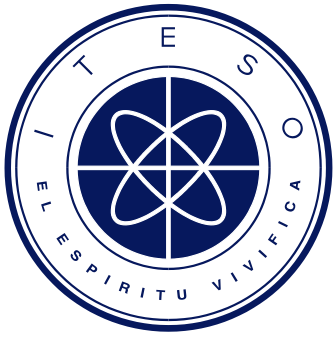
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS



CONFIGURACIÓN DE UN DRIVER LOCAL DE INTERCONEXIÓN 4.0.3 BASADO EN EL ESTÁNDAR AUTOMOTRIZ AUTOSAR

Trabajo recepcional que para obtener el diploma de

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: Jesús García Palomera

Asesor: Raúl Campos Rodríguez

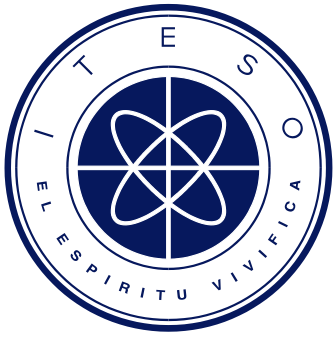
Tlaquepaque, Jalisco. Diciembre de 2016.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS



CONFIGURACIÓN DE UN DRIVER LOCAL DE INTERCONEXIÓN 4.0.3 BASADO EN EL ESTÁNDAR AUTOMOTRIZ AUTOSAR

Trabajo recepcional que para obtener el diploma de

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: Jesús García Palomera
Becario CONACYT No. 424525

Asesor: Raúl Campos Rodríguez

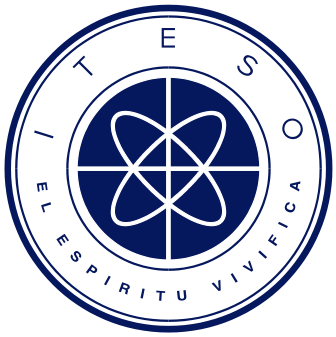
Tlaquepaque, Jalisco. diciembre de 2016.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS



CONFIGURATION OF LOCAL INTERCONNECT DRIVER 4.0.3 BASED ON AUTOSAR AUTOMOTIVE STANDARD

Final work that to obtain the diploma of

EMBEDDED SYSTEM SPECIALIST

Presents: Jesús García Palomera
CONACYT Scholarship No. 424525

Advisor: Raúl Campos Rodríguez

Tlaquepaque, Jalisco, diciembre de 2016.

AKNOWLEDGMENTS

By means of the present work I would like to thank the Dr. Raul Campos Rodriguez as well as to the professors that conform the Embedded System Program, since, its valuable experience and the obtained knowledge has allowed me to grow professionally.

Thanks to the abilities I have developed through this studies, I obtained the opportunity to occupy a new position as Embedded Software Engineer in the area of Research and Development at Continental Automotive Guadalajara company and in which I have been applying much of the knowledge that I can obtain at this postgraduate program at ITESO.

I would like to express my sincere gratitude to my advisor Dr. Raúl Campos Rodriguez for the support of achieving this work done.

I want to thank to Consejo Nacional de Ciencia y Tecnología (CONCACYT) for his support by the Scholarship No. 424525.

Specially tanks to ITESO and Continental Automotive Guadalajara for the equipment and facilities to achieve every activity developed in this program. Special thanks to all my teachers for their patience, motivation, and immense knowledge I received from them.

RESUMEN

Durante el presente trabajo se ilustra el análisis y el diseño de la implementación de la funcionalidad del driver de LIN *Lin_GetStatus(),* cuyo objetivo de dicha función es detectar los posibles estados de comunicación del bus de LIN, de acuerdo a lo especificado en el estándar automotriz de AUTOSAR.

La presente implementación del driver de LIN se basa en las características del microcontrolador NXP MC9S12XEP100 en la placa de experimentación DEMO9S12XEP100 la cual contiene un microcontrolador de 16-bits dual core RISC ECU de grado automotriz.

ABSTRACT

This work is intended to illustrate the analysis and design needed to implement the LIN driver functionality Lin_GetStatus(). The objective of such a functionality is to detect the possible states of the LIN bus as specified by the AUTOSAR standard.

The implementation of the driver developed in this work has been developed in the NXP microcontroller MC9S12XEP100 on the DEMO9S12XEP100 experimentation board. The microcontroller is a 16-bits dual core RISC ECU of automotive grade.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

Cross-compilation	Process whereby compilation is done on a PC (host) with the purpose to be executed on other system, called target (target). When cross-compilation is done, executables for another system are produced. This happens when the target does not have the toolchain native compilation or when the host is faster and has better resources (CPU, memory, etc.).
SDK	Software Development Kit.
X11	X Window System.
RTOS	Real-Time Operating System.
AUTOSAR	Automotive Open System Architecture.
LIN	Local Interconnect Network.
TC	Transmission complete.
TE	Transmitter Enable.
TIE	Transmitter Interrupt Enable.
TCIE	Transmission Complete Interrupt Enable.
TDRE	Transmit Data Register Empty.
PE	Parity Enable.
PT	Parity Type.
NF	Noise Flag.
PF	Parity Error.
BERRV	Bit Error Value.
BERRIF	Bit Error Interrupt Flag.
BERRIE	Bit Error Interrupt Enable.
BERRM	Bit Error Mode.
RIE	Receiver Full Interrupt Enable.
RE	Receiver Enable.
RDRF	Receive Data Register Full Flag.
OR	Overrun.
FE	Framing Error.
ILT	Idle Line Type.
ILIE	Idle Line Interrupt Enable.
RWU	Receiver Wakeup bit.
IDLE	Idle Line Flag.
SCISWAI	SCI Stop in Wait Mode.

1. INTRODUCTION

Abstract: *This chapter briefly presents the background of the object of study, problem definition and justification.*

1.1. AUTOSAR

AUTOSAR is a worldwide development partnership of vehicle manufacturers, suppliers and other companies from the electronics, semiconductor, and software industry. Its objective is to define and to stablish the way for innovative electronics systems and creates one common standard for automotive industry easing the exchange and update of software and hardware over the service life of the vehicle.

Local Interconnect Network (LIN) is a serial communication network protocol widely used in the intercommunication networks inside the vehicles, including utilitarian cars, passenger buses, and cargo trucks, to mention some.

The principal advantages of the LIN protocol are that it is cheap and easy to implement in noisy environments such as those present in automotive applications. In the 1990's the LIN protocol was adopted by a great number of big automotive manufactures, such as BMW, VW, Audi, Volvo, and Mercedes-Benz, for example.

The LIN protocol defines a set of functionalities that has to be satisfied for a suitable communication among the ECU interconnected in a communication network based on this protocol.

1.2. Objectives

The main objective of this work is to implement the functionality of the LIN protocol to support the ability to get the status of the communication bus.

- *Lin_GetStatus()*. To implement this function to detect the possible states of the LIN bus as defined by the AUTOSAR standard

- *DEMO9S12XEP100:* To use the development boar for the MC9S12XEP100 microcontroller from Freescale (now Qualcomm) as implementation framework.

- *Testing*. To test the implementation to verify the functionality of the Lin_GetStatus() functionality.

2. BACKGROUNG

Abstract: *This chapter briefly reviews the background of the functionality to be developed in this project.*

2.1. AUTOSAR FUNCTION SPECIFICATION

The functionality *Lin_GetStatus()* that is defined in AUTOSAR LIN driver 4.0.3 is used to know the current status of the frame transmission request by the LIN channel, the main features of the function are described at following chart:

Table 1. AUTOSAR specification of Lin_GetStatus() function

Service name:	Lin_GetStatus	

Syntax:	`Lin_StatusType Lin_GetStatus(`
	`  uint8 Channel,`
	`  uint8** Lin_SduPtr`
	`)`	

Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Channel	LIN channel to be checked
Parameters (inout):	None	
Parameters (out):	Lin_SduPtr	Pointer to pointer to a shadow buffer or memory mapped LIN Hardware receive buffer where the current SDU is stored.

2.2. AUTOSAR DEFINITION OF AVAILABLE RETURN VALUES

The range of operational values that are defined as enumeration data type are available in order to be returned by the API Lin_GetStatus() according to the status of the LIN bus.

The Table 2 summarizes the values that are available for the return information of the Lin_GetStatus() functionality.

Table 2. AUTOSAR definition of Lin_GetStatus() return values

Data Type:	Enumeration.
Name:	Lin_StatusType.
Range:	
Value	**Description**
LIN_NOT_OK	Development or production error occurred
LIN_TX_OK	Successful transmission.
LIN_TX_BUSY	Ongoing transmission (Header or Response).
LIN_TX_HEADER_ERROR	Erroneous header transmission such as: - Mismatch between sent and read back data - Identifier parity error or - Physical bus error.
LIN_TX_ERROR	LIN frame operation return value. Erroneous response transmission such as: - Mismatch between sent and read back data. - Physical bus error.
LIN_RX_OK	Reception of correct response.
LIN_RX_BUSY	Ongoing reception: at least one response byte has been received, but the checksum byte has not been received.
LIN_RX_ERROR	Erroneous response reception such as: - Framing error - Overrun error - Checksum error or - Short response.
LIN_RX_NO_RESPONSE	No response byte has been received so far.
LIN_OPERATIONAL	LIN channel state return value. Normal operation; the related LIN channel is ready to transmit next header. No data from previous frame available. (e.g. after initialization).

LIN_CH_SLEEP	LIN channel state operation; In this state wake-up detection from slave nodes is enabled.

3. DESIGN AND IMPLEMENTATION

Abstract: *This chapter details the design and implementation of the functionality of the AUTOSAR API Lin_GetStatus().*

3.1. MICROCONTROLLER PERIPHERIAL SUPPORT

The MC9S12XEP100 microcontroller registers needed to detect the status of the LIN bus based on AUTOSAR Lin_GetSatatus() definitions are presented as follows:

3.1.1. LIN_TX_OK

The function *Lin_GetStatus()* shall return LIN_TX_OK value, when:

- The Master response frame and its Header are sent successfully.
- The Slave response frame and its header are sent successfully.

Table 3. SCI Control Register 2 (SCICR2).

Field	Description
6 TCI	**Transmission Complete Interrupt Enable Bit**—TCIE enables the transmission complete flag, TC, to generate interrupt requests. 0 TC interrupt requests disabled. 1 TC interrupt requests enabled.
3 TE	**Transmitter Enable Bit** — TE enables the SCI transmitter and configures the TXD pin as being controlled by the SCI. The TE bit can be used to queue an idle preamble. 0 Transmitter disabled. 1 Transmitter enabled.

Table 4. SCI Status Register 1 (SCISR1).

Field	Description
6 TC	**Transmit Complete Flag**—TC is set low when there is a transmission in progress or when a preamble or break character is loaded. TC is set high when the TDRE flag is set and no data, preamble, or break character is being transmitted.When TC is set, the TXD pin becomes idle (logic 1). Clear TC by reading SCI status register 1 (SCISR1) with TC set and then writing to SCI data register low (SCIDRL). TC is cleared automatically when data, preamble, or break is queued and ready to be sent. TC is cleared in the event of a simultaneous set and clear of the TC flag (transmission not complete). 0 Transmission in progress. 1 No transmission in progress.

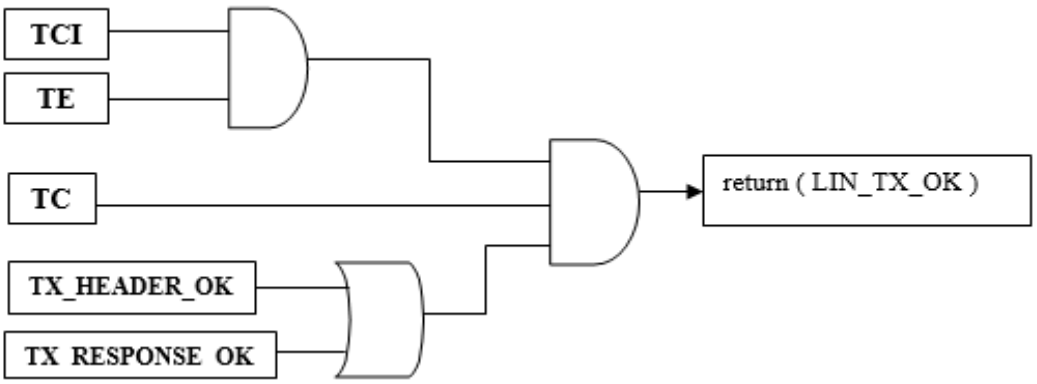


Figure 1. LIN_TX_OK status Block Diagram.

3.1.2. LIN_TX_BUSY

The function *Lin_GetStatus()* shall return LIN_TX_BUSY, when exist an ongoing transmission on the buffer (header or response).

Table 5. SCI Control Register 2 (SCICR2).

Field	Description
7 TIE	**Transmitter Interrupt Enable Bit** — TIE enables the transmit data register empty flag, TDRE, to generate interrupt requests. 0 TDRE interrupt requests disabled 1 TDRE interrupt requests enabled.
3 TE	**Transmitter Enable Bit** — TE enables the SCI transmitter and configures the TXD pin as being controlled by the SCI. The TE bit can be used to queue an idle preamble. 0 Transmitter disabled 1 Transmitter enabled

Table 6. SCI Status Register 1 (SCISR1).

Field	Description
7 TDRE	**Transmit Data Register Empty Flag** — TDRE is set when the transmit shift register receives a byte from the SCI data register. When TDRE is 1, the transmit data register (SCIDRH/L) is empty and can receive a new value to transmit.Clear TDRE by reading SCI status register 1 (SCISR1), with TDRE set and then writing to SCI data register low (SCIDRL). 0 No byte transferred to transmit shift register. 1 Byte transferred to transmit shift register; transmit data register empty.

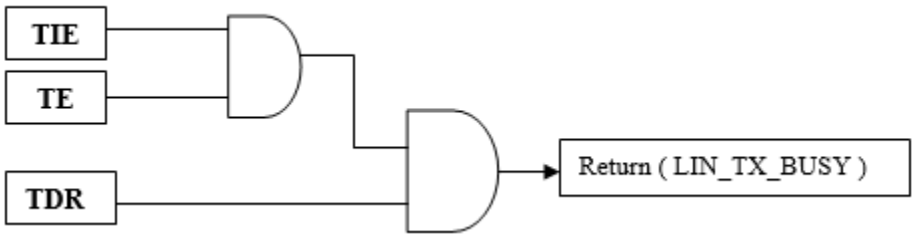


Figure 2. LIN_TX_BUSY status Block Diagram.

3.1.3. TX_HEADER_ERROR

The TX_HEADER_ERROR status shall be returned by the *Lin_GetSatatus()* function if an erroneous header transmission is detected such as:

- Mismatch between sent and read back data.
- Identifier parity error or
- Physical bus error

Table 7. SCI Control Register 1 (SCICR1).

Field	Description
1 PE	**Parity Enable Bit** — PE enables the parity function. When enabled, the parity function inserts a parity bit in the most significant bit position. 0 Parity function disabled 1 Parity function enabled

0 PT	**Parity Type Bit**—PT determines whether the SCI generates and checks for even parity or odd parity. With even parity, an even number of 1s clears the parity bit and an odd number of 1s sets the parity bit. With odd parity, an odd number of 1s clears the parity bit and an even number of 1s sets the parity bit. 0 Even parity 1 Odd parity

Table 8. SCI Status Register 1 (SCISR1).

Field	Description
2 NF	**Noise Flag**—NF is set when the SCI detects noise on the receiver input. NF bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. Clear NF by reading SCI status register 1(SCISR1), and then reading SCI data register low (SCIDRL). 0 No noise 1 Noise
0 PF	**Parity Error Flag** — PF is set when the parity enable bit (PE) is set and the parity of the received data does not match the parity type bit (PT). PF bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. Clear PF by reading SCI status register 1 (SCISR1), and then reading SCI data register low (SCIDRL). 0 No parity error 1 Parity error

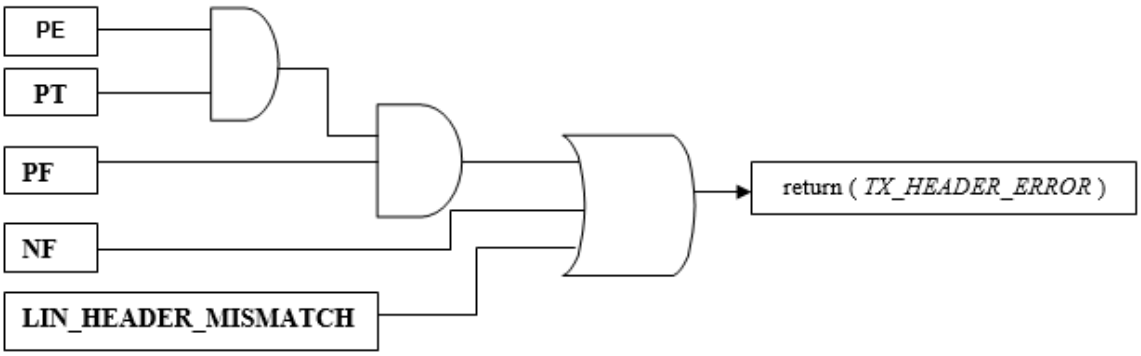


Figure 3. TX_HEADER_ERROR status Block Diagram.

3.1.4. LIN_TX_ERROR

The function *Lin_GetSatatus()* shall return LIN_TX_ERROR value, when:

- Mismatch between sent and read back data
- Physical bus error

Table 9. SCI Alternative Status Register 1 (SCIASR1).

Field	Description
2 BERRV	**Bit Error Value** — BERRV reflects the state of the RXD input when the bit error detect circuitry is enabled and a mismatch to the expected value happened. The value is only meaningful, if BERRIF = 1. 0 A low input was sampled, when a high was expected 1 A high input reassembled, when a low was expected

1 BERRIF	**Bit Error Interrupt Flag** — BERRIF is asserted, when the bit error detect circuitry is enabled and if the value sampled at the RXD input does not match the transmitted value. If the BERRIE interrupt enable bit is set an interrupt will be generated. The BERRIF bit is cleared by writing a "1" to it. 0 No mismatch detected 1 A mismatch has occurred

Table 10. SCI Alternative Control Register 1 (SCIACR1).

Field	Description
1 BERRIE	**Bit Error Interrupt Enable** — BERRIE enables the bit error interrupt flag, BERRIF, to generate interrupt requests. 0 BERRIF interrupt requests disabled 1 BERRIF interrupt requests enabled

Table 11. SCI Alternative Control Register 1 (SCIACR1).

Field	Description
2:1 BERRM[1:0]	**Bit Error Mode** — Those two bits determines the functionality of the bit error detect feature. See following table.

BERRM1	BERRM0	Function
0	0	Bit error detect circuit is disabled
0	1	Receive input sampling occurs during the 9th time tick of a transmitted bit.
1	0	Receive input sampling occurs during the 13th time tick of a transmitted bit
1	1	Reserved.

Table 12. SCI Status Register 1 (SCISR1).

Field	Description
2 NF	**Noise Flag**—NF is set when the SCI detects noise on the receiver input. NF bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. Clear NF by reading SCI status register 1(SCISR1), and then reading SCI data register low (SCIDRL). 0 No noise 1 Noise

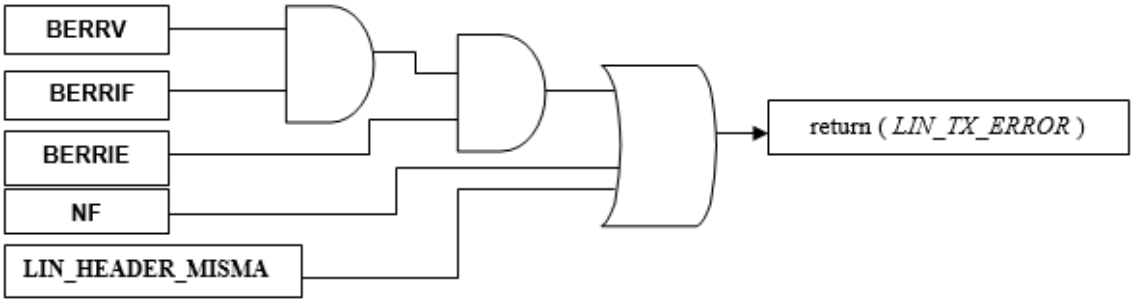


Figure 4. LIN_TX_ERROR status Block Diagram.

3.1.5. LIN_RX_OK

If a correct response is detected, the function Lin_GetStatus() shall return LIN_RX_OK status.

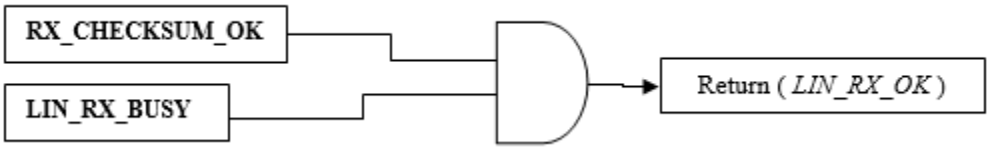


Figure 5. LIN_RX_OK status Block Diagram.

3.1.6. LIN_RX_BUSY

The state LIN_RX_BUSY shall be returned when an ongoing reception is detected and at least one response byte has been received, but the checksum byte has not been received yet.

Table 13. SCI Control Register 2 (SCICR2).

Field	Description
5 RIE	**Receiver Full Interrupt Enable Bit** — RIE enables the receive data register full flag, RDRF, or the overrun flag, OR, to generate interrupt requests. 0 RDRF and OR interrupt requests disabled 1 RDRF and OR interrupt requests enabled
2 RE	**Receiver Enable Bit —** RE enables the SCI receiver. 0 Receiver disabled 1 Receiver enabled

Table 14. SCI Status Register 1 (SCISR1).

Field	Description
5 RDRF	**Receive Data Register Full Flag** — RDRF is set when the data in the receive shift register transfers to the SCI data register. Clear RDRF by reading SCI status register 1 (SCISR1) with RDRF set and then reading SCI data register low (SCIDRL). 0 Data not available in SCI data register 1 Received data available in SCI data register

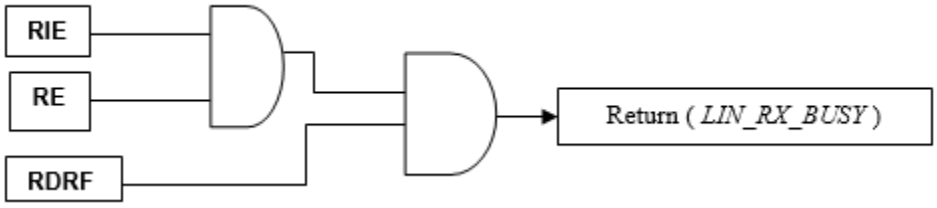


Figure 6. LIN_RX_BUSY status Block Diagram.

3.1.7. LIN_RX_ERROR

The value LIN_RX_ERROR shall be returned in case of an erroneous response reception such as:

- Framing error
- Overrun error
- Checksum error or
- Short response

Table 15. SCI Control Register 2 (SCICR2).

Field	Description
2 RE	**Receiver Enable Bit** — RE enables the SCI receiver. 0 Receiver disabled 1 Receiver enabled

Table 16. SCI Status Register 1 (SCISR1).

Field	Description
3 OR	**Overrun Flag** — OR is set when software fails to read the SCI data register before the receive shift register receives the next frame. The OR bit is set immediately after the stop bit has been completely received for the second frame. The data in the shift register is lost, but the data already in the SCI data registers is not affected. Clear OR by reading SCI status register 1 (SCISR1) with OR set and then reading SCI data register low (SCIDRL). 0 No overrun 1 Overrun **Note:** OR flag may read back as set when RDRF flag is clear. This may happen if the following sequence of events occurs: 1. After the first frame is received, read status register SCISR1 (returns RDRF set and OR flag clear); 2. Receive second frame without reading the first frame in the data register (the second frame is not received and OR flag is set); 3. Read data register SCIDRL (returns first frame and clears RDRF flag in the status register); 4. Read status register SCISR1 (returns RDRF clear and OR set). Event 3 may be at exactly the same time as event 2 or any time after. When this happens, a dummy SCIDRL read following event 4 will be required to clear the OR flag if further frames are to be received.

1 FE	**Framing Error Flag** — FE is set when a logic 0 is accepted as the stop bit. FE bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. FE inhibits further data reception until it is cleared. Clear FE by reading SCI status register 1 (SCISR1) with FE set and then reading the SCI data register low (SCIDRL). 0 No framing error 1 Framing error

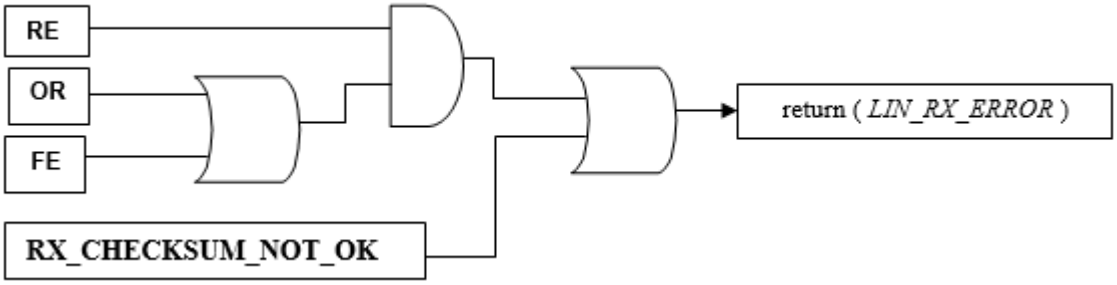


Figure 7. LIN_RX_ERROR status Block Diagram.

3.1.8. LIN_RX_NO_RESPONSE

If a response byte has not been received so far after a valid header the Lin_GetStatus() function shall return LIN_RX_NO_RESPONSE.

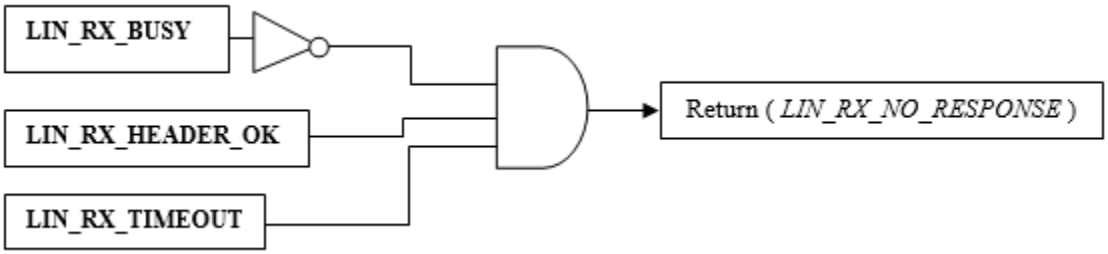


Figure 8. LIN_RX_NO_RESPONSE status Block Diagram.

3.1.9. LIN_RX_NO_RESPONSE

The LIN_OPERATIONAL status is returned when related LIN channel is just initialized or waked up from the LIN_CH_SLEEP and no data has been sent.

Table 17. SCI Control Register 1 (SCICR1).

Field	Description
3 WAKE	**Wakeup Condition Bit—**WAKE determines which condition wakes up the SCI: a logic 1 (address mark) in the most significant bit position of a received data character or an idle condition on the RXD pin. 0 Idle line wakeup 1 Address mark wakeup

Table 18. SCI Control Register 1 (SCICR1).

Field	Description
2 ILT	**Idle Line Type Bit** — ILT determines when the receiver starts counting logic 1s as idle character bits. The counting begins either after the start bit or after the stop bit. If the count begins after the start bit, then a string of logic 1s preceding the stop bit may cause false recognition of an idle character. Beginning the count after the stop bit avoids false idle character recognition, but requires properly synchronized transmissions. 0 Idle character bit count begins after start biT 1 Idle character bit count begins after stop bit

Table 19. SCI Control Register 2 (SCICR2).

Field	Description
4 ILIE	**Idle Line Interrupt Enable Bit** — ILIE enables the idle line flag, IDLE, to generate interrupt requests. 0 IDLE interrupt requests disabled 1 IDLE interrupt requests enabled
1 RWU	**Receiver Wakeup Bit** — Standby state 0 Normal operation. 1 RWU enables the wakeup function and inhibits

Table 20. SCI Status Register 1 (SCISR1).

Field	Description
4 IDLE	**Idle Line Flag**—IDLE is set when 10 consecutive logic 1s (if M = 0) or 11 consecutive logic 1s (if M =1) appear on the receiver input. Once the IDLE flag is cleared, a valid frame must again set the RDRF flag before an idle condition can set the IDLE flag.Clear IDLE by reading SCI status register 1 (SCISR1) with IDLE set and then reading SCI data register low (SCIDRL). 0 Receiver input is either active now or has never become active since the IDLE flag was last cleared 1 Receiver input has become idle Note: When the receiver wakeup bit (RWU) is set, an idle line condition does not set the IDLE flag.

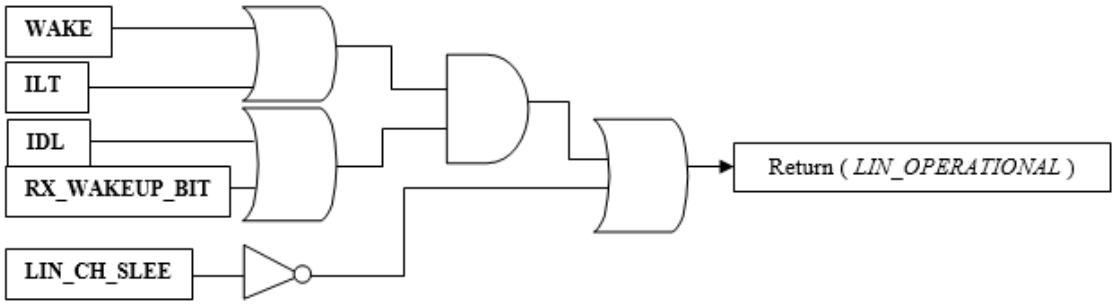


Figure 9. LIN_RX_OPERATIONA status Block Diagram.

3.1.10. LIN_CH_SLEEP

In this state wake-up detection from slave nodes is enabled. The LIN hardware is into a low power mode if the hardware provides such a mode.

Table 21. SCI Control Register 1 (SCICR1).

Field	Description
6 SCISWAI	**SCI Stop in Wait Mode Bit** — SCISWAI disables the SCI in wait mode. 0 SCI enabled in wait mode 1 SCI disabled in wait mode

4. TEST AND RESULTS

Abstract: *This chapter reports some of the test and results of the implementation of the AUTOSAR API Lin_GetStatus() functionality.*

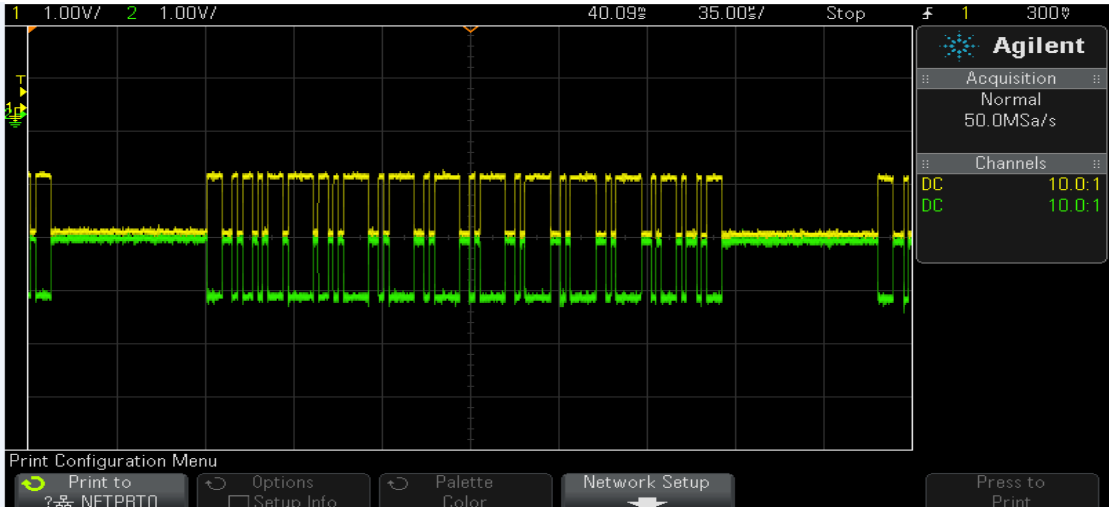
4.1. Experiment and Results

During the first part of this implementation, a single node was activated and its electrical activity was recorded using an oscilloscope in order to observe the Acknowledge slot behavior of the Communication bus and TX signals. Conclusions of each plot obtained are enlisted below:

4.1.1 ACKNOWLEDGE ERROR ON BUS

Can bus lines Low and High were attached to channel 1 and channel 2 of the oscilloscope correspondingly and the oscilograme obtained is shown in figure 1, observing that behavior of the Communication bus corresponds to a standard data frame:

As we can see on figure 1, the Acknowledge slot was set as recessive bit, which means that an acknowledged error was detected because of the inexistence of receiver nodes enabled.



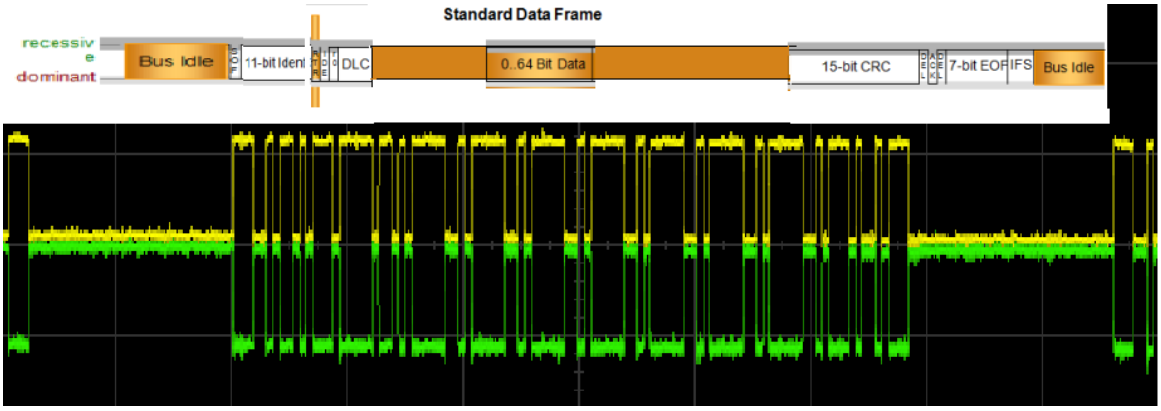


Figure 10. - Electrical Activity of Communication bus with a single node.

Then CAN_L and TX signals were connected to the oscilloscope with only one node attached to the bus, the plot obtained shows the correspondence between TX signal and BUS CAN line, which shows the acknowledge error mentioned above.

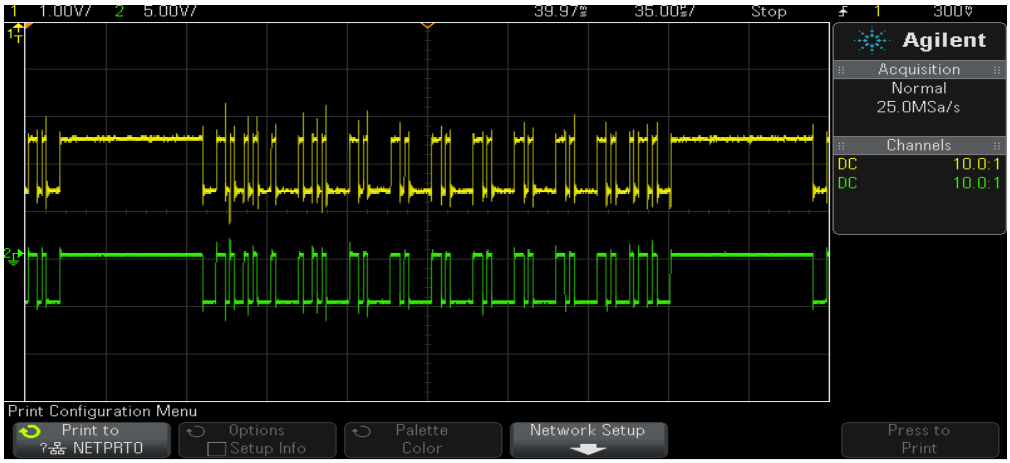


Figure 11. - Electrical CAN Activity of Controller TX signal vs CAN Bus Low with a single node.

4.1.2 **FRAME ACKNOWLEDGE ON BUS CAN**

A second node was connected to the can bus, and the following oscillogram shows the acknowledge bit as dominant state, which means that a frame was recognized by a receiver node.

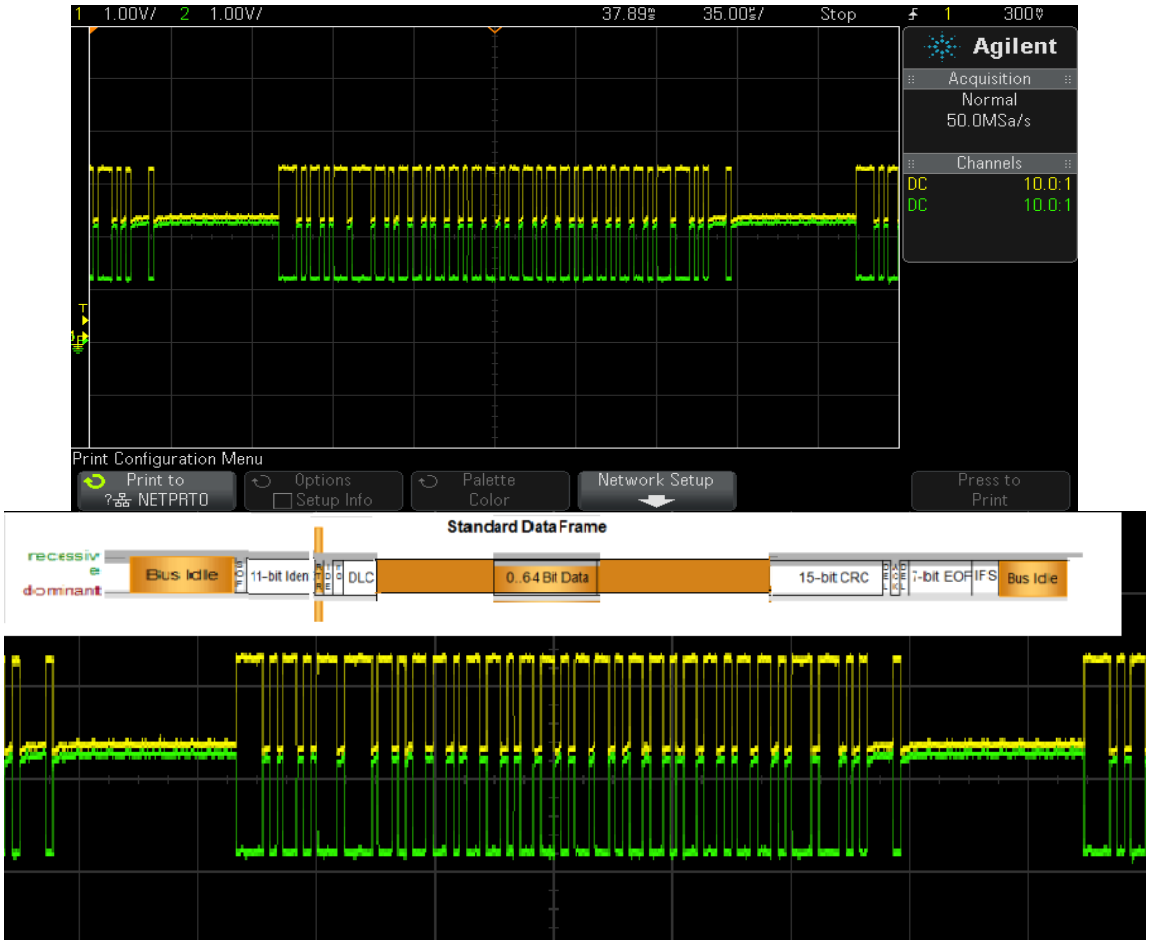


Figure 12.- Electrical Activity of CAN bus with two nodes attached.

The following plot shows the TX acknowledged signal (Channel 1) answered from the receiver node to the frame sent by the TX node (Channel 2); additionally we can appreciate the correspondence of acknowledged bit fields between frames.

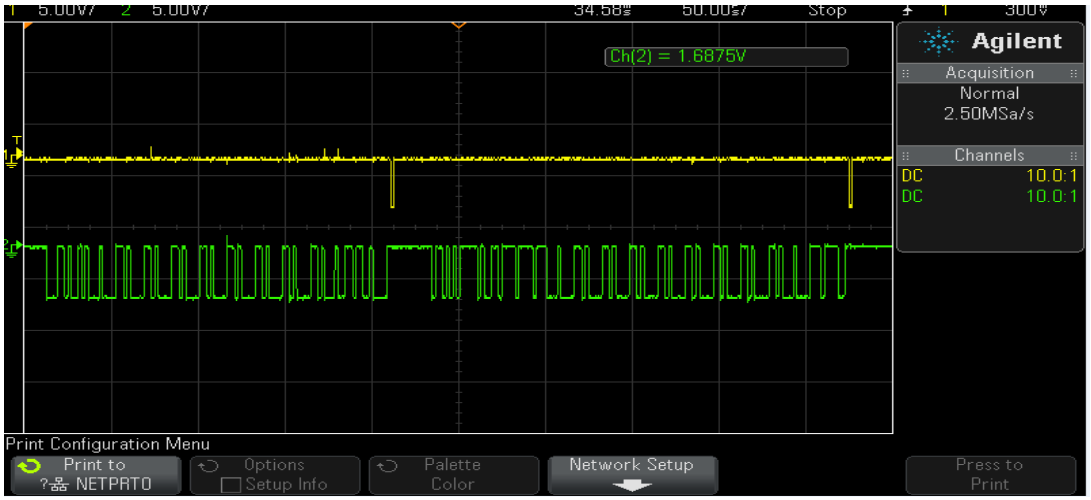


Figure 13.- Electrical CAN Activity of Controllers TX signals with two nodes connected.

4.1.3 AUTOMATIC RECOVERY MODE

The following code was implemented within cnf_mscan.h and cnf_mscan.c files in order to allow a new configuration parameter in order to implement bus-off recovery strategy:

- Declaration of available bus-off recovery modes.

```
65  typedef enum
66  {
67      BUS_OFF_USER_RECOVERY,              /**< Bus off user recovery */
68      BUS_OFF_AUTOMATIC_RECOVERY          /**< Bus off automatic recovery */
69  }tMSCAN_BusOffRecoveryMode;
70
```

```
96  typedef struct
97  {
98      UINT32                          baudrate;       /**< Baudrate */
99      const tMSCAN_RxHWObjectConfig  *rx_hwObj_cfg;   /**< pointer to static Rx hw objects configurat
100     const tMSCAN_TxHwFifoConfig    *tx_hw_fifo_cfg; /**< pointer to static Tx fifo configuration */
101     enum tMSCAN_Device              device;         /**< Device ID */
102     enum tMSCAN_AccFilterModeCfg    filter_cfg;     /**< Rx filter acceptance mode configuration */
103     enum tMSCAN_BusOffRecoveryMode  BusOffMode;     /**< Bus Off recovery mode  */
104     UINT8                           nr_of_rx_hwOb;  /**< number of rx hw objects being configured *
105     UINT8                           rx_buffer_depth;/**< software buffer depth for each rx hw filte
106     UINT8                           tx_fifo_depth;  /**< software queue depth for transmission purp
107 }tMSCAN_DeviceConfig;
```

- Declaration of bus-off recovery mode:

```
/**< Configuration of MSCAN A device */
const tMSCAN_DeviceConfig CAN_device_cfg[] =
{
    {
        (UINT32)CAN_BAUDRATE_500Kbps,       /**< Baudrate */
        &MSCAN_A_rx_msg_cfg[0],             /**< pointer to static Rx hw objects configuration */
        &MSCAN_A_tx_msg_cfg,                /**< pointer to static Tx fifo configuration */
        MSCAN_A,                            /**< Device ID */
        MSCAN_ACC_FILTERS_8_BIT_MODE,       /**< Rx filter acceptance mode configuration */
        BUS_OFF_AUTOMATIC_RECOVERY,         /**< Bus Off recovery mode  */
        sizeof(MSCAN_A_rx_msg_cfg)/sizeof(MSCAN_A_rx_msg_cfg[0]),    /**< number of rx hw objects being configured */
        CAN_RX_BUFFER_DEPTH,                /**< software buffer depth for all the rx FIFO */
        CAN_TX_QUEUE_DEPTH,                 /**< software queue depth for transmission purposes */
    },
};
```

- The next function is in charge of initialize the bus-off recovery mode selected and it will be executed during the system initialization in vfnCAN_Init() function.

```
 * \brief    MSCAN Bus Off recovery mode function
 * \author
 * \param    const tMSCAN_DeviceConfig * controller_cfg
 * \return   void
 */

void vfnCAN_BusOffRecovery(const tMSCAN_DeviceConfig * controller_cfg)
{
  if (controller_cfg->BusOffMode == BUS_OFF_USER_RECOVERY)
  {
  /*Configure Bus-Off User Recovery Mode */
    CAN0CTL1_BORM =1;
  }
  else
    if (controller_cfg->BusOffMode == BUS_OFF_AUTOMATIC_RECOVERY)
    {
    /*Configure Bus-Off Aputomatic Recovery Mode */
    CAN0CTL1_BORM =0;
    }
}
/*****************************************************************************/
```

- If the user recovery mode was selected, an extern interrupt would be called, where the necessary flag register will be cleared to retrieve CAN TX signal.

```
#pragma CODE_SEG __NEAR_SEG NON_BANKED
void interrupt vfnUserRecoveryMode_Isr(void)
{
 if (PTP_PTP0==0)   /*Button User request  */
    {
      if (CAN0MISC_BOHOLD==1) /*If module is bus off */
      {
       CAN0MISC_BOHOLD = 1;    /*Clear Bus off, state hold until user request */
      }
      PIFP_PIFP0 = 1;         /* Clear GPIO interrup flag  */
    }
}
#pragma CODE_SEG DEFAULT
```

- After compilation on automatic recovery mode configured, the following plot obtained shows the normal functionality of TX control signals:

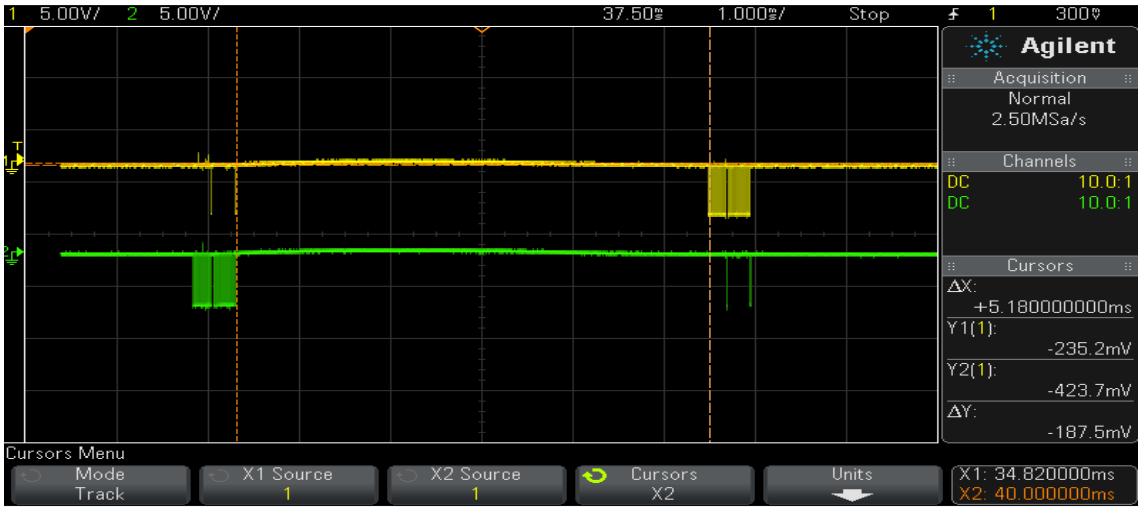


Figure 14.- Electrical CAN Activity of Controllers TX signal with bus-off Automatic recovery mode configured.

Shortcut between CAN_L and CAN_H was created, obtaining the following performance on TX signals:

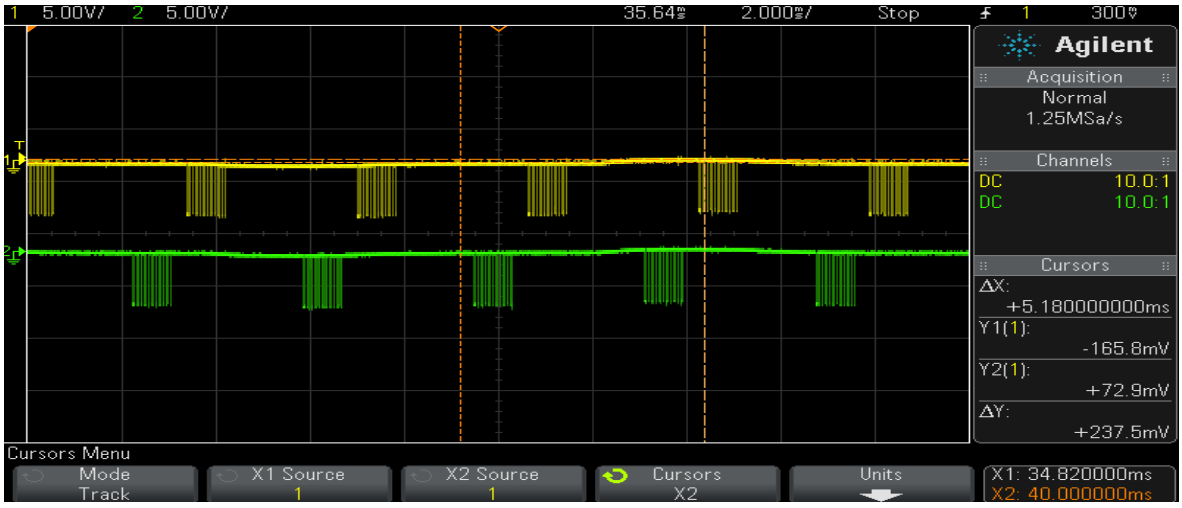


Figure 15.- Bus-off Automatic recovery mode, short between CAN H and CAN L lines.

After removal of short circuit between CAN lines, the bus-off automatic recovery mode is activated and the normal functionality of the bus is retrieved.

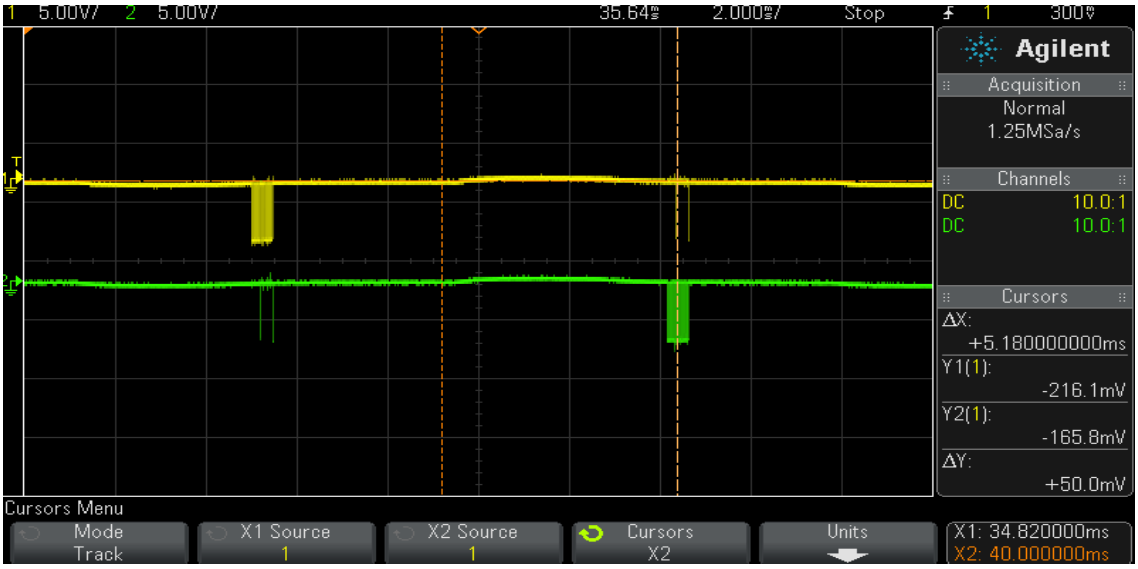


Figure 16.- Record of CAN TX signal after automatic bus-off recovery.

4.1.4 USER RECOVERY MODE

Configuration of one of the nodes was changed to observe the behavior of bus-off user recovery strategy; next plot illustrates the normal functionality of the TX communication signals between nodes.

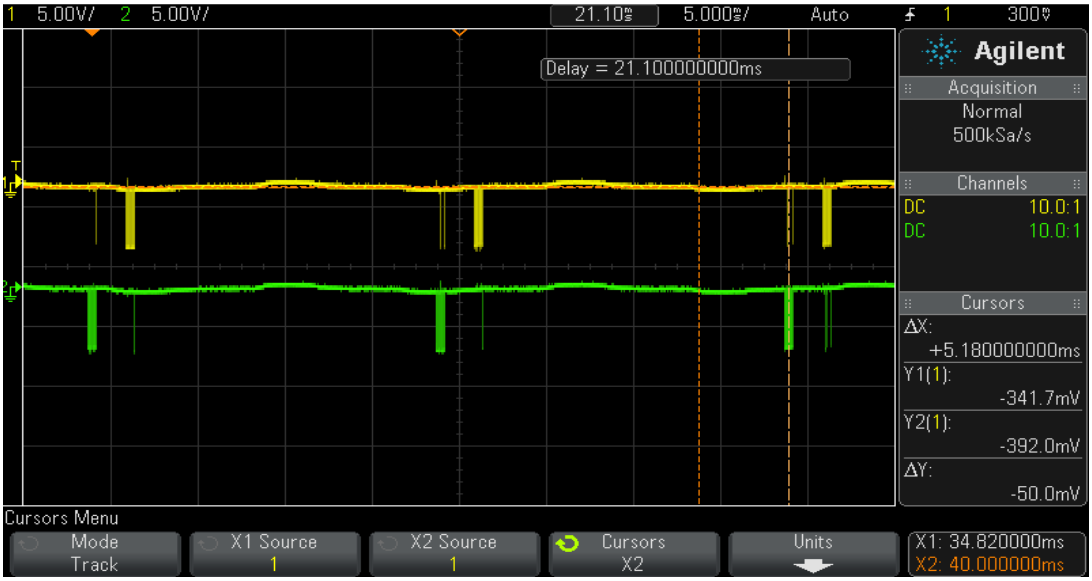


Figure 17.- Electrical CAN Activity of Controllers TX signal with two nodes connected.

After create a short circuit between CAN lines, the node with user recovery mode selected was automatically disabled due to the bus-off CAN functionality, as we can observe at the following plot:

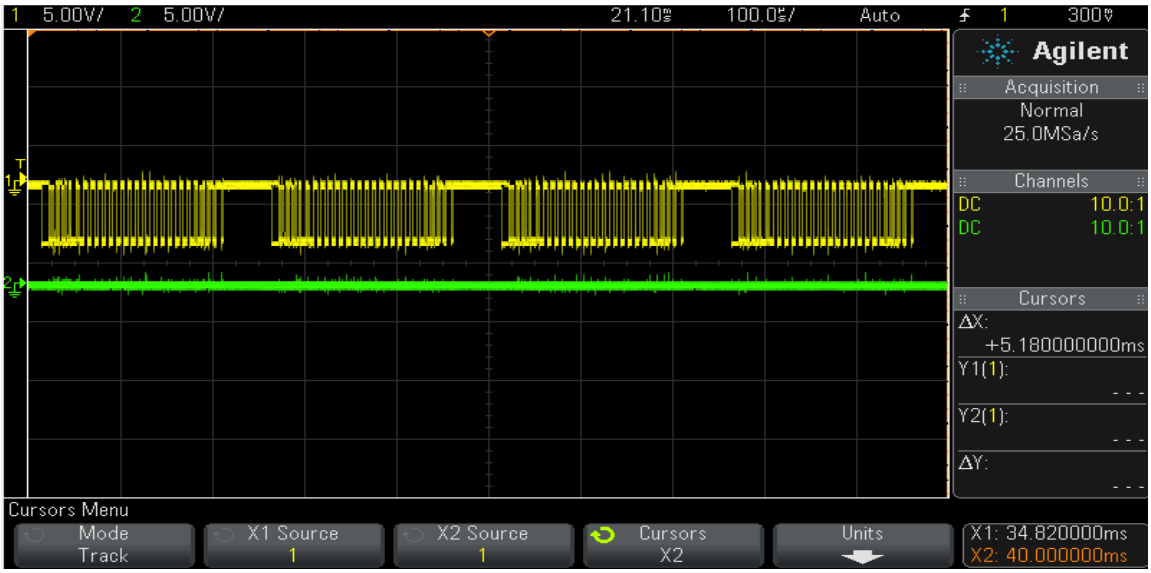


Figure 18.- Electrical CAN Activity of Controllers TX signal after short circuit between CAN L and CAN H lines.

The bus-off state is held up until the user recovery request is done by an interrupt service routine, then, Tx signals will be observed again as follows:

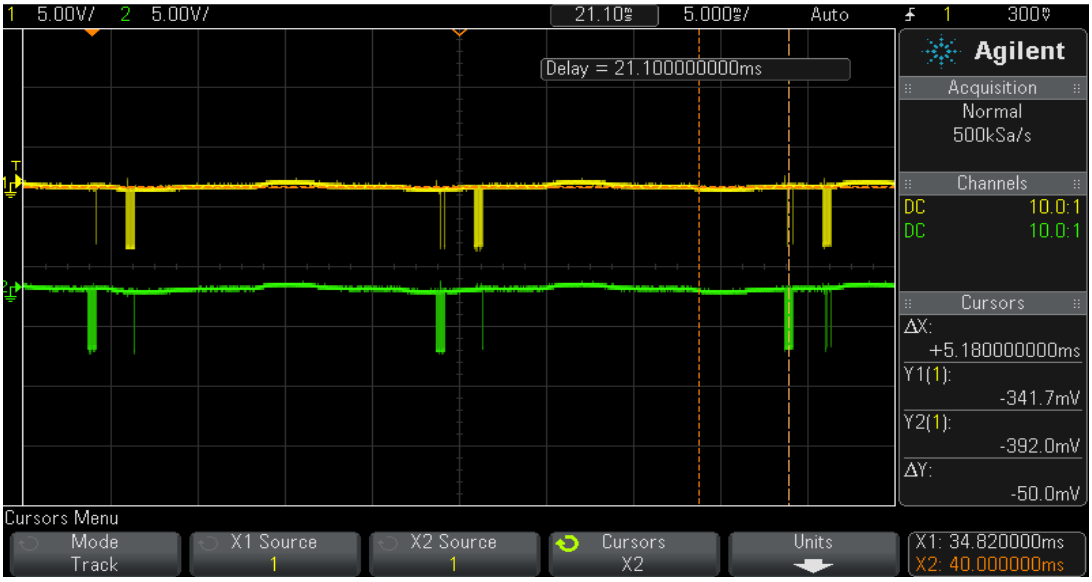


Figure 19.- Electrical Tx CAN Activity after bus-off user recovery.

5. CONCLUSIONS

Abstract: *In this chapter summarizes the conclusions and how the abilities developed in this work was improved my daily activities at work.*

5.1. Conclusions

Nowadays, develop of electronic control units (ECU's) is characterized by factors like demands for more security and flexibility in increasing system complexity applications which requires that the development of ECUs networks will be oriented and standardized by worldwide agreements that allow a common and compatible driver development among Automotive companies.

The abilities developed during the execution of the current work have allowed my mobility within my company since I have moved from a production line to Research and Development of embedded equipment for different OEM customers.

BIBLIOGRAPHY

[1]. DEMO9S12XEP100: Demo Board for the 16-bit MC9S12XE and XS-Families, http://www.nxp.com/products/automotive-products/microcontrollers-and-processors/16-bit-s12-s12x-mcus/demo-board-for-the-16-bit-mc9s12xe-and-xs-families.:DEMO9S12XEP100

[2]. LIN Consortium, http://www.lin-subbus.org/

[3]. ISO 17987-6:2016 Road vehicles -- Local Interconnect Network (LIN) -- Part 6: Protocol conformance test specification, http://www.iso.org/iso/catalogue_detail.htm?csnumber=61227

[4]. Docklight - Test & Simulate Serial Protocols, http://docklight.de/

[5]. Agilent Oscilloscope, http://www.keysight.com/en/pcx-x2015004/oscilloscopes?cc=MX&lc=eng