|  |
| --- |
| 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](#_Toc511679874)

[1. Table of Figures iii](#_Toc511679875)

[2. Introduction 1](#_Toc511679876)

[2.1. Requirements 1](#_Toc511679877)

[2.1.1. Architecture 1](#_Toc511679878)

[2.1.2. Implementation 2](#_Toc511679879)

[3. Functional Description 4](#_Toc511679880)

[3.1. AUTOSAR Implementation 4](#_Toc511679881)

[3.2. LIN functions Implementation 6](#_Toc511679882)

[3.2.1. State description 8](#_Toc511679883)

[3.2.2. IDLE 8](#_Toc511679884)

[3.2.3. SEND\_BREAK 8](#_Toc511679885)

[3.2.4. SEND\_SYNC 8](#_Toc511679886)

[3.2.5. SEND\_PID 8](#_Toc511679887)

[3.2.6. SEND\_EOT 8](#_Toc511679888)

[3.2.7. SEND\_RESPONSE 8](#_Toc511679889)

[4. Results 8](#_Toc511679890)

[5. Conclusions 10](#_Toc511679891)

# **Table of Figures**

[Figure 1 AUTOSAR file structure 3](#_Toc510563993)

[Figure 2 AUTOSAR layers 4](#_Toc510563994)

[Figure 3 Lin\_Init implementation 5](#_Toc510563995)

[Figure 4 Lin\_SendFrame implementation 6](#_Toc510563996)

[Figure 5 Lin\_Isr implementation 6](#_Toc510563997)

[Figure 6 LIN frame 7](#_Toc510563998)

[Figure 7 Header capture from Oscilloscope 8](#_Toc510563999)

[Figure 8 Bit time measure 8](#_Toc510564000)

[Figure 9 Details of generated LIN frame 9](#_Toc510564001)

# 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 a MASTER RESPONSE, a SLAVE response is not part of this goal but it could be easily implemented after have the first.

## 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.

#### linf function implementation

**void Lin\_Init (uint16\_t LinBaudrate)**

- This function will configure the lower layer UART driver

- Baudrate will as per LinBaudrate parameter

- Interrupts shall be configured for each data byte to be transmitted or received

- This function shall provide to the lower layer a function callback (Lin\_Isr) to be invoked at any of the RX or TX UART interrupts

**void Lin\_SendFrame (uint8\_t LinPid)**

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

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

-This function handle the Sate machine workflow, always is called indirectly by lower layer(uart), except for start SEND\_BREAK state, in this case it is called directly from SendFrame function.

Differen stats were defined in followinf enumeration

typedef enum

{

IDLE = 0,

SEND\_BREAK,

SEND\_SYNC,

SEND\_PID,

SEND\_RESPONSE,

READ\_RESPONSE,

SEND\_CHKSUM,

SEND\_EOT

}LinStateType;

The state machine sequence is as follows

IDLE -> SEND\_BREAK -> SEND\_SYNC -> SEND\_PID -> SEN\_RESPONSE -> SEND\_CHECKSUM

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 process is about to start or in progress.

SEND\_RESPONSE: Send aster response date is about to start or in progress.

SEND\_CHECKSUM: Send checksum is about to start or in progress.

# 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.

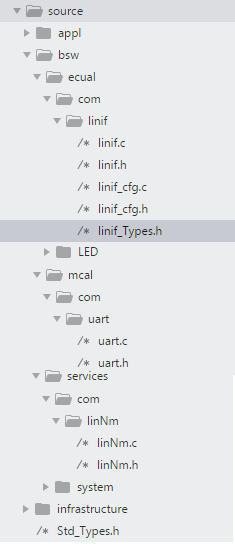


Figure 1 AUTOSAR file structure

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 located 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.

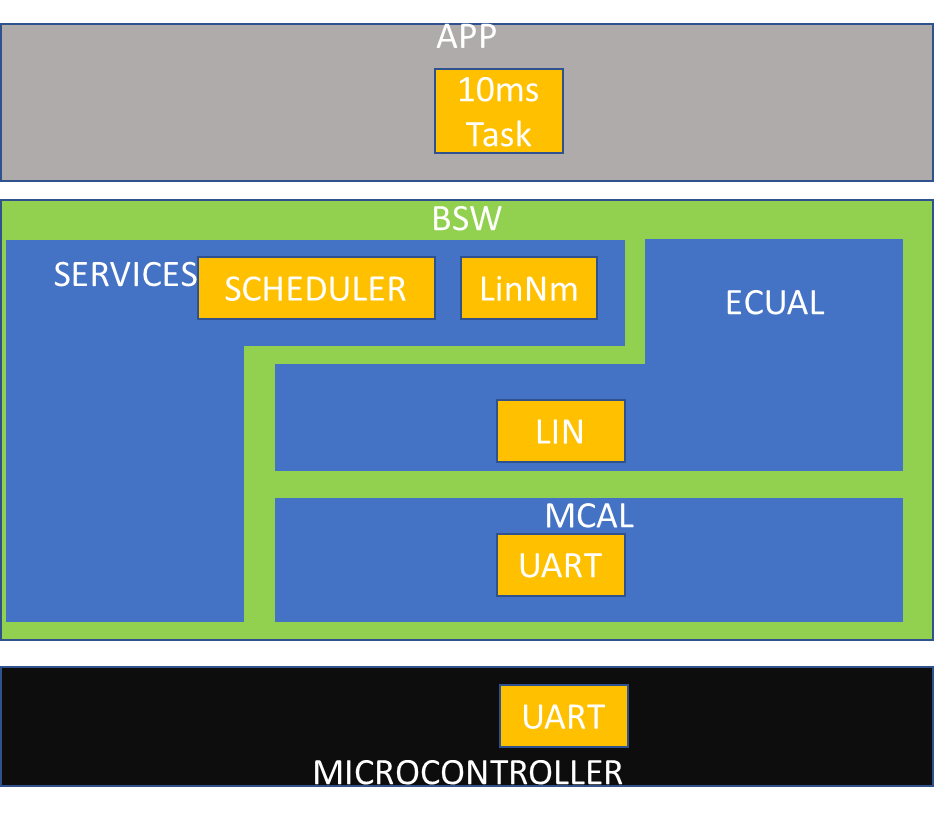


Figure 2 AUTOSAR layers

## LIN functions description

There are 3 main function in Lin to be implemented according to requirement #2, Lin\_Init, Lin\_SendFrame and Lin\_Isr,

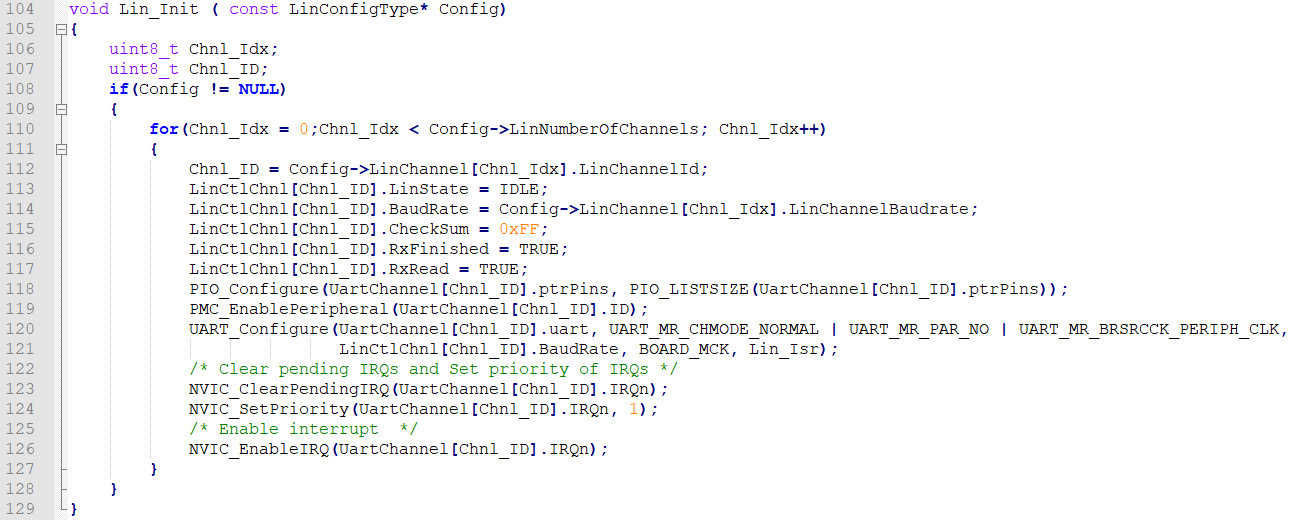


Figure 3 Lin\_Init implementation

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 is just for use one channel, but according to previous implementation it accepts as may channels are available in Hw, but not more than 255.

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 |

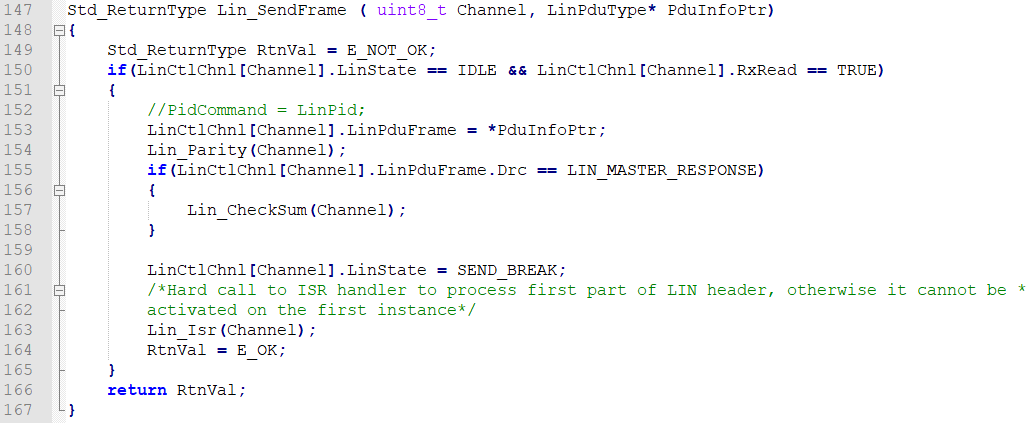


Figure 4 Lin\_SendFrame implementation

Lin\_SendFrame is set to initialize a LIN frame message transmission (see Figure 4), LIN frame transmission is implemented based in a state machine for send each part of the frame.

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

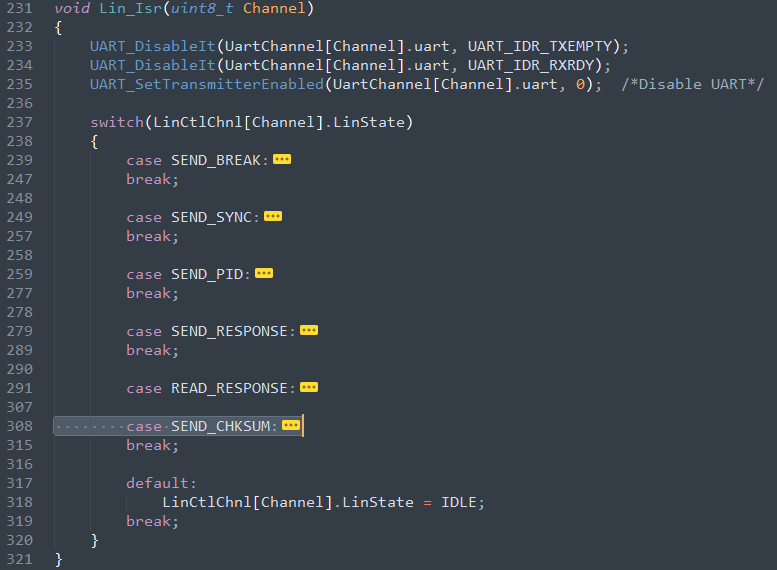


Figure 5 Lin\_Isr implementation

In Figure 5 can be seen Lin\_Isr function, there, is implemented a state machine for control LIN frame states, SEND\_BREAK; SEND\_SYNC, SEND\_PID, SEND\_EOT and SEND\_RESPONSE. Those state 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 7. Baud rate update is done in this state just before send BREAK\_CMD (0x55) value.

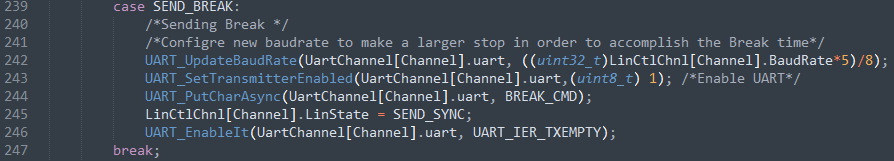


Figure 6 SEND\_BREAK

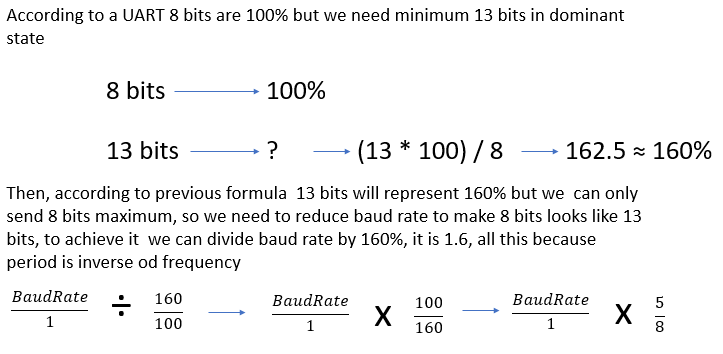


Figure 7 Reduce BaudRate Formula

### SEND\_SYNC

Here basically is needed send the 0x55 sync byte, but because in previous STATE baud rate was reduced, here baud rate is set to the official LIN baud rate before send 0x55

### SEND\_PID

Protected Identifiers is sent here just after SEND\_SYNC finish.

### SEND\_EOT

Not implemented

### SEND\_RESPONSE

Not implemented

# Results

The complete LIN frame standard is showed in Figure 8, however, target in this work is only send the header (Break, Sync and PID).

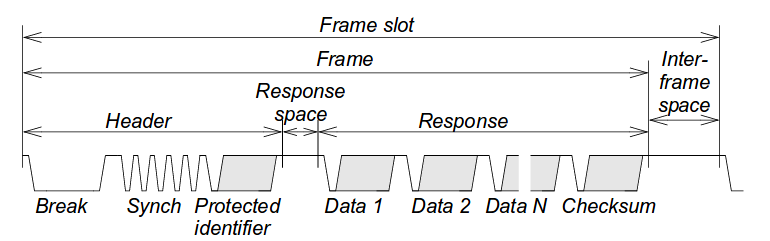


Figure 8 LIN frame

Using the oscilloscope is possible to capture Lin frame for validate that our implementation is sending the header according to the standard. Figure 7 shows the result.

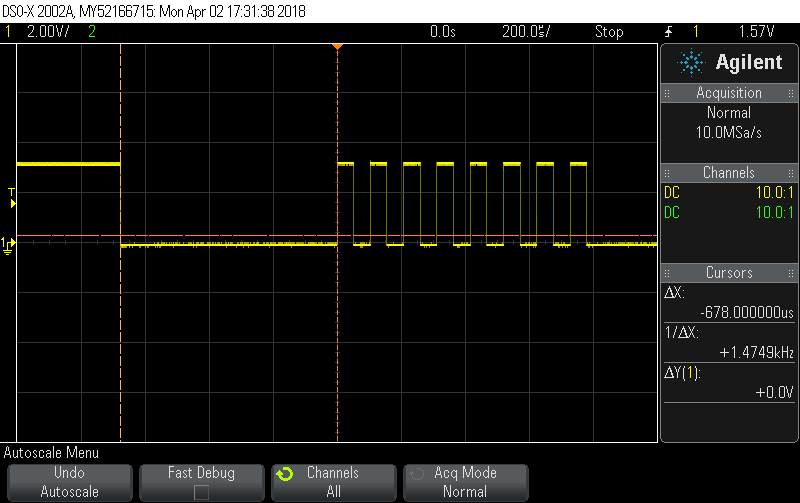


Figure 9 Header capture from Oscilloscope

In figure 9 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 8.

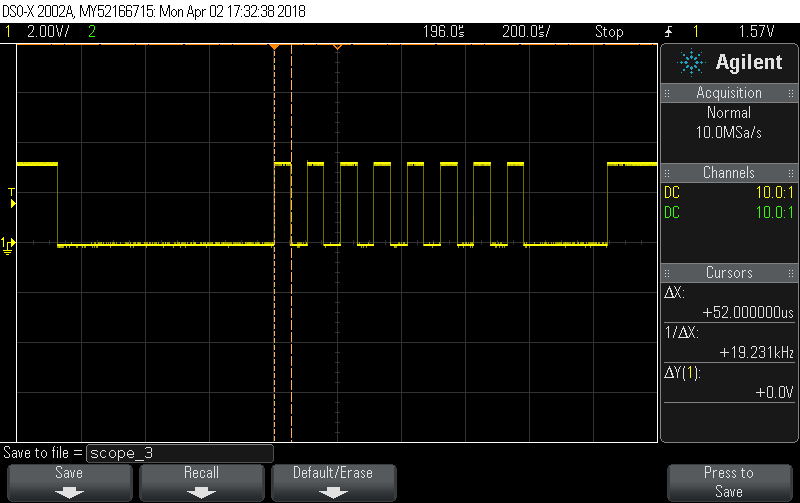


Figure 10 Bit time measure

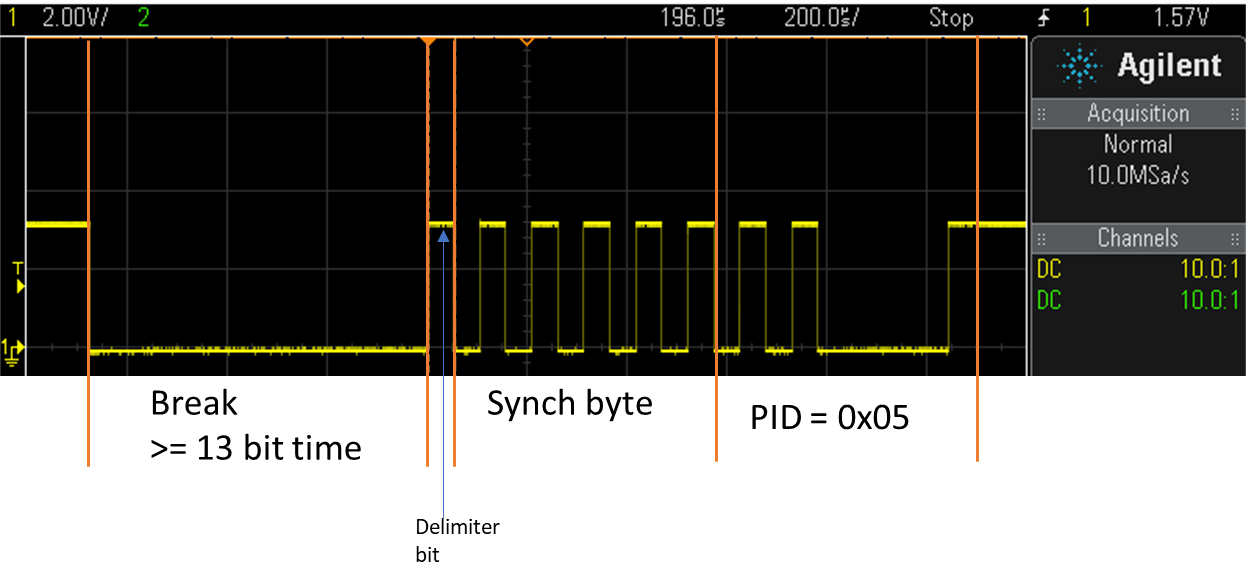
Figure 9 shows how the header and all its parts are included in generated frame(Break, delimiter, synch byte and PID).

Figure 11 Details of generated LIN frame

# Conclusions

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

So far, only the header was implemented, which is sent by a LIN master node, but this is a first step to create a complete LIN driver to be used by master and slave nodes.

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.