|  |
| --- |
| EMBBEDED SYSTEMS SPECIALIZATION PROGRAMM |
| G2 Project #1 LIN |
| Communications Software Development for Embedded Environments |

|  |
| --- |
| **Feliciano Angulo Angulo**  Email: [se704751@iteso.mx](mailto:se704751@iteso.mx)  **Juan Pablo Garcia**  Email: [se716963@iteso.mx](mailto:se716963@iteso.mx)  **Jesus Íñiguez**  Email: [se717040@iteso.mx](mailto:student3_email@iteso.mx) |

**29/03/2018**

Embedded Systems Specialization Program

[www.sistemasembebidos.iteso.mx/alumnos](http://www.sistemasembebidos.iteso.mx/alumnos)

I T E S O A. C., Universidad Jesuita de Guadalajara

Periférico Sur Miguel Gómez Morín #8585, Tlaquepaque, Jalisco, México

Technical Report Number: ESE-O2014-001

® ITESO A.C.

|  |
| --- |
| ***Abstract:***  *This document shows how LIN protocol is implemented using an UART driver in in SAMV71 XPlained evaluation board. Lin and UART implementation are AUTOSAR compliant, whit this implementation is possible to transmit LIN frame through UART, Scope’s captures are included to show LIN frame from Tx line. In first implementation only header was sent, but in this second version a compete master response was implemented.* |
| ***Keywords:***  *LIN*  *UART*  *AUTOSAR*  *FRAME* |

# Table of Contents

[Table of Contents ii](#_Toc512291618)

[1. Table of Figures iii](#_Toc512291619)

[2. Introduction 1](#_Toc512291620)

[2.1. Requirements 1](#_Toc512291621)

[2.1.1. Architecture 1](#_Toc512291622)

[2.1.2. Implementation 2](#_Toc512291623)

[3. Functional Description 4](#_Toc512291624)

[3.1. AUTOSAR description 4](#_Toc512291625)

[3.2. LIN functions description 6](#_Toc512291626)

[3.2.1. State description 8](#_Toc512291627)

[3.2.2. IDLE 8](#_Toc512291628)

[3.2.3. SEND\_BREAK 8](#_Toc512291629)

[3.2.4. SEND\_SYNC 9](#_Toc512291630)

[3.2.5. SEND\_PID 9](#_Toc512291631)

[3.2.6. SEND\_RESPONSE 10](#_Toc512291632)

[3.2.7. SEND\_CHKSUM 10](#_Toc512291633)

[3.2.8. READ\_RESPONSE 10](#_Toc512291634)

[3.2.9. GET\_CHECKSUM 11](#_Toc512291635)

[3.2.10. IDLE 11](#_Toc512291636)

[4. Results 11](#_Toc512291637)

[5. Conclusions 15](#_Toc512291638)

# **Table of Figures**

[Figure 1 AUTOSAR file structure 4](#_Toc512291597)

[Figure 2 AUTOSAR layers 5](#_Toc512291598)

[Figure 3 Lin\_Init implementation 6](#_Toc512291599)

[Figure 4 Lin cfg details 7](#_Toc512291600)

[Figure 5 Lin\_SendFrame implementation 7](#_Toc512291601)

[Figure 6 Lin\_Isr implementation 8](#_Toc512291602)

[Figure 7 SEND\_BREAK 9](#_Toc512291603)

[Figure 8 Reduce BaudRate Formula 9](#_Toc512291604)

[Figure 9 SEND\_SYNC 9](#_Toc512291605)

[Figure 10 SEND\_PID 10](#_Toc512291606)

[Figure 11 SEND\_RESPONSE 10](#_Toc512291607)

[Figure 12 SEND\_CHKSUM 10](#_Toc512291608)

[Figure 13 READ\_RESPONSE 11](#_Toc512291609)

[Figure 14 GET\_CHKSUM 11](#_Toc512291610)

[Figure 15 IDLE 11](#_Toc512291611)

[Figure 16 LIN frame 12](#_Toc512291612)

[Figure 17 Header capture from Oscilloscope 12](#_Toc512291613)

[Figure 18 Bit time measure 13](#_Toc512291614)

[Figure 19 Details of generated Header LIN frame 13](#_Toc512291615)

[Figure 20 MASTER RESPONSE FRAME 13](#_Toc512291616)

[Figure 21 SLAVE RESPONSE FRAME 14](#_Toc512291617)

# Introduction

LIN (Local Interconnect Network) is a serial network protocol used for communication between components in vehicles. LIN implementation is based in UART/SCI hardware, so, when a microcontroller does not have a LIN peripheral integrated but a UART, it can be implemented over this UART easily.

The purpose of this work is to demonstrate the above description, Lin can be implemented over UART. The goal in this second project is to send a complete LIN frame over the UART, specifically the functionality of a master node, this is, a complete MASTER RESPONSE frame and a SLAVE\_RESPONSE frame shall be implemented on this project.

## Requirements

Provide a SW solution with defined interfaces that will allow to form the LIN Frame through the UART controller available in the SAM V71 controller.

Software must be based on the attached Base Project.

**Objective:** Implement a functional LIN driver

1.- Based on the Task #1, implement a LIN driver that meets the LIN Specification

2.- Note:

Std\_Types.h shall provide the Std\_ReturnType data type as an enumeration with the values of:

E\_OK: 0

E\_NOK: 1

3.- LIN implementation shall follow specification from next file.



### Architecture

The SW must be compliant with the AUTOSAR SW layered architecture, so a File Structure must be provided. This means that (if using UART interfaces) the UART driver interfaces will be at MCAL Layer and LIN interfaces will be at the ECU Abstraction Layer.

File Structure:

../source

->/bsw

->/mcal

->/com

->/uart

(uart files here)

->/ecual

->/com

->/linif

(lin files here)

->/services

->/com

->linNm

(lin files here)

->/system

->/scheduler

(scheduler files here)

Note: In the case that uart driver is used from the provider, file structure shall be kept AS IS.

### Implementation

#### lin\_cfg implementation

Lin\_Cfg.h –for definition configurable parameters, LIN configuration types: It contains **LinChannelType** and **LinConfigType** according to section 5.1.1 LINCONFIG in Lin Specification.pdf file.

Lin\_Cfg.c –for configurable parameters: **LinChennel[]** and **LinConfig** are created here to configure Lin driver.

#### lin\_Types implementation

Consists of types specified within LIN Interface except for configuration types defined above:

**LinStateType, LIN\_ENHANCED\_CS, LinFrameCsModelType, LinFrameResponseType, LinFramePidType, LinFrameDlType, LinPduType** according to section 3.2 Types Definition in Lin Specification.pdf file.

#### lin\_Nm function implementation

**void LinNm\_InitData (void)**

-This function contains initialization of any available Pdu data to be sent

**void LinNm\_10ms (void)**

-this function is the interface with upper layer to be called when a Lin transmission/reception need to be started.

During this function the system is going to wait for the Slaver Response.

#### linf function implementation

**void Lin\_Init (const LinConfigType\* Config)**

- This function will configure the lower layer UART driver

- The function will configure all the logical instances declared linking it with physical channel and its characteristics defined.

- In this function the global variables are set with the parameter gave enabling tx and rx interrupts for the uart hardware.

- Interrupts (NVIC), PIO and clocks shall be configured for each instance.

- Initialize parameter of the LinCtlChnl structure to be used in future operations

- Build and structure with Phy-to-log channels

**Std\_ReturnType Lin\_SendFrame (uint8\_t Channel, LinPduType\* PduInfoPtr)**

- This function will send a complete LIN frame (MASTER RESPONSE) as per the LIN protocol with the rate define in the Lin\_Init function.

- This function shall be asynchronous, i.e. it will trigger the "send command" and will continue its operation without waiting for the header to be completely sent over the bus.

- The frame shall be composed in order of:

- 1. Break = (from 10 to 13 bit times) ideally >= 13 bit time

- 2. Synch = 0x55

- 3. ID = LinPid

-4. Data

-5. Checksum

-In this function the logical channel is sent by parameter and is used to:

-assign the PDU

-calculate the checksum (Lin\_CheckSum(Channel)) and parity bits

-Define Slave or Master Task request

In order to support the underlined LIN SW component, a state machine was implemented in next function:

**void Lin\_Isr(uint8\_t PhyChannel)**

-Search the logical channel from the physical structure

-This function handles the Sate machine workflow, always is called indirectly by lower layer(uart)

The workflow follows enumeration

typedef enum

{

IDLE = 0,

SEND\_BREAK,

SEND\_SYNC,

SEND\_PID,

SEND\_RESPONSE,

READ\_RESPONSE,

GET\_CHKSUM,

SEND\_CHKSUM,

SEND\_EOT,

}LinStateType;

The state machine sequence is as follows

IDLE -> SEND\_BREAK -> SEND\_SYNC -> SEND\_PID ->

\*SEND\_RESPONSE ->SEND\_CHKSUM

\*READ\_RESPONSE ->GET\_CHKSUM

* IDLE: No activity.
* SEND\_BREAK: Send break process is about to start or in progress.
* SEND\_SYNC: Send sync process is about to start or in progress.
* SEND\_PID: Send pid and defines if the system has to read some response or just send response.
* SEND\_RESPONSE: Send master response data is about to start or in progress, define the workflow to send chksum.
* READ\_RESPONSE: read response is about to start or in progress, define the workflow to get chksum.
* SEND\_CHECKSUM: Send checksum is about to start or in progress, sends to IDLE.
* GET\_CHECKSUM: Get checksum and validate it, sends to IDLE.

# Functional Description

## AUTOSAR description

To make implementation AUTOSAR compliant, the file structure was created according to requirement number 1, it can be seen in figure 1 with required files in specify folder.

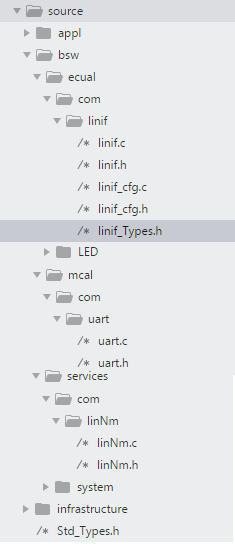


Figure 1 AUTOSAR file structure

Interaction between files showed in Figure 1, are based in AUTOSAR layer format, Figure 2 shows such layers, according to this, UART driver is part of MCAL, LIN is implemented as part of ECU Abstraction and LinNm is in services layer, and a 10 ms task is part of the application layer, all executions are controlled by the scheduler, which is part of services laver as well.

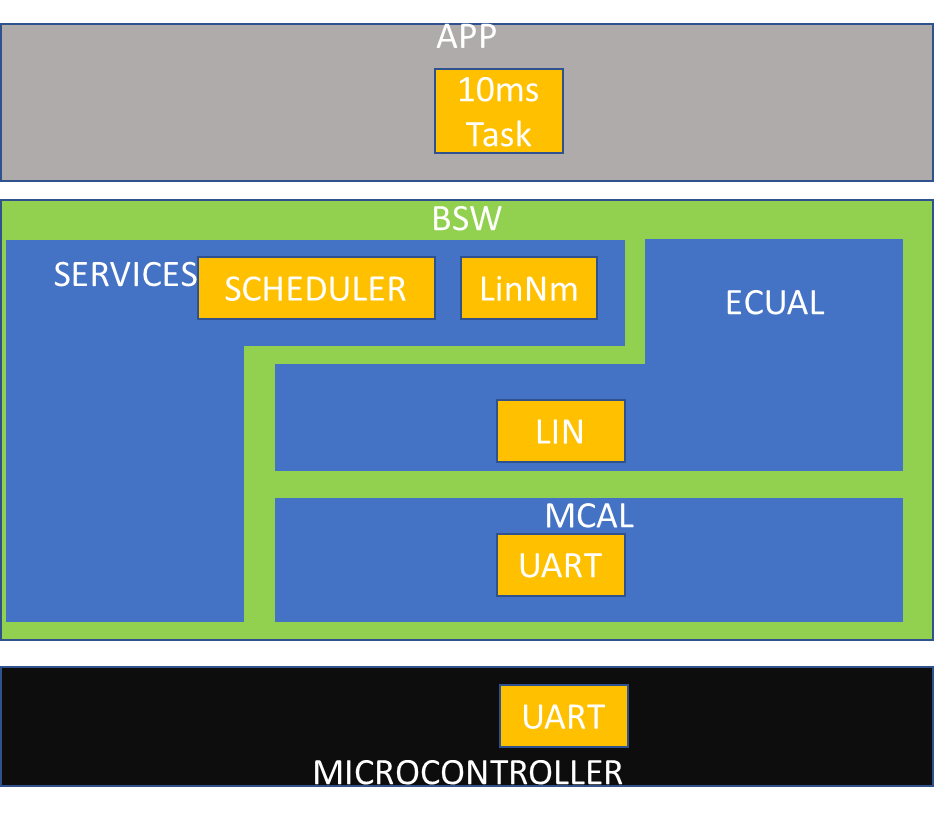


Figure 2 AUTOSAR layers

## LIN functions description

There are 3 main function in Lin to be implemented, Lin\_Init, Lin\_SendFrame and Lin\_Isr,

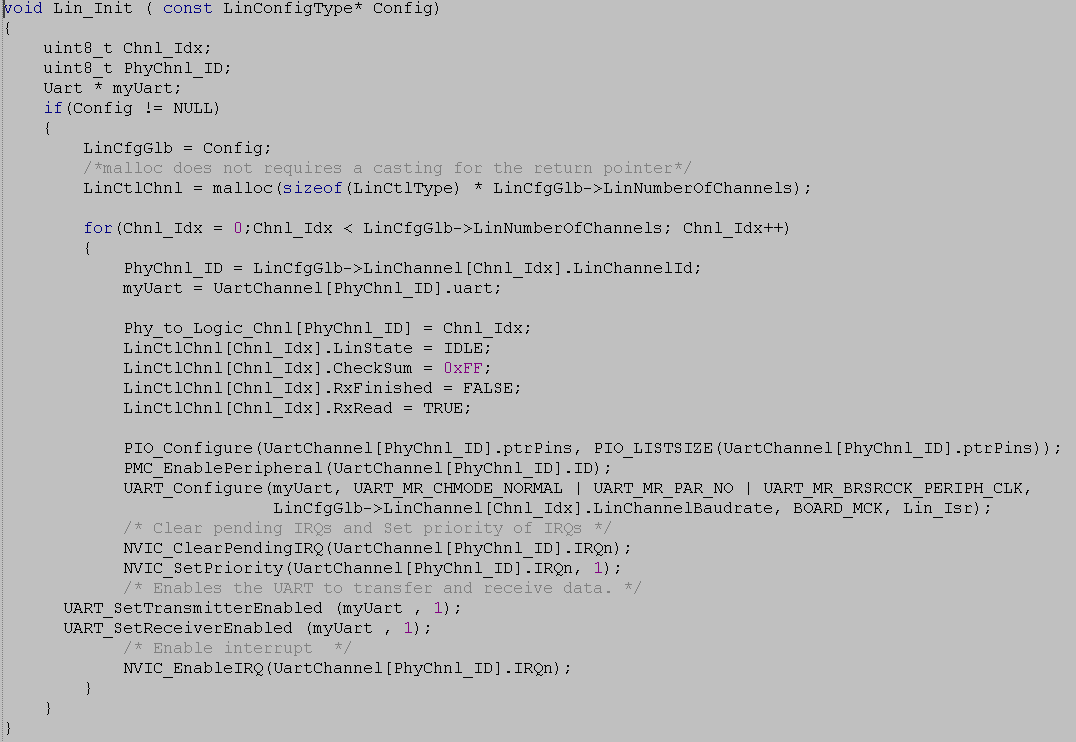


Figure 3 Lin\_Init implementation

As was mentioned before, LIN implementation is in ECU abstraction layer, and according to AUTOSAR, lower interface(MCAL) must be call from here, in this case, needed function from UART driver must be called here, as can be seen in Figure 3, Lin\_Init function initialize UART driver, first enabling used GPIO then UART\_Configure function is called to configure UART, and at the end interruptions are set. All previous configurations are done for all available Lin channels in the configuration list.

In Scope of current work, the channels are defined and sent by parameter, configuring each of them.

They are defined in linif\_cfg.c

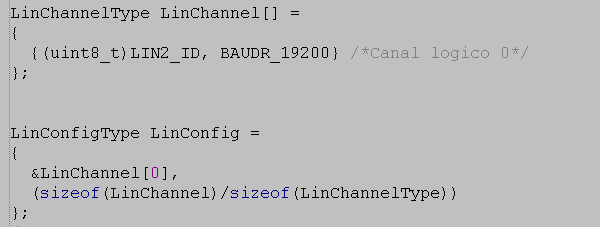


Figure 4 Lin cfg details

For our Lin channel one UART channel is used, details are in Table 2.

Table 1 UART I/O Lines details

|  |  |  |  |
| --- | --- | --- | --- |
| Instance | Signal | IO Line | Peripheral |
| UART2 | URXD2 | PD25 | C |
| UART2 | UTXD2 | PD26 | C |

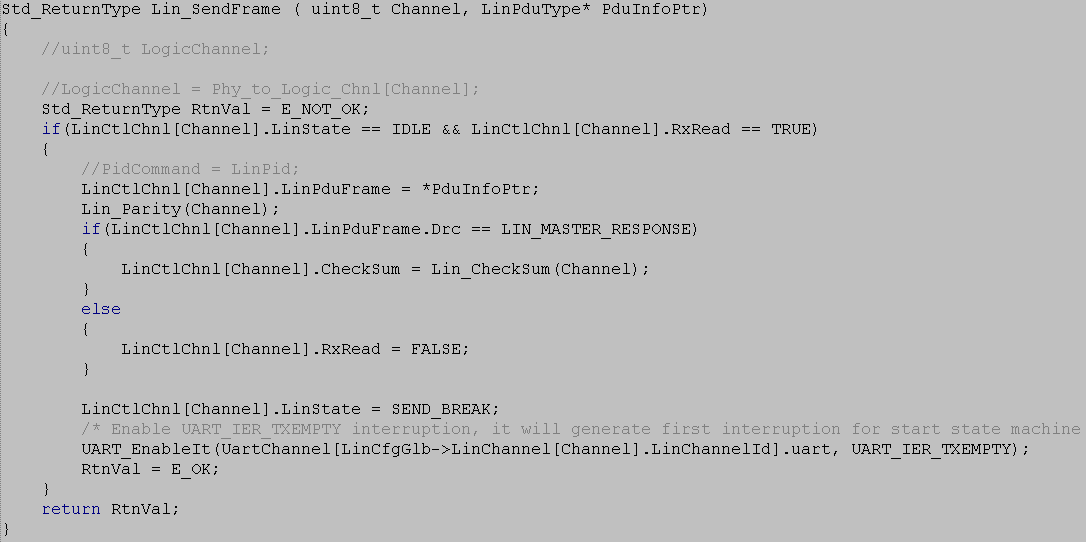


Figure 5 Lin\_SendFrame implementation

**Lin\_SendFrame** is set to initialize a LIN frame message transmission (see Figure 5), LIN frame transmission is implemented based in a state machine for send each part of the frame.

Before starting the state machine, checksum is calculated in case of current LIN frame type is a LIN\_MASTER\_RESPONSE.

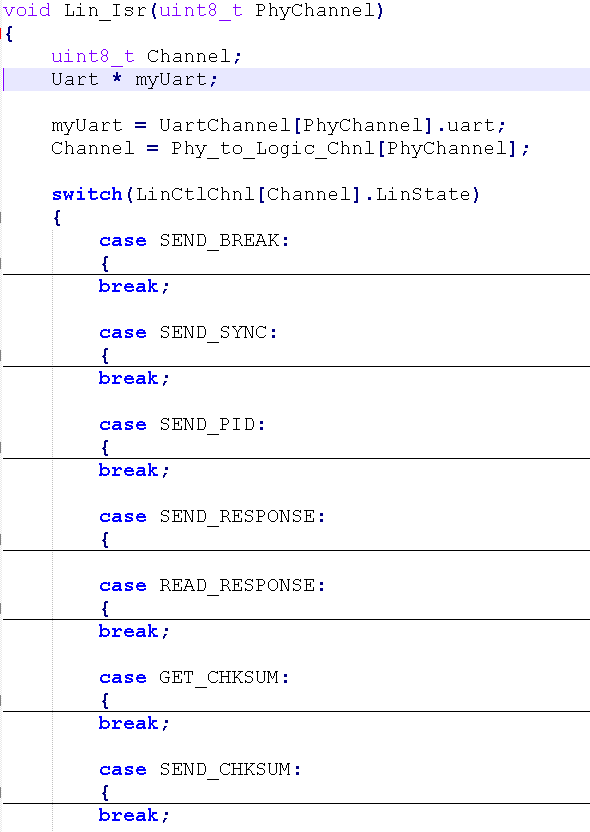


Figure 6 Lin\_Isr implementation

In Figure 6 Lin\_Isr function can be seen, there, is implemented a state machine for control LIN frame states, SEND\_BREAK; SEND\_SYNC, SEND\_PID, SEND\_RESPONSE GET\_RESPONSE, GET\_CHKSUM, READ\_CHKSUM. Those states are described in next section.

### State description

### IDLE

No transmission is done here.

### SEND\_BREAK

Break is sent here, BREAK consist in a minimum of 13 recessive bits (0), since in a normal UART driver only can be send a max of 8 recessive bits, is necessary to reduce official LIN baud rate to make 8 recessive bits look like at least 13 recessive bits (reduction is done to 5/8 of official baud rate in order to make 8 bit times look like 13 bit times), see figure 8. Baud rate update is done in this state just before send BREAK\_CMD (0x55) value.

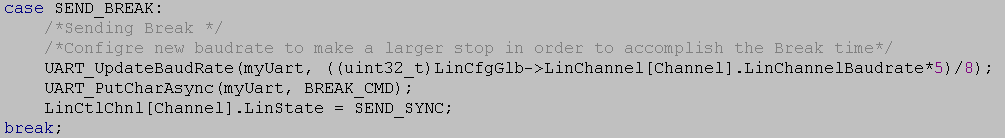


Figure 7 SEND\_BREAK

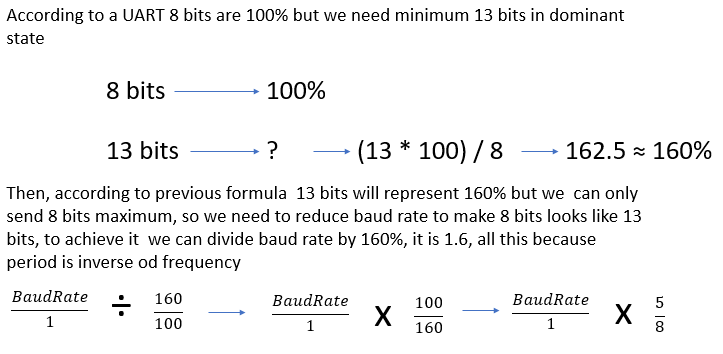


Figure 8 Reduce BaudRate Formula

### SEND\_SYNC

Here basically is needed send the 0x55 sync byte(SYNC\_CMD), but because in previous STATE baud rate was reduced, now baud rate is set to the official LIN baud rate before send SYNC\_CMD, details in figure 9.

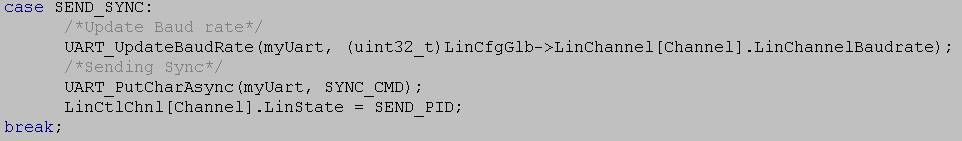


Figure 9 SEND\_SYNC

### SEND\_PID

Protected Identifiers is send here just after SEND\_SYNC finish, Pid was previously calculated. Code from Figure 10 shows how after sending it, a decision is done for determinate next state depending of current frame type, LIN\_MASTER\_RESPONSE or LIN\_SLAVE\_RESPONSE.

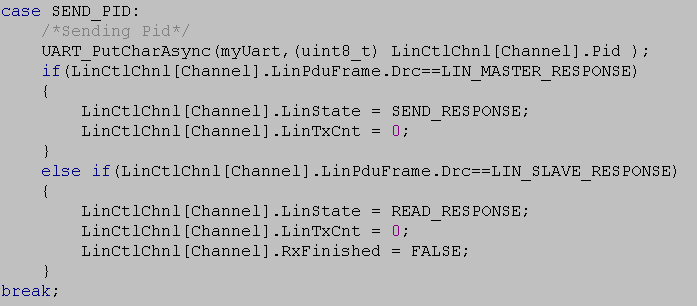


Figure 10 SEND\_PID

### SEND\_RESPONSE

For send response a Tx counter is used to send as many data as data length(Dl) indicates, once Tx counter is equal to data length next state is defined as SEND\_CHKSUM.

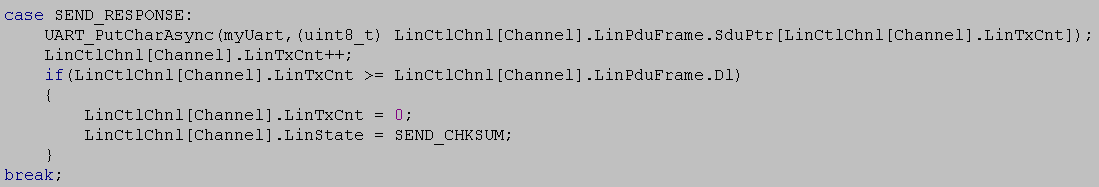


Figure 11 SEND\_RESPONSE

### SEND\_CHKSUM

As checksum value was calculated before here such value is send and next state is defined as IDLE.

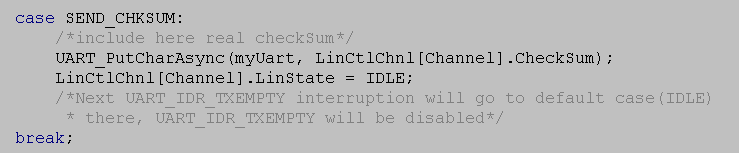


Figure 12 SEND\_CHKSUM

### READ\_RESPONSE

“READ\_RESPONSE” is executed as many times as response data length, each time this state start UART\_GetChar function is used for getting last received byte and save it in rx buffer, first time in current state is result of last TXEMPTY interrupt, this first time, TXEMPTY interrupt is disabled and RXRDY interrupt is enabled to allow state machine to enter again. After all data was received GET\_CHECKSUM state is set.

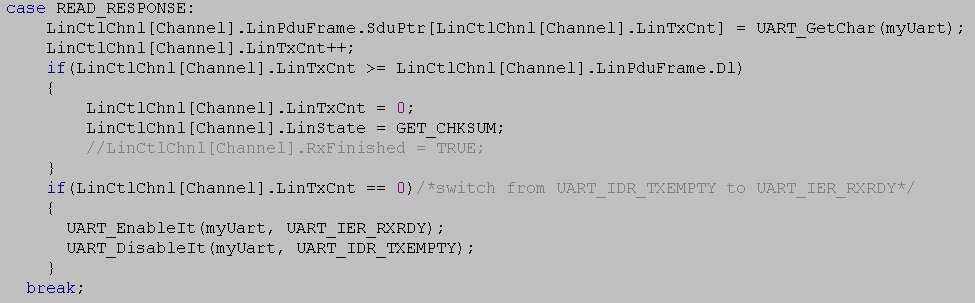


Figure 13 READ\_RESPONSE

### GET\_CHECKSUM

Since checksum is only one byte, after reading it here, RxFinished flag is set to TRUE, it helps read response interface to return the complete received data to its callers. Nest state pass to be IDLE and RXRDY interrupt is disabled.

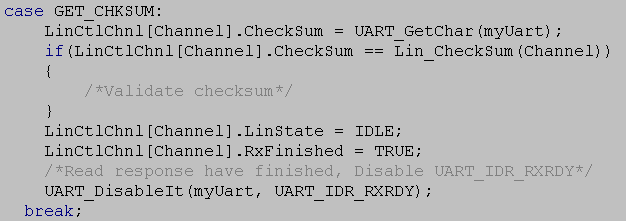


Figure 14 GET\_CHKSUM

### IDLE

IDLE state is defined as the default case of the switch.

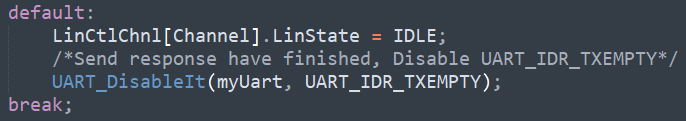


Figure 15 IDLE

# Results

The complete LIN frame standard is showed in Figure 16, the target in this work is send such frame complete.

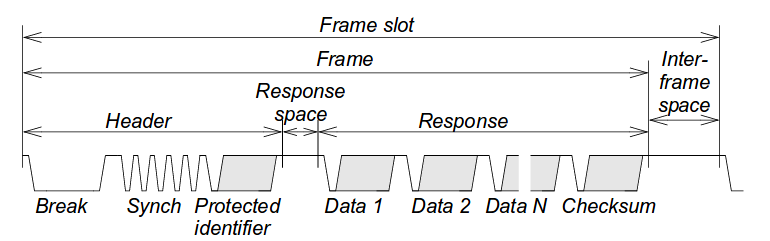


Figure 16 LIN frame

Using the oscilloscope is possible to capture Lin frame for validate that our implementation is sending the complete frame according to the standard.

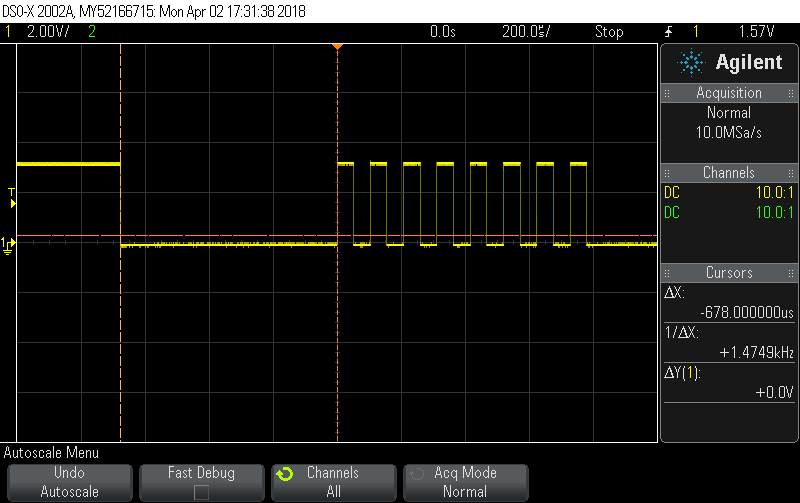


Figure 17 Header capture from Oscilloscope

In figure 17 cursors are measuring the BREAK as 678 microseconds, such measure corresponds to 13 bits in dominant state, since LIN baud rate was set to 19200, 1-bit time is 52 microseconds and 13-bit times are 676 microseconds. Bit time measure is showed in figure 18.

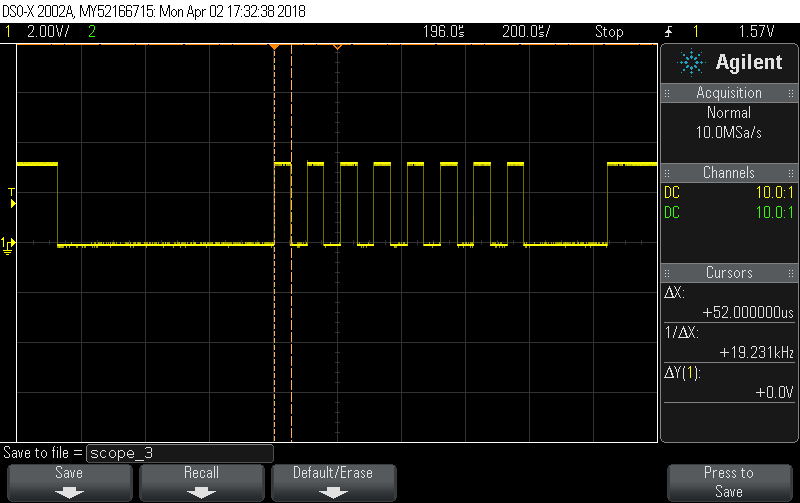


Figure 18 Bit time measure

Figure 19 shows how the header and all its parts are included in generated frame (Break, delimiter, synch byte and PID).

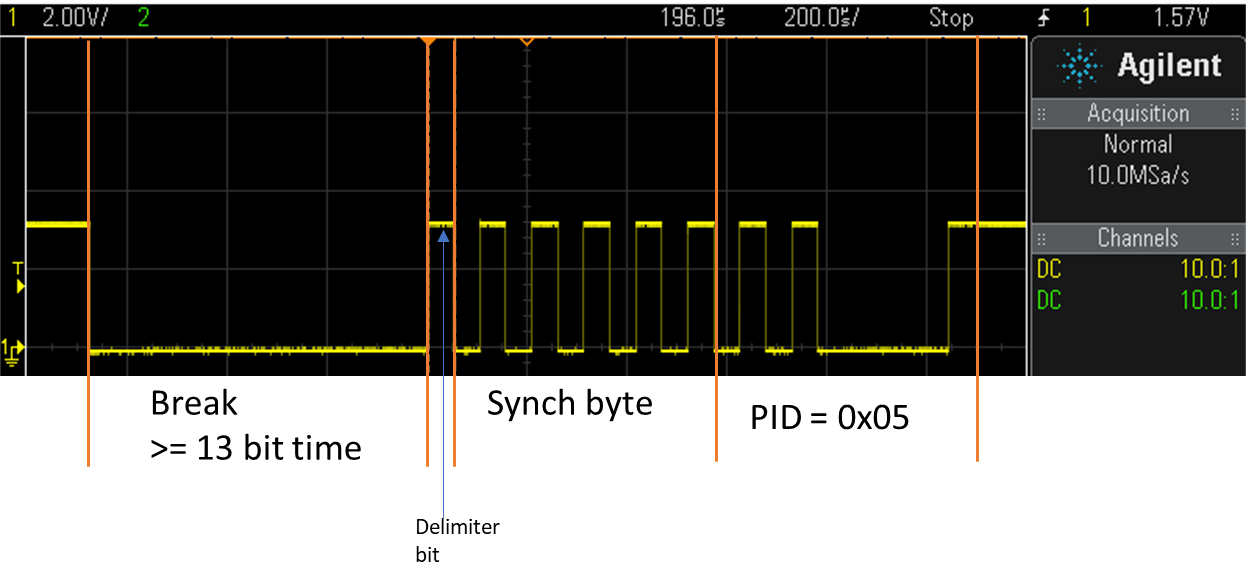


Figure 19 Details of generated Header LIN frame

In figure 20 a complete MASTER RESPONSE FRAME can be observed with all its parts.

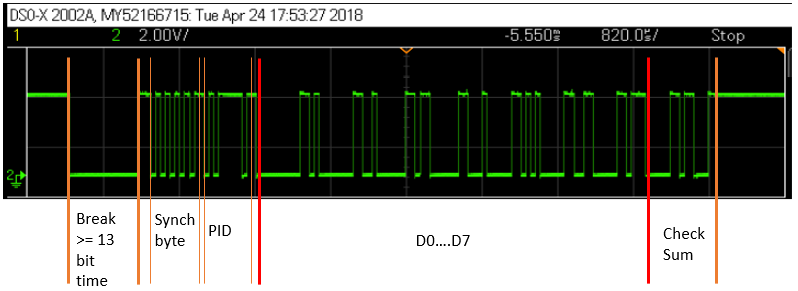


Figure 20 MASTER RESPONSE FRAME

Figure 21 shows a SLAVE RESPONSE FRAME, there, channel 1 is rx line and channel 2 is tx line. For receive a slave response a dummy node was used, this dummy node is sending data continuously so, when master node enter to READ\_SLAVE\_RESPONSE state, it will read dummy data in rx line.

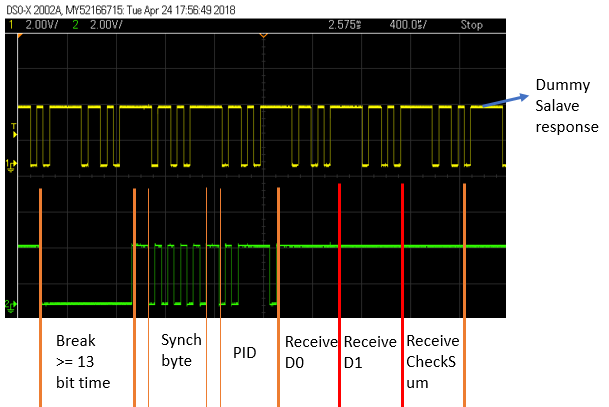


Figure 21 SLAVE RESPONSE FRAME

# Conclusions

It was demonstrated that LIN frame can be sending using an UART driver since LIN is a Serial Based protocol.

Complete LIN frame was implemented, which is sent by a LIN master node.

Using AUTOSAR architecture standards allows create software with high scalability, modularity and therefore, reusable.

It was demonstrated as well how usage of asynchronous function for send UART messages allows the CPU work in other task while the transmission is in progress.