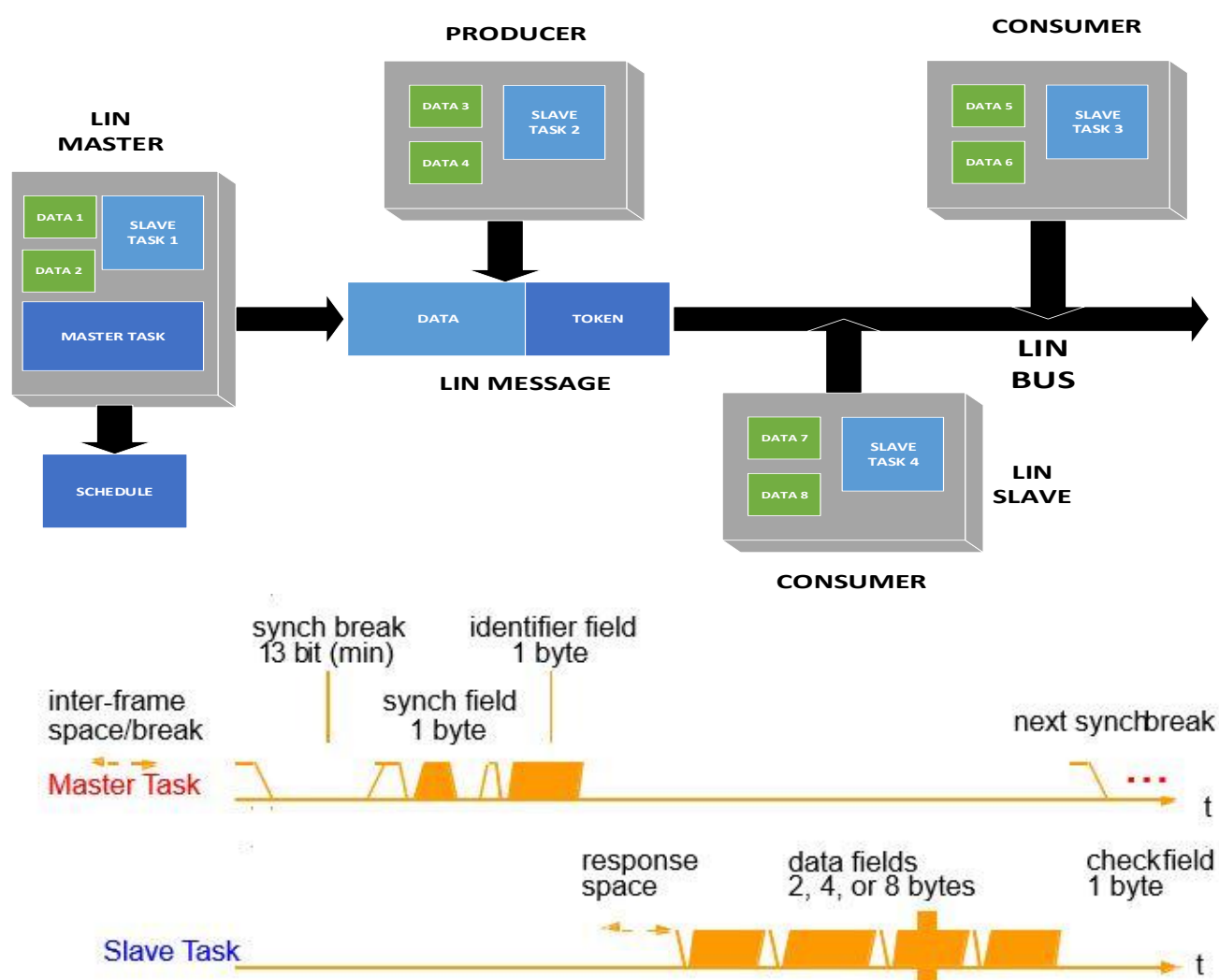Up to six CAN
Up to six SPI
Up to 10 LINFlex

Figure 3: MPC56X B/C Boolero Arquitecture Family

12 Network Graphical Description Diagrams

### 12.1 Figure 4: Deployment Diagram (A)

Deployment Diagram. Represents the physical configurations of software and hardware items.
The network showed in the figure above represents the full integrated system, which consists of the master node and the four slaves shown.

Our assignment is to develop the slave node labeled as Slave3, with all its internal components and the required actions to perform.

12.2 FIGURE 5: Use Case Diagram
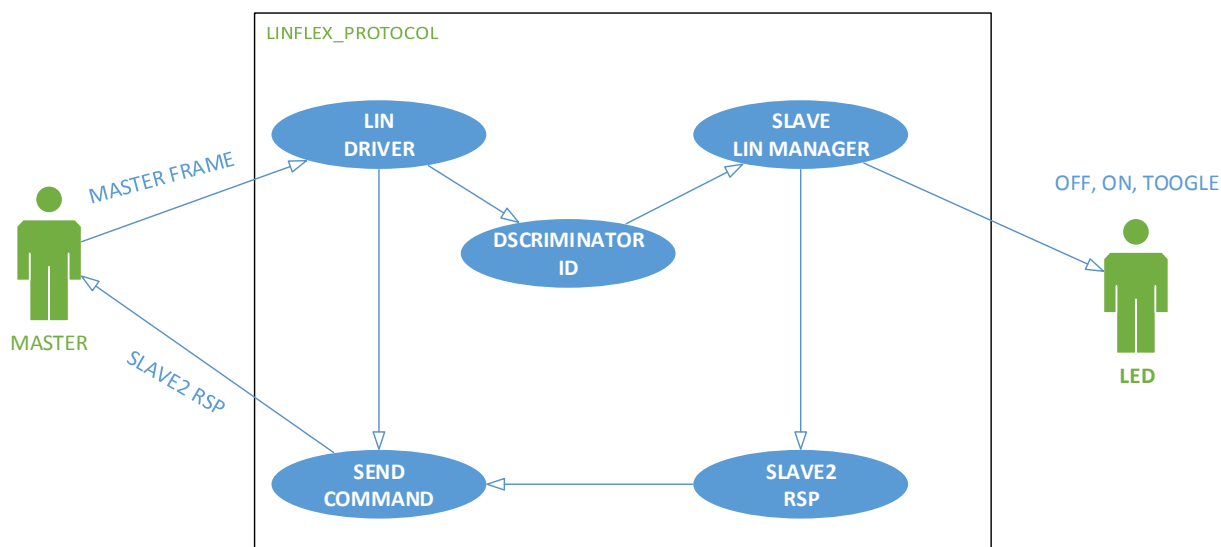
Master: Node that controls the network.
Relevant to Slave3 node are shown.
This node can send up to of 13 pre-defined messages, but in this diagram only the messages

LED: Physical On-board LEDs which will respond to the commands given by the system.

MASTER_CMD_ALL: Message sent to every slave node to execute a command.

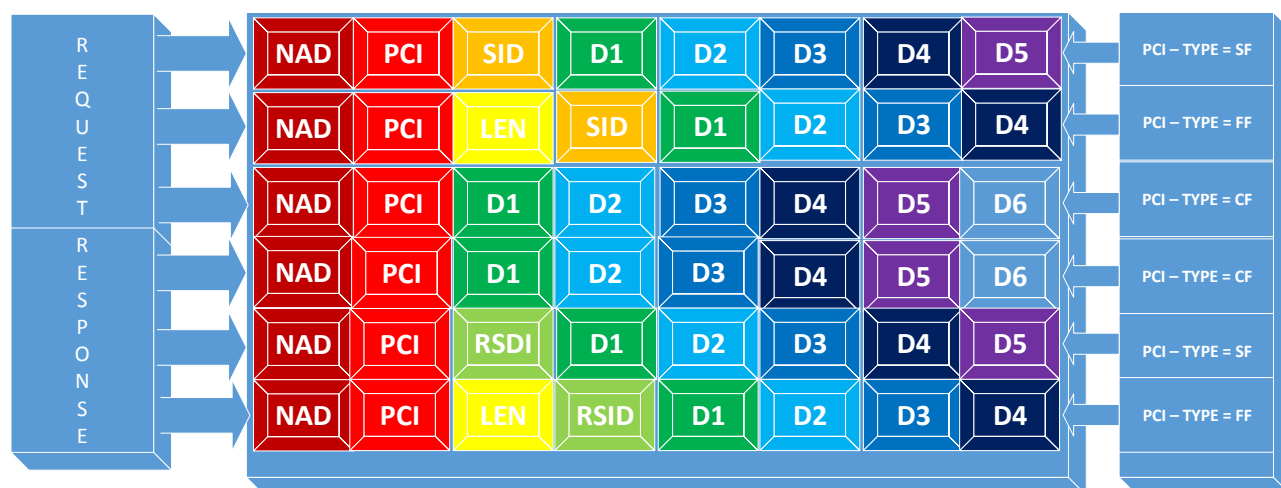**MASTER_CMD_SLV2:** Message sent specifically to slave node 3 to execute a command.

SLAVE2_RSP: Message sent from Slave3 to the master node with a specific response.

SLAVE2_ID: Message sent from Slave2 node to the master with the corresponding ID.

12.3 FIGURE 6: Component diagram request & response (A/D)

12.3.1 Components Descriptions

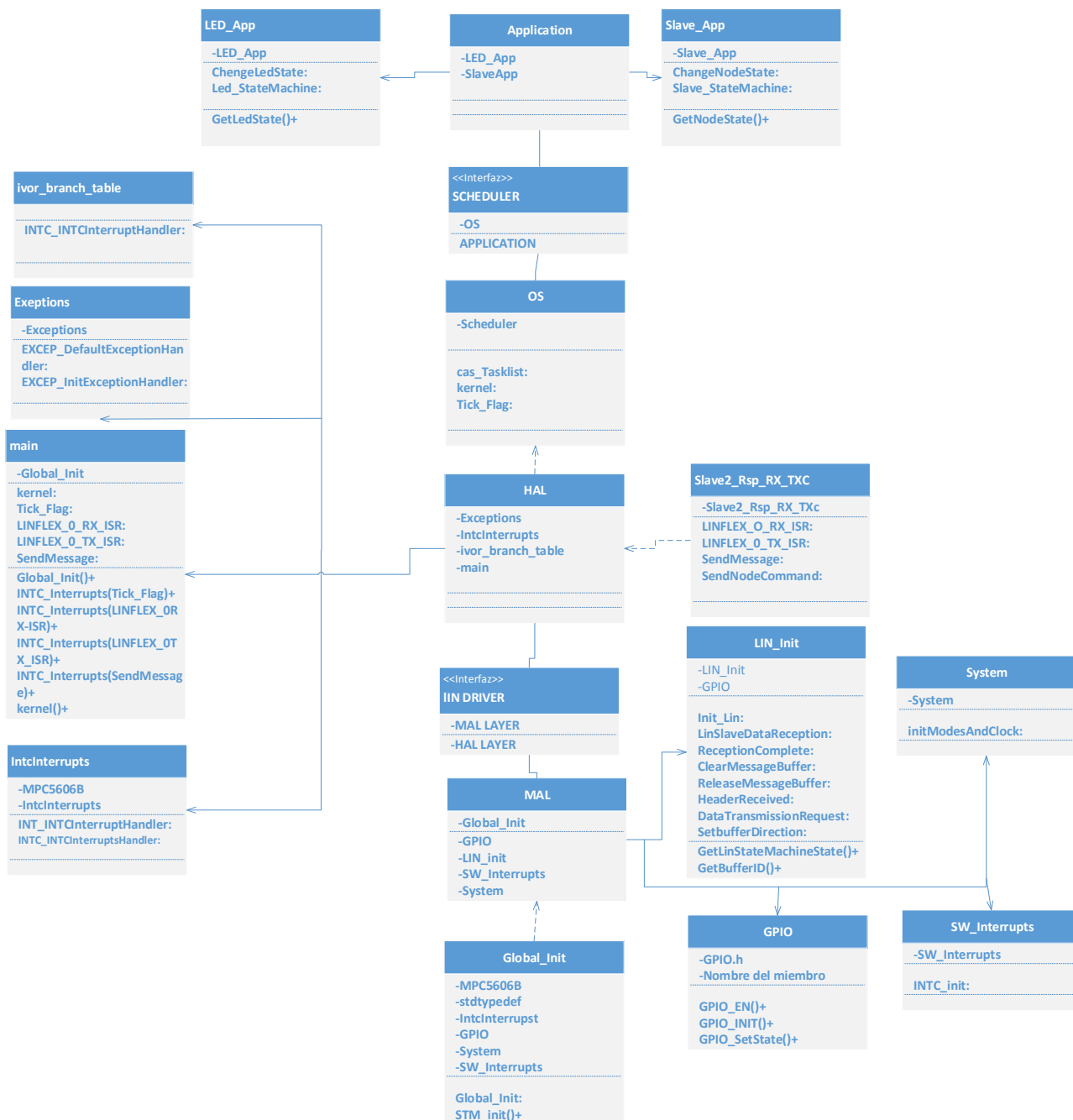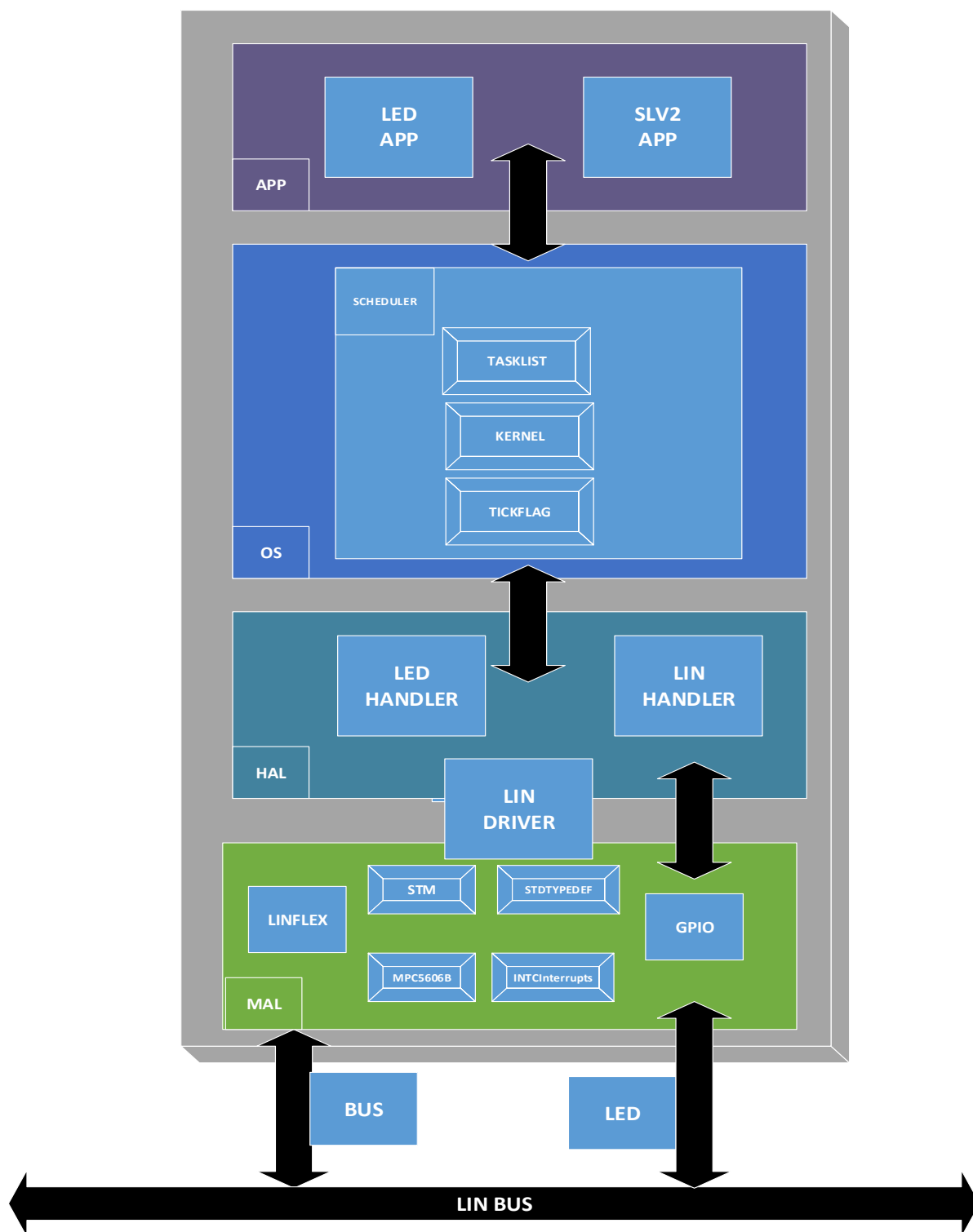| NAD	Is the address of the slave node being addressed in a request, only slave nodes have an addressed NAD is also used to indicate the source of a response. NAD values are in the range 1 to 127 (ox7F) while 0 and 128 (0x80) to 255 (oxFF) are reserved for other purpose
PCI	The PCI (Protocol Control Information) contains the transport layer flow control information. Three interpretations of the PCI byte exist,

LEN	A LEN byte is only used in FF; it contains the eight least significant bits of the message length. Thus, the maximum length of a message is 4095 (0xFFF) bytes.
SID	The service identifier (SID) specifies the request that shall be performed by the slave node addressed. 0 to 0xAF
RSID	The response service identifier (RSID) specifies the contents of the respons. The RSID for a positive response is always SID + 0x40. If the service is supported in the slave node the response is mandatory (seven if the request is broadcast). The support of a specific service will be listed in the NCF.
D0 TO D7	The interpretation of the data bytes (up to five in a node configuration PDU) depends on the SID or RSID. If a PDU is not completely filled the unused bytes shall be recessive, their value shall be 255 this is necessary since a diagnostic frame is always eight bytes long.

12.4 FIGURE 7: CLASES DIAGRAM

Detailed Software Design Document

DSD_template_Protocol_Lin.doc

1.5

Project: "Protocol_LIN_ver 2.2A" 8-dic-15

Continental

Page 27 / 43

12.5 FIGURE 8: COMPONENT DIAGRAM SW

12.6 FIGURE 9: SEQUENCE DIAGRAM
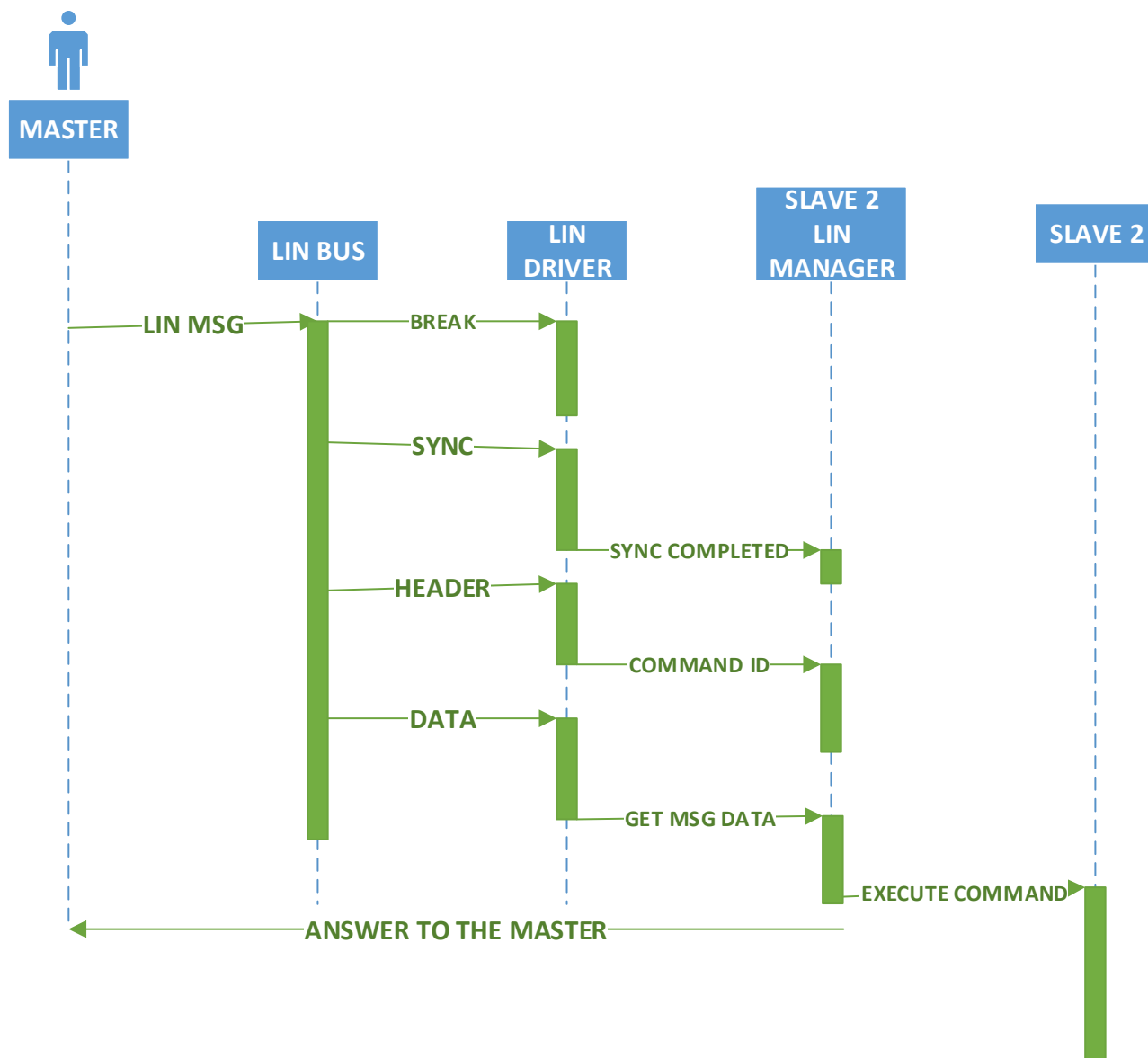
12.7 FIGURE 10: STATE MACHINE TRANSMISSION SLAVE2
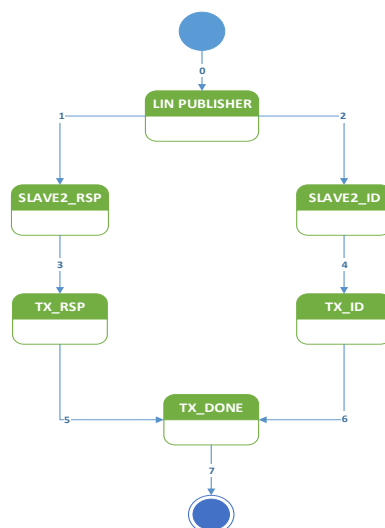
TRANSITION	CONDITION	ACTION
0	(Signal Msg == SLAVE2_RSP)II(Signal Msg==SLAVE2_ID)	Initialize LIN Publisher.
1	If Signal Msg == SLAVE2_RSP	Initialize SLAVE2_RSP.
2	If Signal Msg == SLAVE2_ID	Initialize SLAVE2_ID.
3	(ID field==valid)&&(Header==sent)	Initialize TX_RSP.
4	(ID field==valid)&&(Header==sent)	Initialize TX_ID.
5	Data TX_RSP==transmitted	Go to TX_Done.
6	Data Tx_ID==transmitted	Go to TX_Done.
7	Transmission completed.	Exit state machine.

12.8 FIGURE 11: LIN STATE MACHINE SLAVE2
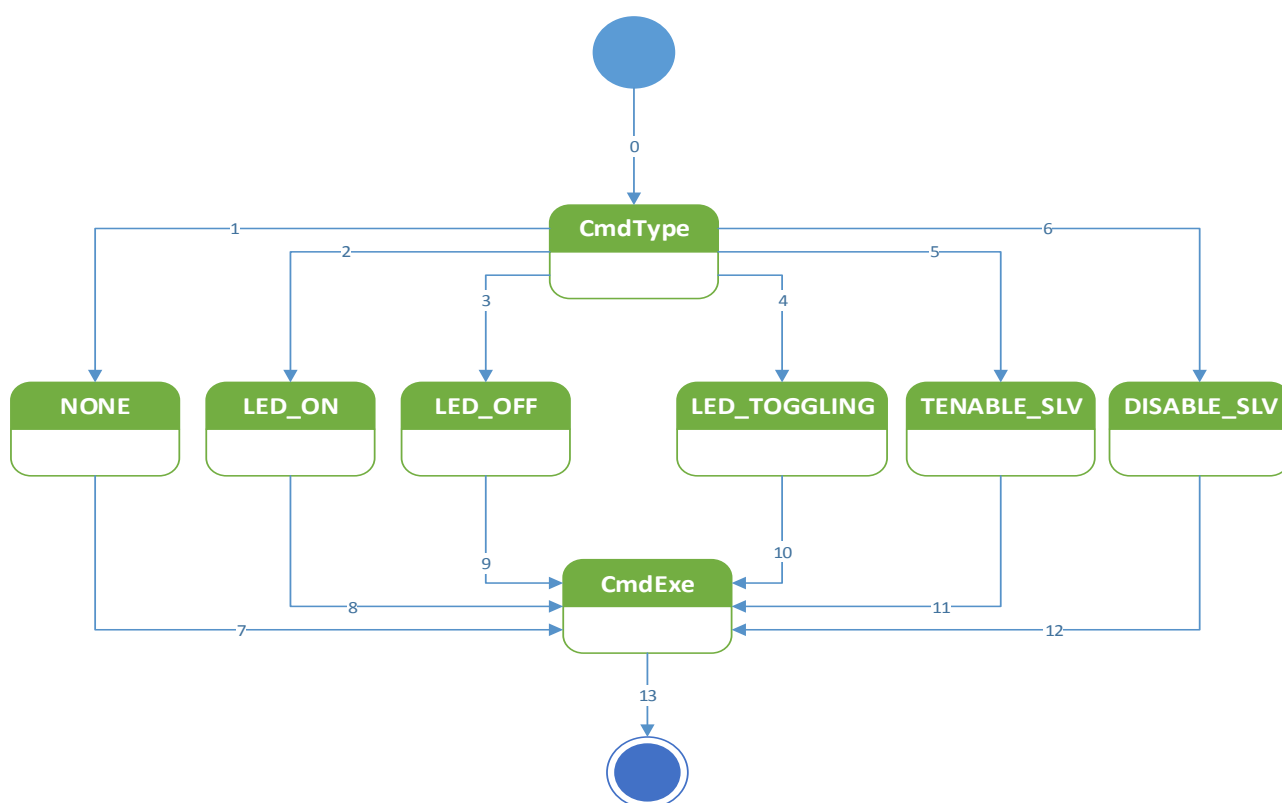
TRANSITION	CONDITION	ACTION
0	Signal Msg == MASTER_CMD	Initialize CmdType
1	t_cmdType==cmd_NONE	Initialize none.
2	t_cmdType==cmd_LED_on	Initialize LED_ON
3	t_cmdType==cmd_LED_off	Initialize LED_OFF
4	t_cmdType==cmd_LED_toggling	Initialize LED_TOGGLING
5	t_cmdType==cmd_enable_slv	Initialize ENABLE_SLV
6	t_cmdType==cmd_disable_slv	Initialize DISABLE_SLV
7	Command variable updated to None.	Go to CmdExe
8	Command variable updated to LED on.	Go to CmdExe
9	Command variable updated to LED off.	Go to CmdExe
10	Command variable updated to LED toggling.	Go to CmdExe

11	Command variable updated to Enable.	Go to CmdExe
12	Command variable updated to Disable.	Go to CmdExe
13	Cmd==executed.	Exit state machine.

12.9 FIGURE 12: LIN PROTOCOL STATE MACHINE
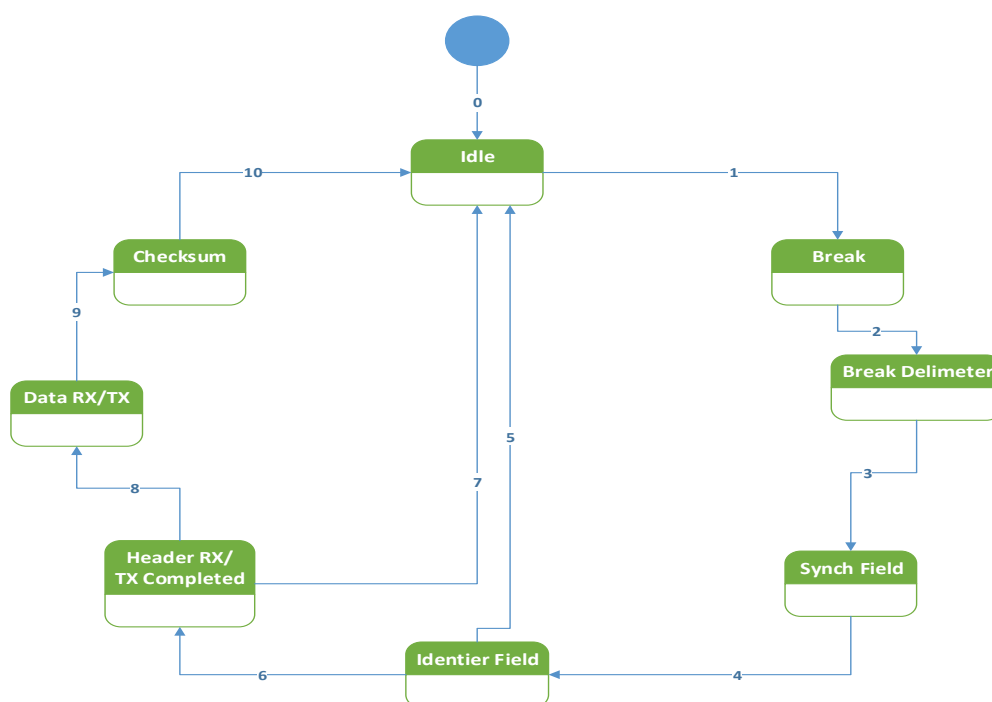
TRANSITION	ACTION
0	Initialize Idle
1	Go to break.
2	Enter Break Delimiter.
3	Enter Synch Field.
4	Initialize Identifier Field.
5	Return to Idle.
6	Enter Header Rx/ Tx compl.
7	Return to Idle.
8	Go to Data Rx/Tx.
9	Enter Checksum.
10	Return to Idle.

12.10 FIGURE 13: STATE MACHINE OF MASTER NODE TRANSMISSION HANDLER
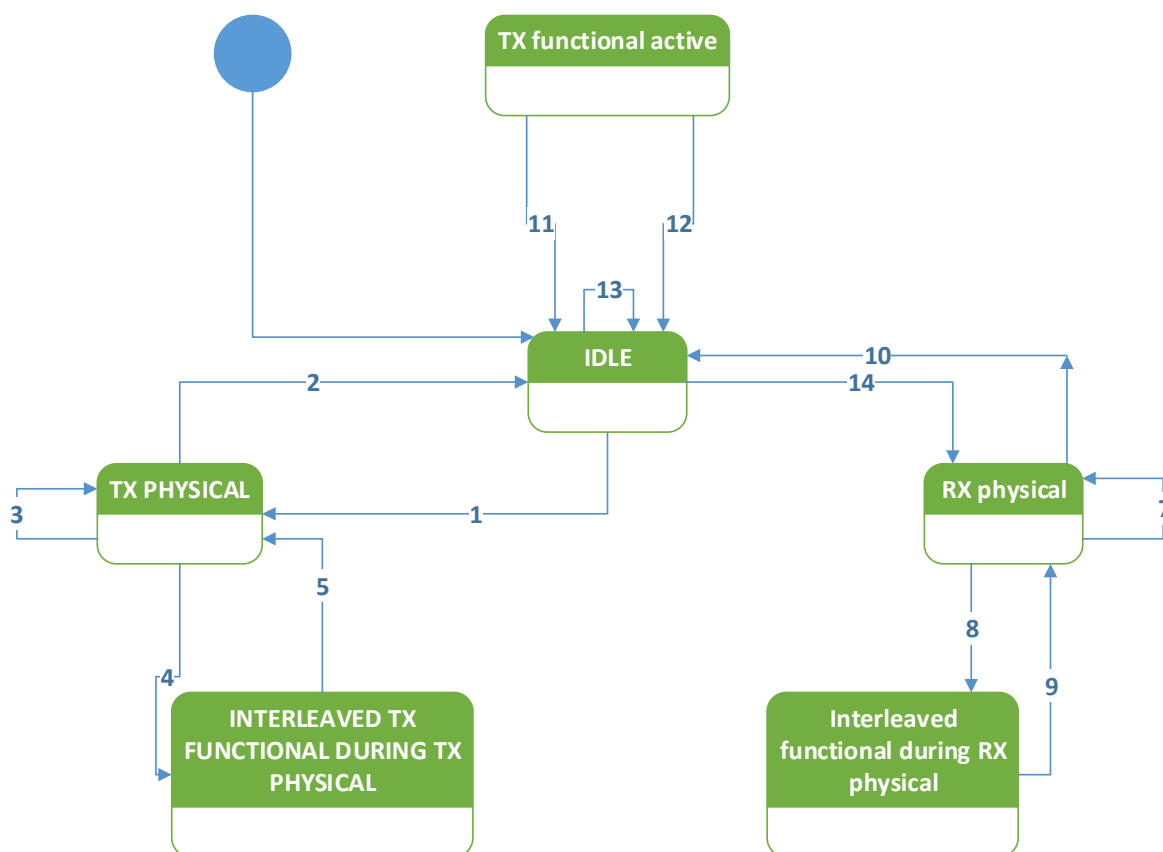
1.- IDLE STATE	Tx physical active state Trigger: Start of a physical transmission from the back-bone bus to a slave node. Effect: Start executing diagnostic master request frame schedule tables and handle the transport protocol.
2.- Tx physical active state	Idle state Condition: routing of the physical transmission from the back-bone bus to the cluster has finshed or a transport protocol transmission error Action: stop executing master request frame schedule tables.

3.- **Tx physical active state**	Tx physical active state Condition: routing of the physical transmission from the back-bone bus to the slave node has not finished Action: continue executing master request schedule table and handle transport protocol on LIN
4.- **Tx physical active state**	Interleaved functional during Tx physical state Condition: functional addressed request from the back-bonebus has been received. Action: interrupt the physical transmission to the slave node and execute a single master request frame schedule to transmit the functional addred request onto the cluster.
5.- **interleaved functional during Tx physical state**	Tx physical active state Condition: functional addressed request has been routed onto the cluster. Action: continue the interrupted physical transmission to the slave node.
6.- **Tx physical active state**	Rx physical active state Condition: functional transmission to the slave node has been successfully completed (according to the transport layer specification). Action: stop executing master request frame schedule tables, start executing slave response frame schedule tables and handle the incoming response from the previously addressed slave node.
7.- **interleaved functional during Rx physical state**	Rx physical active state Condition: functional addressed request has been routed onto the cluster. Action: restart executing slave response frame schedules and continue the interrupted reception from the slave node.

12.11 FIGURE 14: SLAVE TASK STATE MACHINE

- **Break and synch detector**

A slave task is required to be synchronized at the beginning of the protected identifier field of a frame, so it must be able to receive the protected identifier field correctly. This sequence is unique in the whole LIN communication and provides enough information for any slave task to detect the beginning of a new frame and be synchronized at the start of the identifier field.

- **Frame processor**

Frame processing consists of two states, Dormant and Active. Active contains five sub-states. As soon as BreakAndSynch is signaled the Active state is entered in the Receive Identifier sub-state. This implies that processing of one frame will be aborted by the detection of a new break and synch sequence.
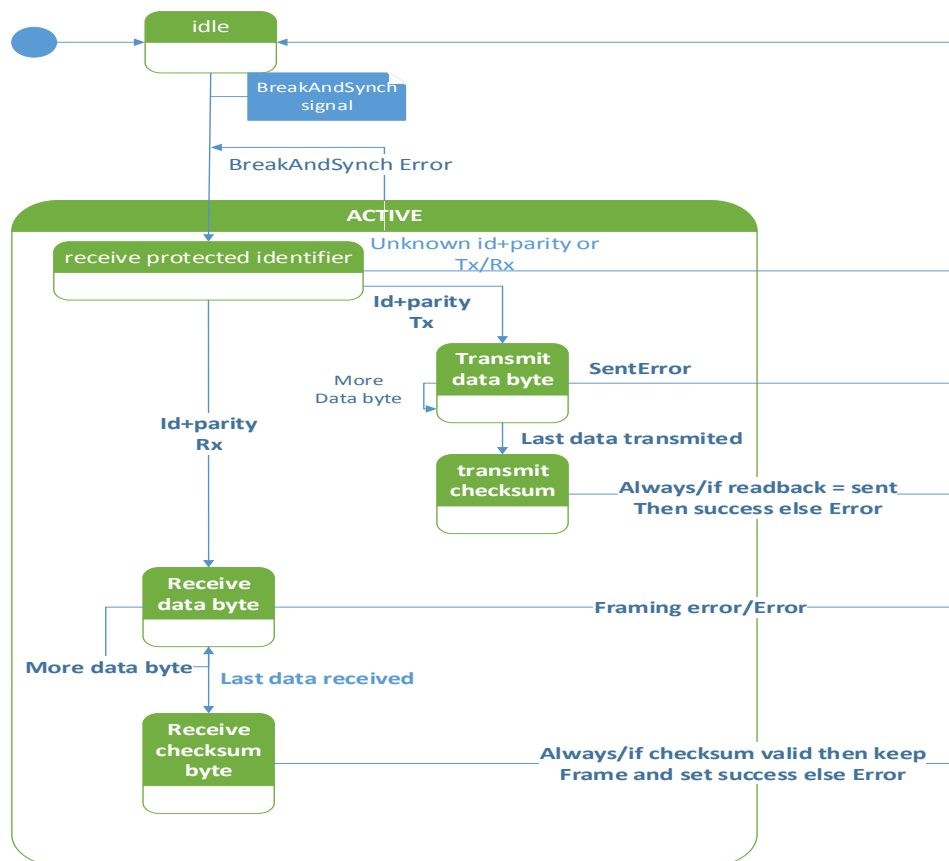
13 FUNCTIONS MAL LAYER

Global_Init.c	
NAME	Global_Init()
DESCRIPTION	Initialise the system resources
PARAMETERS	void
RETURN	void
NAME	STM_init()
DESCRIPTION	Configured the STM options and its channels
PARAMETERS	void
RETURN	void

GPIO.C	
NAME	GPIO_EN
DESCRIPTION	Program the drive enable pin
PARAMETERS	T_SWORD ch, T_ULONG state
RETURN	void
NAME	GPIO_INIT()
DESCRIPTION	Call GPIO_EN for the ports that will be used
PARAMETERS	void
RETURN	void
NAME	GPIO_SetState()
DESCRIPTION	Set the value received to the pin specified in ch
PARAMETERS	T_UWORD, T_UBYTE
RETURN	void

Detailed Software Design Document

DSD_template_Protocol_Lin.doc

1.5

Project: "Protocol_LIN_ver 2.2A"   8-dic-15

Continental

Page 36 / 43

LIN_Init.c	
NAME	Init_Lin()
DESCRIPTION	-Send Init mode request -INIT_MODE /*SLEEP=0, INIT=1*/ -Enable LIN transmission cannel 0 -Config Baud rate -Enter normal mode with filters -Master CMD for all filters -Master CMD slv2 filter -Slv2 automatic resynch enable
PARAMETERS	Void
RETURN	Void
NAME	LinSlaveDataReception()
DESCRIPTION	Receive the Buffer data register least significant field (BDRL) Data 0 to data 7
PARAMETERS	Unsigner char
RETURN	Void
NAME	GetLINStateMachineState()
DESCRIPTION	LINSR Lin Status Register Describe LINFLEX LIN STATUS
PARAMETERS	Unsigned char
RETURN	void
NAME	ReceptionComplete()
DESCRIPTION	Program the drive enable pin
PARAMETERS	Void
RETURN	void
NAME	HeaderReceived()
DESCRIPTION	This bit is set by hardware and indicates a valid header reception is completed. This bit must be cleared by software.
PARAMETERS	Void
RETURN	Void

NAME	INTC_init()
DESCRIPTION	Activates the use of interrupts
PARAMETERS	Void
RETURN	void

System.c

NAME	InitModesAndClock()
DESCRIPTION	Initialise the modes and clock of the MPC5606B
PARAMETERS	Void
RETURN	void

13.1 FUNCTIONS OS LAYER

NAME	Kernel()
DESCRIPTION	This function starts by doing an indexing of the task, assigning the offset. This process occurs only once in each execution of the code. Then an infinite loop starts, the tick of the scheduler is defined by a flag called ub_TickFlag, which was configured to raise every millisecond. Once it was acquired it is restarted to zero. Now, inside the if there is a for cycle which who through all the task accessing to their offset with and if, whenever the offset is over zero it will decrease, for every task. When one of the offset, of whichever task, reach zero, the else of the if will be active and will <u>stablish</u> the period of the task and will call it using the pointer previously defined.
PAAMETERS	void
RETURN	void

NAME	Tick_Flag()
DESCRIPTION	Check if the cannel 0 of the STM as reached 1ms and raise a flag when reached
PARAMETES	Void
RETURN	void

13.2 FUNCTIONS HAL LAYER

NAME	LINFLEX_0_RX_ISR()
DESCRIPTION	Modified rx_data – receive data Switch select both -master cmd all -master cmd slv2 id
PARAMETERS	None
RETURN	none

```c
switch(lin_bidr)
{
    case MASTER_CMD_ALL_ID:
        //LINFLEX_0.BIDR.B.DFL = 0; one byte verify page 526
        lub_LinStatus = GetLINStateMachineState();
            /* wait for RMB */
        ReleaseMessageBuffer();
        /* get the data */
        LinSlaveDataReception(laub_RxData);
        ProcessCommand(laub_RxData);
        ReceptionComplete();
        ClearMessageBuffer();
        //HeaderReceived();
        break;
    case MASTER_CMD_SLV2_ID:

        //LINFLEX_0.BIDR.B.DFL = 0; one byte verify page 526
        lub_LinStatus = GetLINStateMachineState();
            /* wait for RMB */
        ReleaseMessageBuffer();
        /* get the data */
        LinSlaveDataReception(laub_RxData);
        ProcessCommand(laub_RxData);
        ReceptionComplete();
        ClearMessageBuffer();
        //HeaderReceived();

        break;
    default:
        break;
}
```

NAME	LINFLEX_0_TX_ISR()
DESCRIPTION	Modified tx_data – transmit data Switch select both -master cmd slave2 rsp id -master cmd slave id id "F,J,Q,D,A,D"
PARAMETERS	None
RETURN	none

CODE

```c
switch(lub_lin_bidr){
case SLAVE2_RSP_ID:
        SetBufferDirection(TRANSMIT); /* BDR direction - write */
        /* fill the BDR registers */
        /*Return Led status*/
        LINFLEX_0.BDRL.B.DATA0 = GetLedState();
        /*Return Node status*/
        LINFLEX_0.BDRL.B.DATA1 = GetNodeState();
        //LINFLEX_0.BIDR.B.DFL = 1;  Verify page 525, sec. 24.7.2.1
        /* trigger the data transmission */
        DataTransmissionRequest();
        HeaderReceived();
    break;
case SLAVE2_ID_ID:
        SetBufferDirection(TRANSMIT); /* BDR direction - write probably not necessary*/
        /* fill the BDR registers */
        LINFLEX_0.BDRL.B.DATA0 = 4;
        LINFLEX_0.BDRL.B.DATA1 = 'F';
        LINFLEX_0.BDRL.B.DATA2 = 'J';
        LINFLEX_0.BDRL.B.DATA3 = 'Q';
        LINFLEX_0.BDRM.B.DATA4 = 'D';
        LINFLEX_0.BDRM.B.DATA5 = 'A';
        LINFLEX_0.BDRM.B.DATA6 = 'D';
        /* trigger the data transmission */
        DataTransmissionRequest();
        HeaderReceived();
        break;
default:
    break;
}
```

NAME	SendMessage()
DESCRIPTION	Emulates the master commands of function ProcessComman by using the third switch of the board. When switch is pressed and interrupt take place. Each command correspond to a value of counter variable. This function was created for testing purposes to verify the correct functionality of the state machines.
PARAMETERS	Void
RETURN	Void

<table>
<tr><th colspan="2">CODE</th></tr>
<tr><td colspan="2">

```
if (SWITCH_INTERR_FLAG)
{
    switch (counter)
        {

            case 2:
                SendLedCommand(ON);
                break;
            case 3:
                SendLedCommand(OFF);
                break;
            case 4:
                SendLedCommand(TOGGLING);
                break;
            case 5:
                SendNodeCommand(DISABLED);
                counter = 0;
                    break;
            case 1:
                SendNodeCommand(ENABLED);
                break;
            default :
                break;
        }

        SWITCH_INTERR_FLAG = 1;
}
else
{
  /*DO NOTHING*/
}
```

</td></tr>
</table>

NAME	ProcessCommand()
DESCRIPTION	Reads the response of the master and determines which command the slave will execute: whether to do nothing, request the Led_StateMachine to turn on/off or toggle, or request the Node_StateMachine to disable or enable the node.
PARAMETERS	Void
RETURN	Void
CODE	

```c
switch (laub_Msg[0])
{
case CMD_NONE:
    break;
case CMD_LED_ON:
    SendLedCommand(ON);
    break;
case CMD_LED_OFF:
    SendLedCommand(OFF);
    break;
case CMD_LED_TOGGLING:
    SendLedCommand(TOGGLING);
    break;
case CMD_DISABLE_SLV:
    SendNodeCommand(DISABLED);
        break;
case CMD_ENABLE_SLV:
    SendNodeCommand(ENABLED);
    break;
default :
    break;
}
```

13.3 FUNCTIONS APPLICATION LAYER

NAME	ChangeLedState()
DESCRIPTION	Change the state variable depending on the command.
PARAMETERS	T_UBYTE POSIBLE STATES: OFF, ON, TOOGLE
RETURN	VOID

NAME	LED_STATEMACHINE()
DESCRIPTION	This state machine will be called every 1ms. Initial state is OFF. Led variable state could have three states: OFF, ON, and TOGGLING. If the node is enabled then the master command is listened. When the command is received the rub_LedFlag is changed depending on the command. The state variable will be changed in the subsequent tick. If the node is disable, the current state is kept, until a different command, when node is enabled, is listened.
PARAMETERS	VOID
RETURN	VOID

NAME	SLAVE_STATEMACHINE()
DESCRIPTION	This state machine will be called every 500ms. Initial state is DISABLE. Node variable state could have two states: DISABLED and ENABLED. When master commands, the state variable is changed by using rub_NodeFlag variable that is the command.
PARAMETERS	VOID
RETURN	VOID