Scientific Research Publishing

World Journal of Engineering and Technology, 2018, 6, 214-224
http://www.scirp.org/journal/wjet
ISSN Online: 2331-4249
ISSN Print: 2331-4222

Graphical User Interface for the Monitoring of CAN Frames by Means of a SoC Cyclone V

Josefina Castañeda Camacho, Salvador Morales Caro, Gerardo Mino-Aguilar, Liliana Cortez, Italo Cortez, Ana María Rodríguez Domínguez, Rodrigo Lucio Maya Ramírez

Benemerita Universidad Autonoma de Puebla, Puebla, Mexico
Email: gmino44@ieee.org

How to cite this paper: Camacho, J.C., Caro, S.M., Mino-Aguilar, G., Cortez, L., Cortez, I., Domínguez, A.M.R. and Ramírez, R.L.M. (2018) Graphical User Interface for the Monitoring of CAN Frames by Means of a SoC Cyclone V. *World Journal of Engineering and Technology*, **6**, 214-224.
https://doi.org/10.4236/wjet.2018.61012

Received: October 31, 2017
Accepted: February 25, 2018
Published: February 28, 2018

Copyright © 2018 by authors and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/

Open Access

Abstract

In this paper, the design of a Graphical User Interface for CAN data frame monitoring is presented. The GUI has been developed in the Qt Creator IDE. A touch screen for visualization and control is used, which in turn is controlled by a development board with a SoC Cyclone V, through which a Linux operating system is executed.

Keywords

SoC, Touch Screen, Qt, Graphical User Interface, CAN Frames, Monitoring

1. Introduction

Currently, CAN networks are applied not only in cars. Also they are used in different industry branches. The understanding of the concept of a CAN network is relatively simple since it is based on a bus-type topology, that is, each node in the network is connected to a bus that serves as an information conveyance, making communication possible between all the nodes. What makes CAN networks special is the serial protocol they use, since they have well-defined rules allowing the correct exchange of information between nodes. The CAN protocol handles four types of bit frames with which it establishes the communication between the network nodes; these are: data frame, remote frame, overload frame and error frame.

Error and overload frames are generated automatically when some node in the network detects an error and when a node is not yet ready to send information. The data frame is the most important, since through this, the required informa-

tion is sent, and through the remote frames, it is possible to request the frames sending of data of a specific node. The data frames are divided into several fields, being the most important the DLC and Data. By means of the DLC, it is indicated that data length will contain the data frame, being the maximum 8 bytes.

When designing a CAN network, it is important that the engineer in charge of this task has a support in a reliable tool to verify that the exchange of information between nodes is performed according to specifications, and analyzers protocols are used which monitor and analyze the data traffic generated by some communication protocol, in this case the CAN protocol.

In the market, there are several analyzers of CAN frames. Usually these consist of a software that runs under a PC and it interacts with separately designed hardware to interact with the software.

In this work, the development of a graphical user interface for the monitoring of CAN frames is proposed, using a SoCKit development card from Arrow Development Tools, which contains as a central element a Cyclone V SoC in conjunction with a touch screen which serves as a means of visualization and control of the monitoring system.

The SoC Cyclone V is a device that is divided in two parts: a HPS "Hard Processor System" part, and a FPGA part. The HPS contains an ARM Cortex-A9 microprocessor, and specific purpose peripherals, among which are: I2C drivers, CAN, SPI, and some others. Therefore, one of the CAN controllers is used for the development of this work [1]-[8].

Since the HPS has an ARM microprocessor, it is possible to run a Linux distribution on the device. An image file provided by Terasic is used, which contains the necessary components to run a Linux distribution on the development board in conjunction with the touch screen. **Figure 1** shows the block diagram of the BSP provided by Terasic [9] [10] [11].

As can be seen in **Figure 1**, a Qt library is incorporated, which allows to run in the SoC graphical interfaces developed under the IDE Qt Creator.

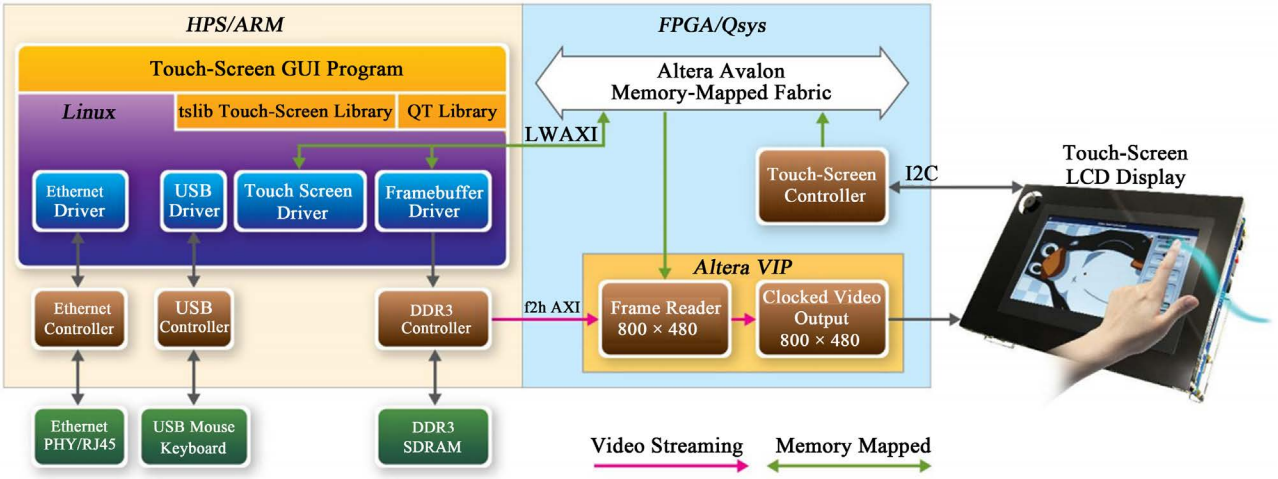

Figure 1. BSP provided by Terasic.

2. Can Controller

The CAN controller presented in the HPS is compatible with the CAN 2.0 A and 2.0 B protocols. Handles a speed of up to 1Mbs, it can maintain up to 128 messages and three test modes. All message transmission and reception activities are performed through message objects, which are stored in a message RAM and has the capacity to store up to 128 message objects.

It is possible to configure two types of message objects; for the reception and for the transmission. During the transmission the ID, DLC and Data fields (up to 8 bytes) that have to be sent in a data frame are configured. Receiving objects support any data frame that matches the configured ID. It is important to note that each configured message object has an object number from 1 to 128. In this way it is possible to use, for example, a number of message objects for all transmission activity and the remainders for reception.

Before using the CAN controller, it is necessary to initialize it. Figure 2 shows the initialization flowchart.

During controller initialization, the message RAM is initialized to delete existing message object configurations. The operating mode of the controller is set, in this case as shown in Figure 2, the controller is put into combined mode, which is one of the three test modes that the controller has. Finally, the bit time is adjusted according to the desired transfer rate and the initialization is terminated.

Once the initialization process is completed, the message objects are configured. Figure 3 shows the flowchart of the setting of a message object for transmission.

Figure 4 shows the flowchart of the setting of a message object for the reception.

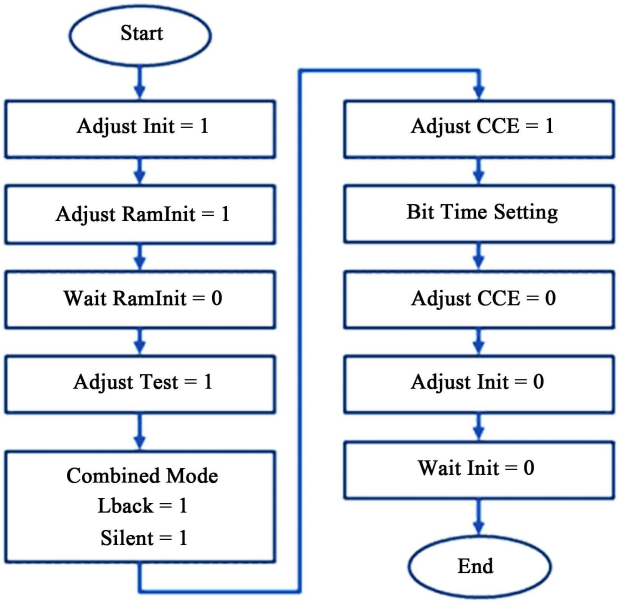

Figure 2. CAN driver initialization flowchart.

To simplify the testing of the developed graphical user interface, the CAN driver was configured to operate in combined mode. Through this operation mode, it is possible to test the CAN driver without the need for extra hardware. Figure 5 shows the CAN controller in combined mode [2].

In combined mode, the physical sending and receiving pins are disconnected from the CAN kernel, whereby the frames generated by the controller do not affect the CAN bus activity and so the controller does not take into account any of the present frames on the bus. However, as seen in Figure 5, the frames generated by the controller are directly received by it. With this, the generation and reception of data frames can be performed, emulating the behavior of the controller on a CAN bus.

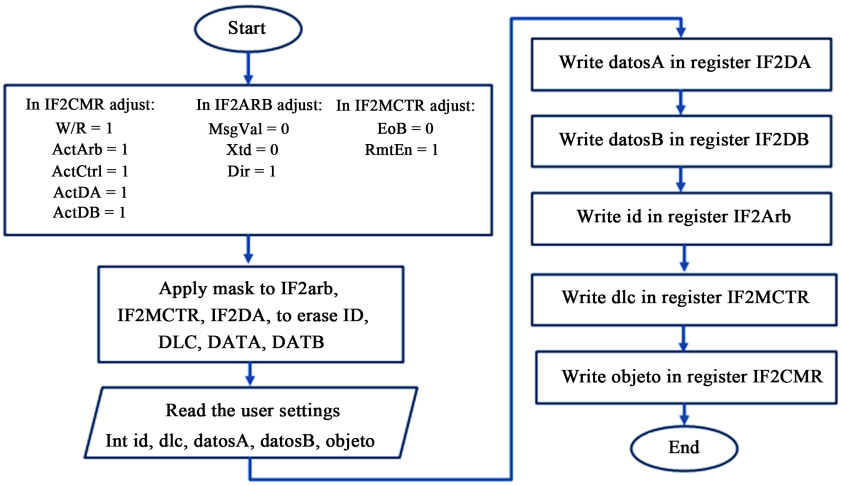

Figure 3. Flowchart for the initialization of transmission message objects.

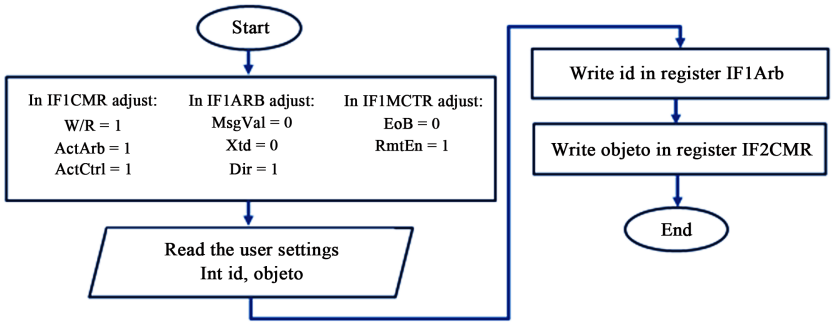

Figure 4. Flowchart for initialization of receiving message objects.

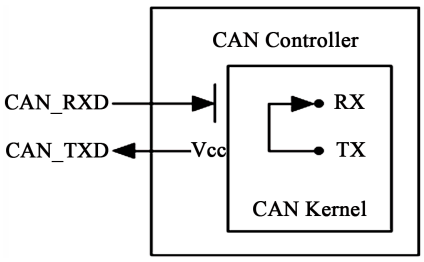

Figure 5. CAN controller in combined mode.

3. Development of the Graphic Interface

When running an application under Linux, it is not possible to directly access the configuration logs of the HPS peripherals, if it is desired to access directly then it is necessary to develop a specific driver for each peripheral. Therefore, we use the method of memory mapping to gain access to the memory space of the peripheral configuration registers.

The design of the interface was done through Qt Creator software. The graphical interface developed consists of two windows. The first includes a simple button which performs an initialization routine. Figure 6 shows the design of the main window.

By pressing the initialization button, the program emits a signal. This signal executes a slot containing the functions to open the physical memory of the device, to map the HPS peripherals and to initialize the CAN controller. Figure 7 shows the slot that is running.

As you can see in Figure 7, an object of the class called "functions" is created inside the slot. In the function class, all the functions related to the configuration and manipulation of the CAN controller were incorporated. Table 1 shows the member functions contained in the functions class.

When the initialization routine is finished, a second window which contains two tabs that are used for the sending of custom frames and the reception of the same ones is displayed. Figure 8 shows the appearance of the CAN data frame sending tab.

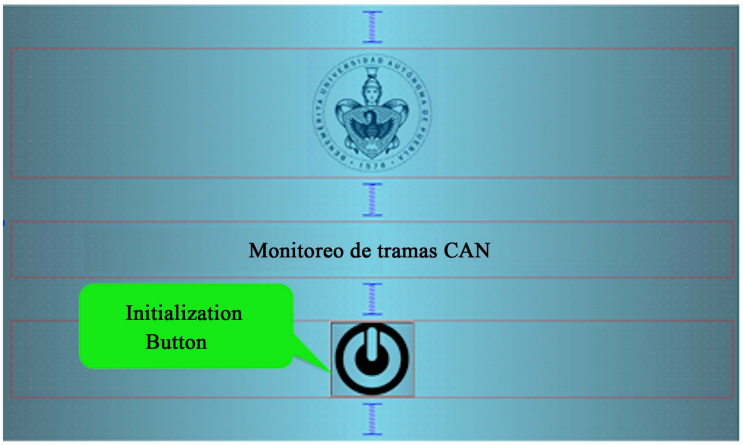

Figure 6. Main window.

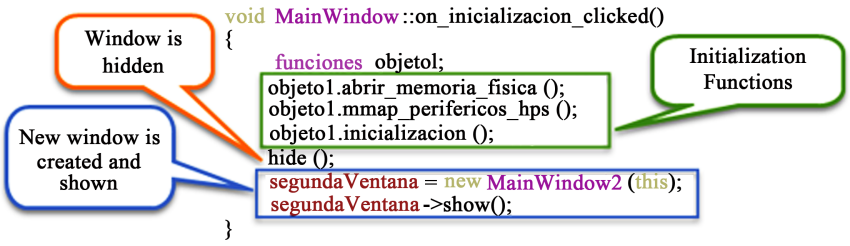

Figure 7. Slot executed at boot.

Table 1. Functions member of the functions class.

Functions	Description
void abrir_memoria_fisica( )	Open device descriptor file "dev/mem".
void cerrar_memoria_fisica( )	Close device descriptor file "dev/mem".
void mmap_perifericos_hps( )	Map HPS peripheral region
u_int32_t monitoreo_bin(u_int32_t registro, char nombre[ ])	Performs binary monitoring of records, prints the contents on the screen and returns the value.
u_int32_t monitoreo(u_int32_t registro)	Monitors and returns from the registry.
void inicialización( )	Initializes HPS peripherals.
void config_Tx(int id, int dlc, int datosa, int datosb, int objeto)	Configures a Tx message object.
void config_Rx(int id, int objeto)	Configures a Rx message object.
void iniTx( )	Initializes a Tx message object.
void iniRx( )	Initializes a Rx message object.
void enviarTrama(int obj)	Sends the data frame that is configured on the object.
void solicitar(int obj)	Sends a remote frame.
void leerRx(int obj)	Make the necessary configuration to read the received frame.
int leerDA( )	Returns the value of the first four bytes of the received data frame.
int leerDB( )	Returns the value of the bytes 4 to 7 of the received data frame.
int leerDLC( )	Returns the DLC value of the received data frame.
void valoresRx(int & dlc0, int & b7, ···, int & b0)	This function is used to obtain the values of a received data frame, and convert them into strings of characters.

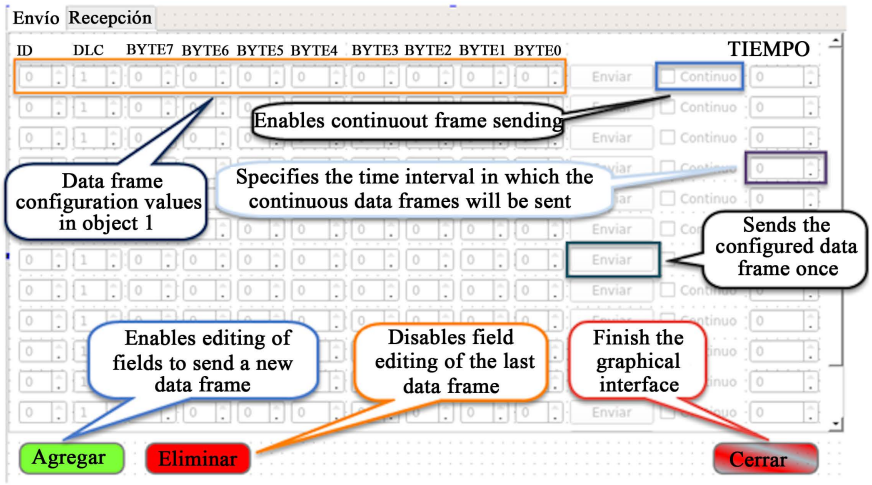

Figure 8. Frame sending tab.

The frame sending tab consists of the following elements:

- ID field. This defines the ID of the message to be sent.
- DLC field. This defines the data length (up to 8 bytes). If values less than 8 are entered in this field then the unused bytes fields are not disabled. However, the message object only takes into account the number of bytes configured according to the DLC.

- Data byte fields. These define the data to be sent. Because each byte consists of 8 bits, the maximum number in decimal allowed for each of these fields is bounded to 255.
- Send button. By means of this button the frame with the data that has configured in the moment that happens the event is sent.
- Continuous Shipment and Time. The checkbox corresponding to the continuous sending acts in conjunction with the time field. Once the continuous send checkbox is activated the send button is disabled and then the parameter configured in the time field is taken to send the frame every x milliseconds, where x is the value set in the time field. For convenience, this parameter was set to only be configured in intervals of 500 milliseconds.
- Add button. Each time you press this button, the edit fields of a new message are enabled.
- Delete button. This button disables the edit fields of the last added message.

In the receiving tab, the message objects that will receive the CAN data frames are configured. Figure 9 shows the appearance of the receiving tab.

As shown in Figure 9, in the receiving tab, the user only needs to configure the ID field, which serves as an acceptance filter for receiving frames of data with the same ID. In addition, this tab contains a data frame request button and a monitoring checkbox. When pressing the request button, a remote frame is generated. The node has configured the same ID. The monitoring checkbox enables the reading of the object from time to time for new data. Once data has been received, it is displayed in the DLC and Bytes 7 - 0 fields.

The monitoring process is similar to the continuous sending of frames. First the registers are initialized for the configuration of reception objects, then the value present in the ID field is obtained to use in the configuration of the reception message object.

Once the receiving object is configured, the data reception routine is executed which follows the flow diagram shown in Figure 10.

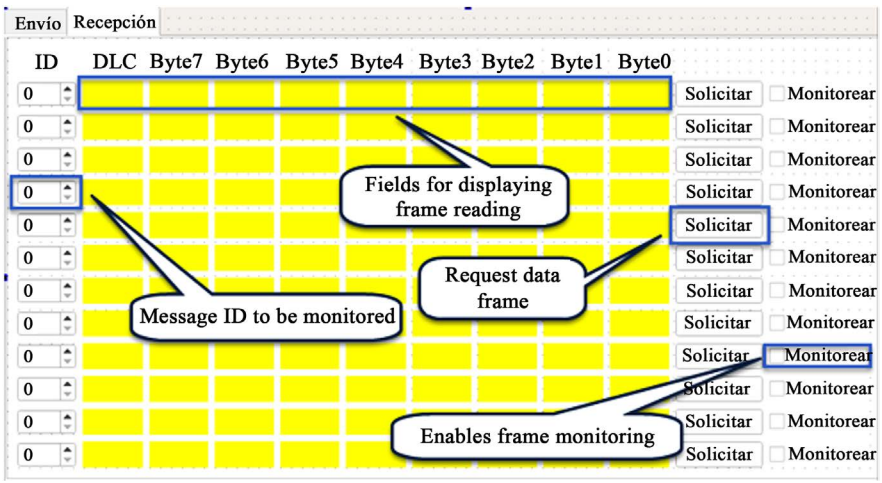

Figure 9. Frame reception tab.

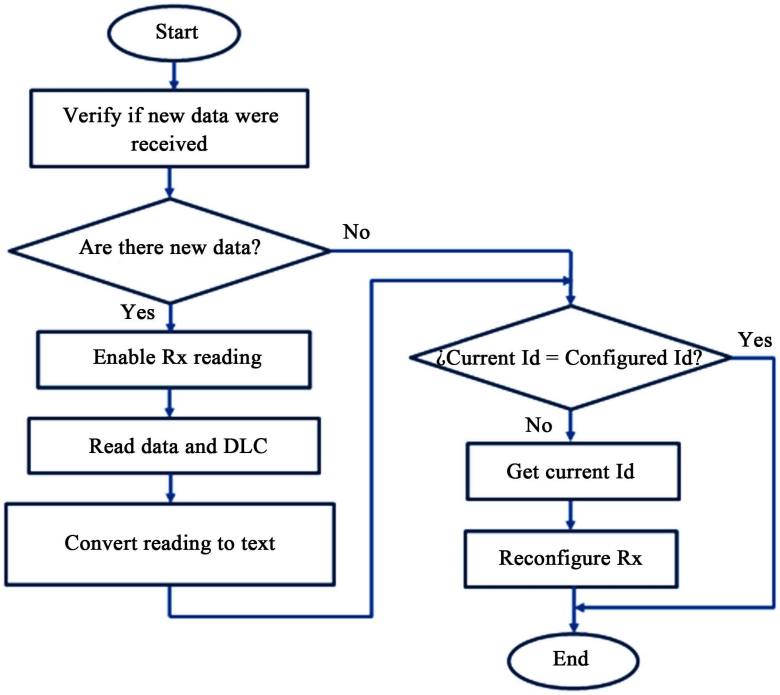

Figure 10. Flow diagram for receiving and reading data frames.

First it is check if new data was received, if there is new data unread, it is enable reading of the corresponding message object. The value of the DLC and the received data are converted to text using Qt's own functions. The resulting text is placed in the spaces dedicated to the display of the frame information. Once the read process is completed, it is checked whether the user-configured ID has changed, if so, the message object is reconfigured to match the current ID. The receive fields always reflect the last message reading of the current ID.

The frame request acts in conjunction with frame monitoring because a transmission message object is not required for the sending of remote frames, and therefore it is sufficient to use a receive object and generate the remote frame from itself.

4. Tests and Results

Once the graphical interface is developed, it is executed on the development board. For test purposes a modification was made to this, in the send tab a field of reception was added, so that the changes in this one can be displayed in an agile way when receiving data frames.

Figure 11 shows the appearance of the main window, running on the touch screen.

Pressing the initialization button, as already mentioned, a routine that initializes the general parameters of the controller, besides opening the physical memory of the device and mapping the peripherals of the HPS is executed. Figure 12 shows the monitoring of the records involved in the initialization process.

Figure 11. Main window.

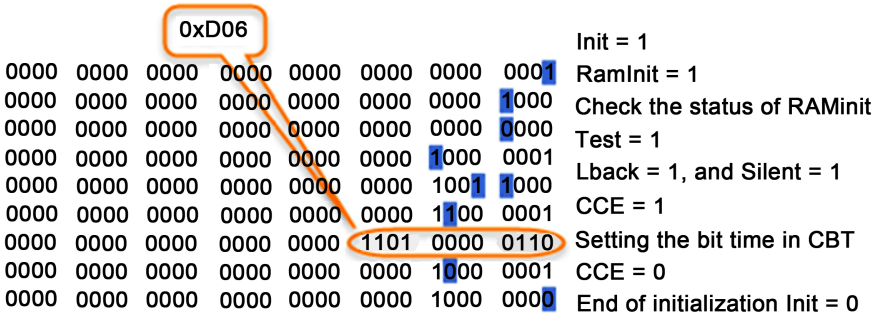

Figure 12. Monitoring of the initialization routine.

Figure 13 shows the appearance of the modified shipping tab.

Continuous frame sending tests were performed, and it was verified that the data was indeed received and reflected in the reception fields. Figure 14 shows the sending and receiving of a continuous frame with ID equal to six.

As you can see, in the monitoring fields the data corresponding to the configured frame is reflected, in addition to the assigned DLC. In order to verify that the data frames were indeed being generated and received, the behavior of the controller registers was monitored. Figure 15 shows the monitoring of the records involved in the reception of the data frame shown in Figure 14.

When configuring the message objects, it is possible to enable interrupt flags that are put to one, when a data frame is transmitted or received satisfactorily. For the case shown in Figure 14, the transmission interrupt and the receive interrupt were enabled. With the help of the flags that were enabled, it was possible to verify that data frames were indeed generated and that they were received satisfactorily, as can be seen in Figure 15. The case of the request for data frames is similar.

5. Conclusions

A graphical user interface was developed through the touch screen, the configuration of the CAN controller in the HPS to inject personalized data frames and the monitoring.

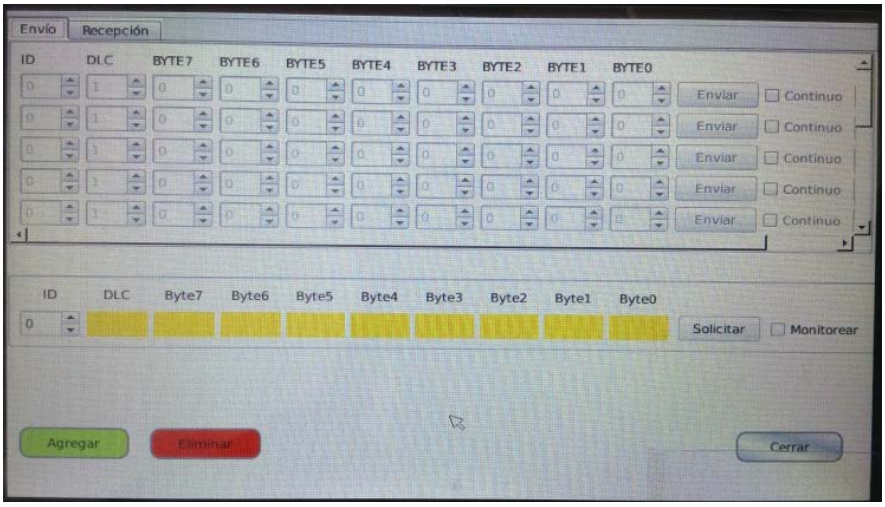

Figure 13. Shipping tab.

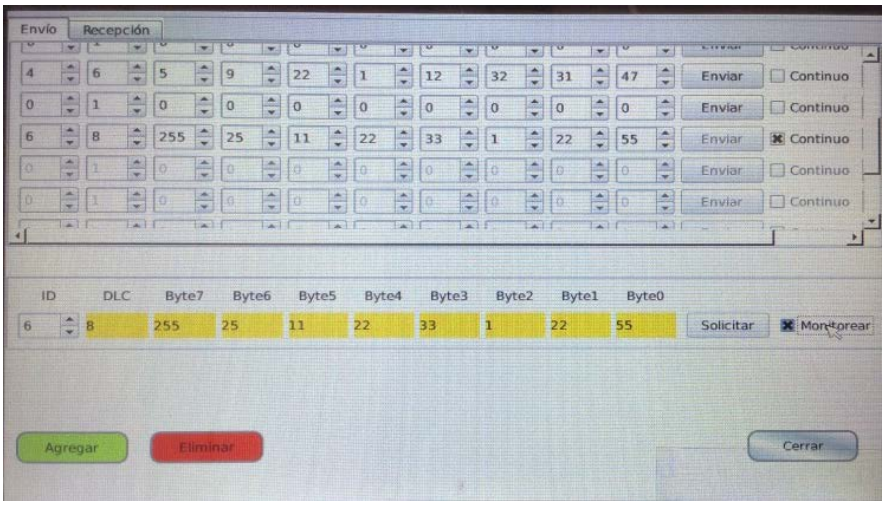

Figure 14. Sending and receiving a continuous data frame.

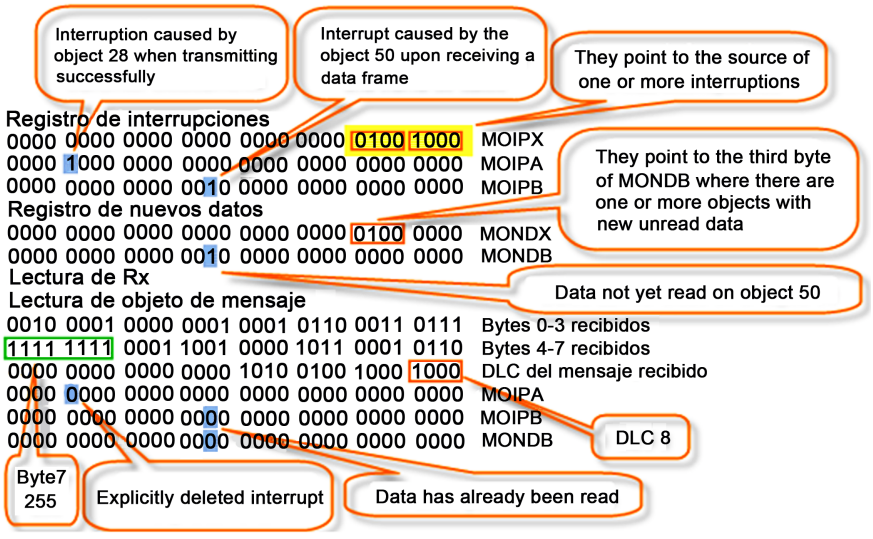

Figure 15. Monitoring of records in the reception mode of a data frame.

It was possible to successfully test the functionality of the graphical interface developed, thanks to the combined operation mode of the controller, and by verifying the status of the controller registers, it was confirmed that data frames are actually generated and monitored.

References

[1] Di Natale, M., Zeng, H., Guisto, P. and Ghosal, A. (2012) Understanding and Using the Controller Area Network Communication Protocol. Springer.

[2] Eng, L.Z. (2016) Qt5 C ++ GUI Programming Cookbook. Pack Publishing.

[3] Chin, D. (1998) Executing System on a Chip: Requirments for a Successfull SOC Implementation. *IEEE IEDM*, San Francisco, CA, 6-9 December 1998, 3-8.

[4] Zajc, M., Sernec, R. and Tasic, J. (2002) An Efficient Linear Algebra SoC Design: Implementation Considerations. *IEEE MELECON*, Cairo, 7-9 May 2002, 322-326.

[5] Pham, D., Anderson, H.-W., *et al.* (2006) Key Features of the Design Methodology Enabling a Multi-Core SOC Implementation of a First-Generation CELL Processor. *IEEE Asia and South Pacific Conference on Design Automation*, Yokohama, 24-27 January 2006, 871-878.

[6] Pham, D., *et al.* (2005) The Design and Implementation of a First-Generation CELL Processor. *ISSCC Digest of Technical Papers*, San Francisco, CA, 10 February 2005, 184-185.

[7] Flachs, B., *et al.* (2005) The Microarchitecture of the Streaming Processor for a CELL Processor. *IEEE Journal of Solid-State Circuits,* **41**, No. 1.

[8] Ikram, S., Akkawi, I., Perveiler, J., *et al.* (2014) A Framework for Specifying, Modeling, Implementation and Verification of SOC Protocols. 27*th IEEE International System-on-Chip Conference* (*SOCC*), Las Vegas, NV, 2-5 September 2014, 268-273. https://doi.org/10.1109/SOCC.2014.6948939

[9] VEEK-MT-SoCKit. Terasic Technologies Inc, 2013-2014.

[10] Cyclone V Hard Processor System Technical Reference Manual. Altera Corporation, 11-05-2015.

[11] SoCkit User Manual. Terasic Technologies Inc, 2003-2015.