INTERFAZ GRÁFICA DE USUARIO PARA EL MONITOREO DE TRAMAS CAN POR MEDIO DE UN SOC CYCLONE V

Salvador Morales Caro

Benemérita Universidad Autónoma de Puebla

salva_673@hotmail.com

Josefina Castañeda Camacho

Benemérita Universidad Autónoma de Puebla

josefinacastaneda@yahoo.com.mx

Gerardo Mino Aguilar

Benemérita Universidad Autónoma de Puebla

gmino44@ieee.com

Resumen

En este trabajo se presenta el diseño de una Interfaz Gráfica de Usuario para el monitoreo y generación de tramas de datos CAN, desarrollada en el IDE Qt Creator. Además, se presenta la metodología utilizada para realizar la configuración de un controlador FCAN el cual permite la generación de tramas de datos CAN personalizadas, las cuales fueron utilizadas para validar el funcionamiento de la interfaz gráfica desarrollada. Como medio de visualización y control se utilizó una pantalla táctil, la cual a su vez es controlada por una tarjeta de desarrollo que tiene como elemento central un SoC Cyclone V, en el cual se ejecuta un sistema Operativo Linux embebido el cual incluye librerías Qt, lo cual hizo posible la ejecución de la aplicación desarrollada. El SoC Cyclone V se divide en dos partes, una parte conformada por el denominado Hard Processor System y la otra compuesta por el FPGA, se hizo uso del HPS debido a que cuenta con periféricos dedicados a los cuales puede acceder directamente, de entre ellos se encuentran dos controladores FCAN, de modo que se hizo uso de uno configurado en modo combinado para la generación y lectura de tramas de datos CAN.

Palabra(s) Clave: Controlador CAN, GUI, HPS, LINUX Embebido, Monitoreo de tramas, Qt Creator.

Abstract

This paper presents the design of a graphical user interface for monitoring and generating CAN data frames, it was developed in the QT Creator IDE. Also it is presented the methodology used to make the configuration of an FCAN controller which lets the generation of custom CAN data frames that were used to validate the operation of the developed graphical user interface. As a medium of visualization and controlling it was used a touch panel which in turn is controlled by a development board that has as central element a SoC Cyclone V in which is executed an embedded LINUX Operating System and through Qt libraries it was possible to execute the application developed. The SoC Cyclone V is divided into two parts, one part conformed by the denominated Hard Processor System and the another one by an FPGA, it was used the HPS due to it has dedicated peripherals that can be accessed directly, among them there are two FCAN controllers, one of them was configured in combined mode to generate and read CAN data frames.

Keywords: *CAN Controller, Embedded LINUX, Frame monitoring, GUI, HPS, Qt Creator.*

1. Introducción

En los inicios de la electrónica en los automóviles, cada función era implementada en una "Unidad de Control Electrónico" (ECU), de tal modo que si una ECU necesitaba información de otra ECU esta se comunicaba por medio de cableado punto a punto, pero al ir aumentando las funciones electrónicas presentes en los automóviles el número de cables que pasaban por el chasis de estos creció a tal grado que ya no era factible manejar el mismo esquema de comunicación. Debido a dicha situación surgió la "Red Controladora de Área" (CAN), la cual está basada en una topología de red tipo bus. En la topología de red tipo bus, también conocida como bus lineal, el elemento central es un cable al cual todos los nodos (por ejemplo, las ECUs) están conectados mediante cables.

La información es transmitida por los nodos en forma de mensajes y se distribuyen por todo el bus, haciendo posible la comunicación entre todos los nodos.

El bus CAN tiene una estructura multi-maestro donde cada nodo conectado puede enviar o recibir datos siempre y cuando el bus esta libre, pero únicamente un nodo puede enviar datos a la vez mientras todos los demás escuchan. Si dos o más dispositivos intentan enviar datos al mismo tiempo, al que tiene la prioridad más alta se le permite enviar sus datos mientras los otros escuchan o dicho de otro modo regresan a modo recepción, esto debido a que el protocolo nunca interrumpe las transmisiones en curso, pero asigna prioridades a los mensajes para prevenir conflictos y asegurar que los mensajes urgentes sean entregados primero. Cada nodo selecciona aquellos mensajes que son relevantes y los demás los ignora.

Un nodo CAN esencialmente está compuesto de los elementos que se muestran en la figura 1. El microcontrolador se encarga de la gestión de los datos recopilados por sensores, y de la activación de actuadores, cuando se requiere enviar información de un nodo a otro esta pasa al controlador CAN y se generan las tramas seriales de bits propias del protocolo CAN. El bus maneja niveles de voltaje diferenciales, por lo cual antes de que se pueda transmitir una trama esta tiene que ser ajustada a los niveles de voltaje que maneja la red CAN, esto se logra a través del transceptor CAN.

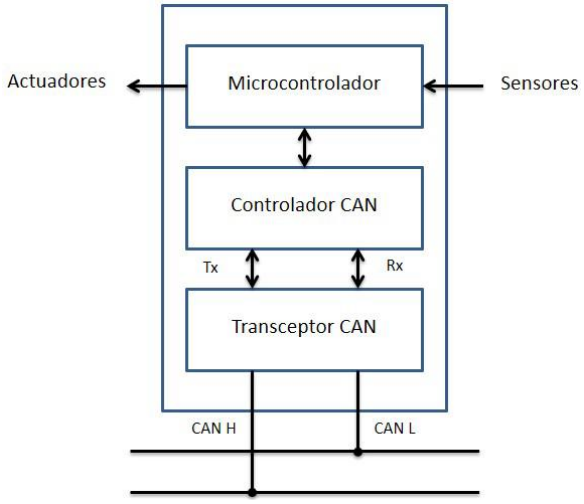


Figura 1 Ejemplo de los componentes típicos de un nodo CAN.

El protocolo CAN utiliza cuatro tipos de tramas de bits para la transmisión de mensajes; trama de datos, trama remota, trama de sobrecarga y trama de error. Ya que el protocolo CAN es del tipo serial, cuando se habla de tramas CAN se entiende que es una cadena de bits trasmitida en formato serial, es decir, se envía bit a bit uno tras otro hasta completar la totalidad de la trama.

Lo importante a destacar de estas cadenas de bits es el hecho de que están divididas en segmentos bien definidos, dependiendo del tipo de trama que se transmita. De los cuatro tipos de tramas que maneja el protocolo CAN la más importante es la trama de datos, ya que a través de ella se realiza el envío de la información que se desea difundir a uno o varios nodos dependiendo de los requerimientos de la red. Existen dos tipos de tramas de datos, la 2.0 A también llamada estándar, y la 2.0 B también llamada extendida. En la figura 2 se muestran los campos que conforman una trama de datos estándar.

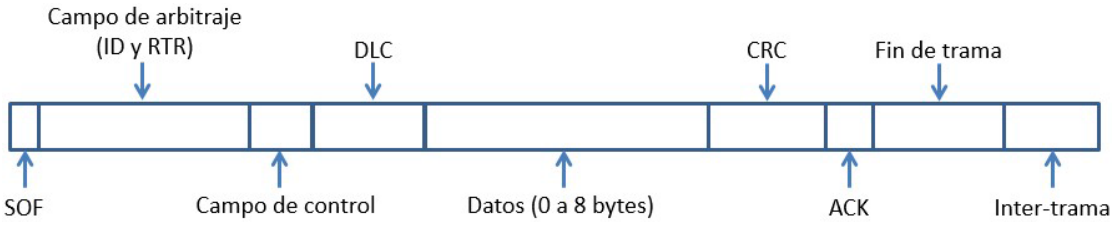


Figura 2 Ejemplo de los campos de la trama de datos CAN estándar.

De los campos que se muestran en la figura 2 los de mayor relevancia son el de identificador (ID), el de longitud de datos (DLC) y el de datos. Como se mencionó anteriormente el protocolo CAN asigna prioridades a los mensajes salientes, y esto se logra por medio del ID. El campo de DLC es de importancia ya que por medio de este se indica la longitud que tendrá el campo de datos siendo este de un máximo de 8 bytes. Hablando de tramas de datos solo resta decir que la diferencia entre la 2.0 A y la 2.0 B es el tamaño del identificador, siendo de 11 bits para la estándar y de 29 para la extendida. Las tramas remotas son de utilidad para solicitar información de un nodo en particular. Para ello hace uso del ID, y como respuesta el nodo que contenga un ID de mensaje coincidente con la trama remota responderá con una trama de datos.

Las tramas de error y sobrecarga son generadas automáticamente cuando se detecta un error y cuando un nodo aún no está listo para enviar información respectivamente. Durante el diseño y depuración de redes CAN es importante verificar que el intercambio de información entre nodos se realice según las especificaciones realizadas. Para ello se utilizan analizadores de protocolos, los cuales monitorean y analizan el tráfico de datos generado por algún protocolo de comunicación en específico, en este caso el protocolo CAN.

En el mercado existen varios analizadores de tramas CAN, generalmente estos consisten de un software que corre bajo una PC e interactúa con hardware diseñado por separado para brindar retroalimentación desde el medio físico al software.

En este artículo se plantea el desarrollo de una interfaz gráfica de usuario para el monitoreo de tramas CAN utilizando una tarjeta de desarrollo SoCKit de la empresa Arrow Development Tools®, la cual contiene como elemento central un SoC Cyclone V, en conjunto con una pantalla táctil que sirve como medio de visualización y control del sistema de monitoreo. En la figura 3 se presenta el diagrama a bloques de lo que sería el sistema de monitoreo si se implementara en una red CAN.

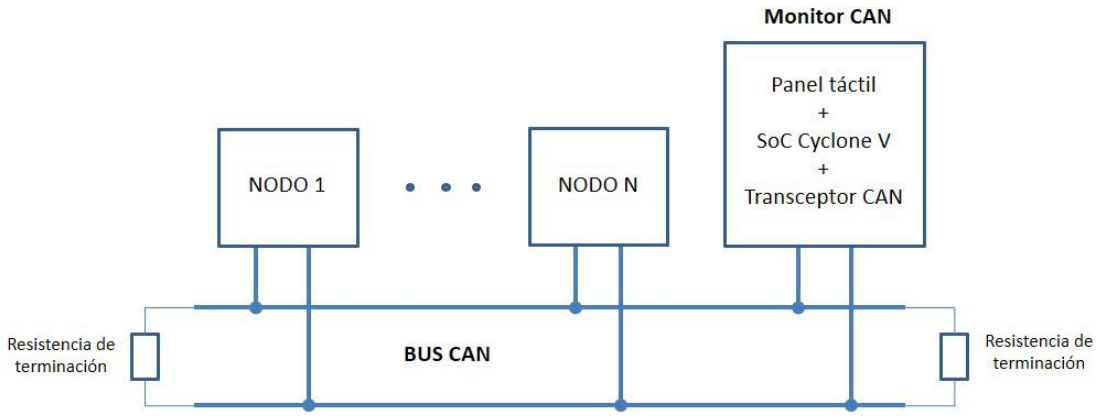
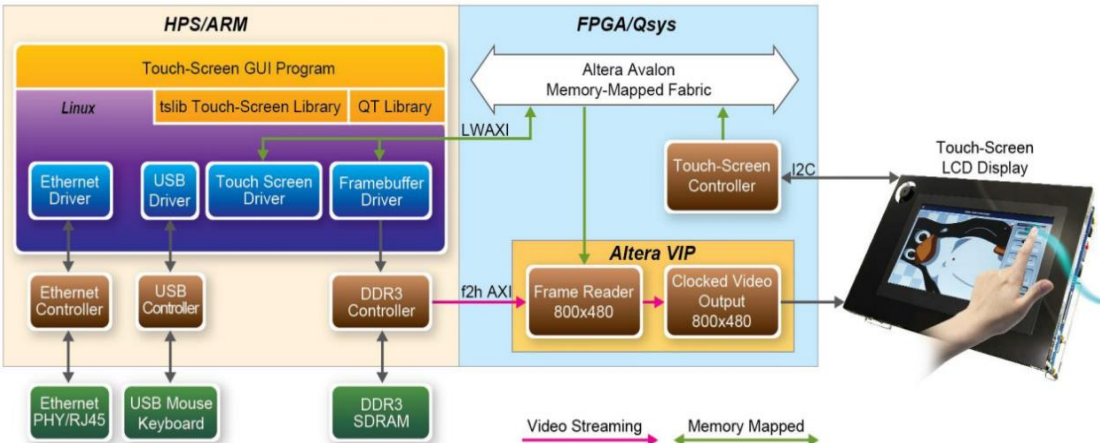


Figura 3 Ejemplo del sistema de monitoreo en una red CAN.

El SoC Cyclone V, es un dispositivo que está dividido en dos partes; una parte llamada "Hard Processor System" (HPS), y una parte "Field Programable Gate Array" (FPGA). El HPS contiene un microprocesador ARM Cortex-A9, y periféricos

de propósito específico, entre los cuales se encuentran; controladores i2C, UART, CAN y SPI.

Ya que el HPS posee un microprocesador ARM, es posible correr una distribución de Linux en el dispositivo. Se utilizó un archivo de imagen provisto por Terasic®, el cual contiene todos lo necesario para correr Linux en la tarjeta de desarrollo en conjunto con el panel táctil. En la figura 4 se muestra el diagrama a bloques del "Board Support Package" (BSP) provisto por Terasic.



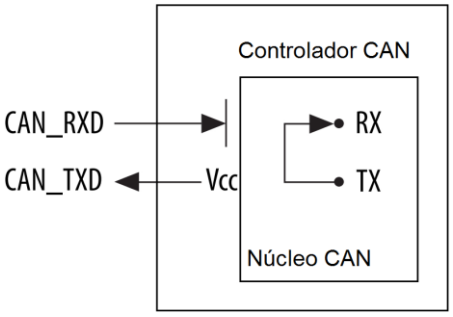
Fuente: Terasic Technologies Inc. [2014]

Figura 4 Ejemplo de TERASIC Linux BPS for Touch-Screen LCD Display Module.

Como se puede ver en la figura 4 el BSP incorpora una librería de Qt, la cual permite correr en el SoC interfaces gráficas desarrolladas bajo el "Entorno de Desarrollo Integrado" (IDE) Qt Creator.

El controlador CAN presente en el HPS, es compatible con el protocolo CAN 2.0A y 2.0B, maneja una velocidad de hasta 1 Mbs, puede mantener hasta 128 mensajes. Toda la actividad de transmisión y recepción de mensajes, es realizada a través de objetos de mensaje, los cuales son almacenados en una memoria RAM de mensaje, que tiene capacidad para almacenar hasta 128 objetos de mensaje. Es posible configurar dos tipos de objetos de mensaje; para recepción y para transmisión. En los objetos de transmisión se configuran los campos de ID, DLC y datos que se desean enviar en una trama de datos. Los objetos de recepción admiten cualquier trama de datos que coincida con el ID configurado.

Es importante destacar que cada objeto de mensaje configurado, posee un número de objeto desde el 1 hasta el 128. De tal modo que es posible utilizar, por ejemplo, un cierto número de objetos de mensaje para toda la actividad de transmisión y los restantes para recepción. Una característica importante que posee este controlador es su modo de funcionamiento, ya que puede ser configurado para funcionar en uno de cuatro modos; normal, silencioso, retroalimentación y modo combinado. En modo silencioso el controlador es capaz de recibir tramas sin participar en el envío de estas, poniendo su pin de transmisión en alto. En modo retroalimentación el controlador trata sus propios mensajes trasmitidos como mensajes recibidos y los guarda en el buffer de recepción. El modo combinado, el cual es de especial importancia para este trabajo, se logra al poner el controlador en modo silencioso y modo retroalimentación al mismo tiempo. A través de este modo de operación, es posible probar el controlador CAN sin necesidad de hardware extra. En la figura 5 se muestra el controlador CAN en modo combinado.



Fuente: Altera Coporation. [2016]

Figura 5 Ejemplo del CAN Core in Combined Mode.

En modo combinado, los pines físicos de envío y recepción son desconectados del núcleo CAN, en consecuencia, las tramas generadas por el controlador no afectan la actividad del bus CAN y del mismo modo el controlador no toma cuenta de ninguna de las tramas presentes en el bus. Sin embargo, como se ve en la figura 5, las tramas generadas por el controlador, son directamente recibidas por él mismo. Con lo cual se puede realizar la generación y recepción de tramas de datos, emulando el comportamiento del controlador en un bus CAN.

2. Metodología

Para el desarrollo de este trabajo fue necesario cubrir dos aspectos importantes; la configuración del controlador CAN y el diseño y programación de la interfaz gráfica de usuario. El primer aspecto que se cubrió fue el de la configuración del controlador, pero para ello fue necesario conocer los principios de funcionamiento del protocolo CAN por medio de la consulta de literatura referente al tema, una vez hecho esto se procedió a hacer un estudio detallado de cómo utilizar el SoC Cyclone V y finalmente se estudió como desarrollar interfaces graficas con el IDE Qt Creator. En la figura 6 se muestra el diagrama de flujo de los pasos generales que se siguieron para el desarrollo de este trabajo.

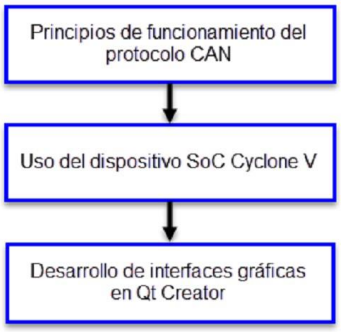


Figura 6 Ejemplo del diagrama de flujo de la metodología seguida para este trabajo.

A continuación, se detalla el proceso que se siguió para realizar la configuración del controlador y posteriormente se trata lo referente a la interfaz gráfica. Antes de poder utilizar el controlador CAN es necesario inicializarlo. En la figura 7 se muestra el diagrama de flujo de la inicialización.

El primer paso es inicializar la RAM de mensaje para eliminar las configuraciones de objetos de mensaje existentes. Se ajusta el modo de funcionamiento del controlador, en este caso como se ve en la figura 7, se pone el controlador en modo combinado. Por último, se ajusta el tiempo de bit de acuerdo a la tasa de transferencia deseada y se da por terminada la inicialización.

Una vez concluido el proceso de inicialización, se procede a configurar los objetos de mensaje. En la figura 8 se muestra el diagrama de flujo del ajuste de un objeto de mensaje para transmisión.

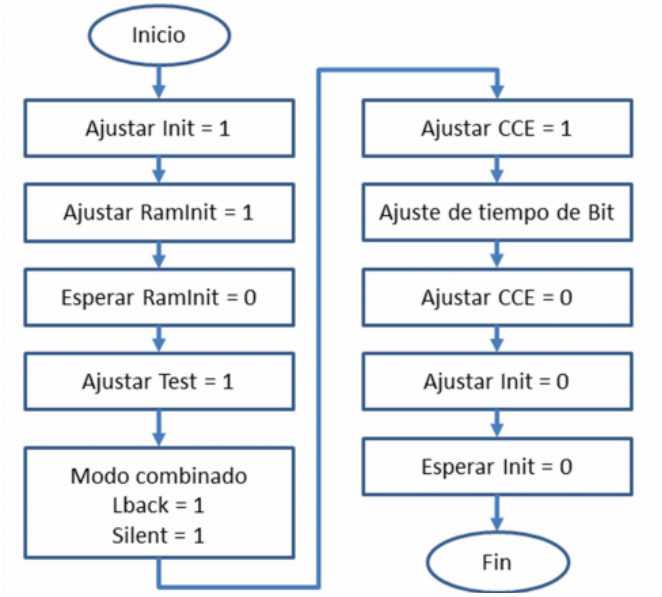


Figura 7 ejemplo del diagrama de flujo de inicialización del controlador CAN.

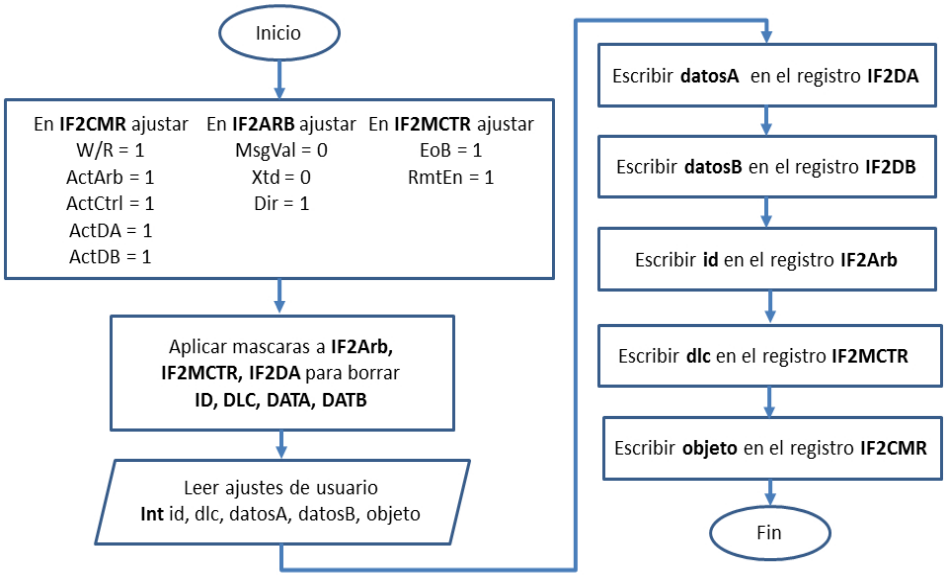


Figura 8 Diagrama de flujo para la inicialización de objetos de mensaje de transmisión.

En la figura 9 se muestra el diagrama de flujo del ajuste de un objeto de mensaje para recepción, concluyendo lo referente a la configuración del controlador CAN. Cuando se corre una aplicación bajo Linux, no es posible acceder directamente a los registros de configuración de los periféricos del HPS, si se desea acceder de manera directa, es necesario el desarrollo de un driver específico para cada

periférico. Por lo cual, se utilizó el método de mapeo de memoria, utilizando este método se gana acceso al espacio de memoria de los registros de configuración del periférico que se desea utilizar, o de todos ellos. La interfaz gráfica desarrollada consiste de dos ventanas. La primera con un botón simple el cual al ser presionado realiza una rutina de inicialización. En la figura 10 se muestra el diseño de la ventana principal.

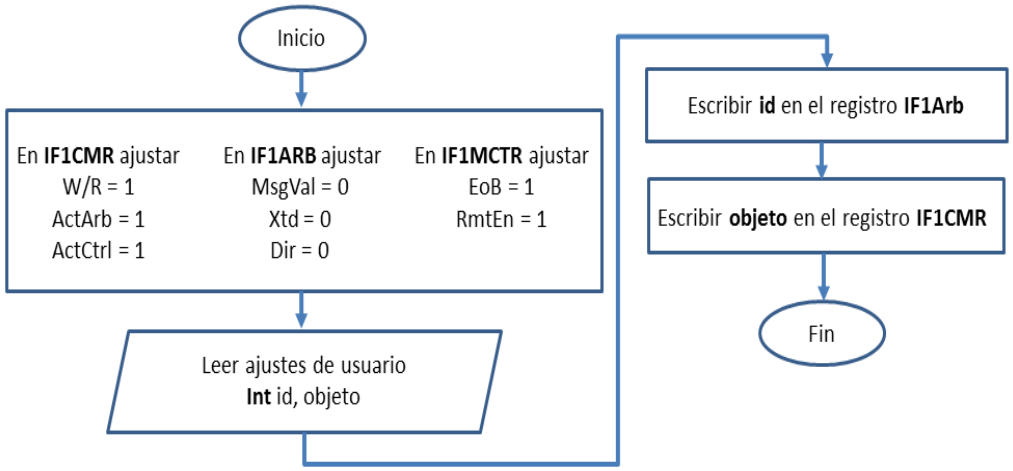


Figura 9 Diagrama de flujo para la inicialización de objetos de mensaje de recepción.

Figura 10 Ejemplo de la ventana de inicio.

Al presionar el botón de inicialización se emite una señal, la cual ejecuta un slot que contiene la llamada a las funciones para abrir la memoria física del dispositivo, mapear los periféricos del HPS e inicializar el controlador CAN. En la figura 11 se muestra el slot que se ejecuta.

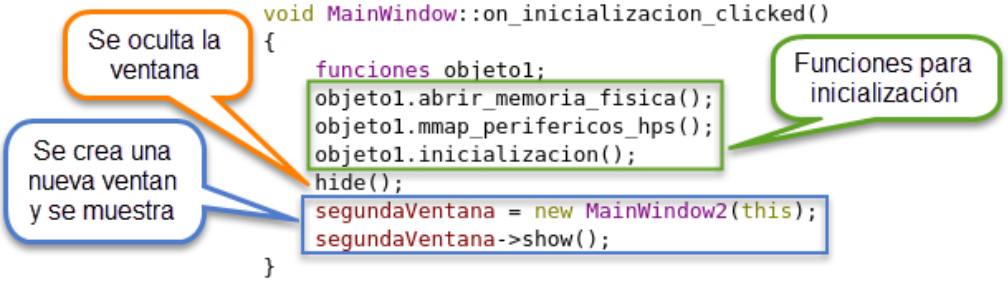


Figura 11 Ejemplo del Slot ejecutado al inicializar.

Como se puede ver en la figura 11, dentro del slot se crea un objeto de la clase llamada "funciones". En la clase funciones se incorporaron todas las funciones relacionadas con la configuración y manipulación del controlador CAN. En la tabla 1 se muestran las funciones miembro contenidas en la clase funciones.

Tabla 1 Ejemplo de las funciones miembro de la clase funciones.

Funciones	Descripción
void abrir_memoria_fisica( )	Abre el archivo descriptor de dispositivos "/dev/mem".
void cerrar_memoria_fisica( )	Cierra el archivo descriptor de dispositivos "/dev/mem".
void mmap_perifericos_hps( )	Mapea la región de periféricos del HPS
u_int32_t monitoreo_bin(**u_int32_t** registro, **char** nombre[ ])	Realiza monitoreo binario de los registros, imprime en pantalla el contenido y regresa el valor.
u_int32_t monitoreo(**u_int32_t** registro)	Monitorea y regresa del registro.
void inicializacion( )	Inicializa los periféricos del HPS
void config_Tx(**int** ID, **int** DLC, **int** DatosA, **int** DatosB, **int** objeto)	Configura un objeto de mensaje de Tx.
void config_Rx(**int** ID, **int** objeto)	Configura un objeto de mensaje de Rx.
void iniTx( )	Inicializa los objetos de mensaje de
void iniRx( )	Inicializa los objetos de mensaje de
void enviarTrama(**int** obj)	Envía la trama de datos que se encuentre configurada en el objeto.
void solicitar(**int** obj)	Envia una trama remota.
void leerRx(**int** obj)	Realiza la configuración necesaria para poder leer la trama recibida.
int leerDA( )	Devuelve el valor de los primeros 4 bytes de la trama de datos recibida.
int leerDB( )	Devuelve el valor de los bytes 4-7 de la trama de datos recibida.
int leerDLC( )	Devuelve el valor del DLC de la trama de datos recibida.
void valoresRx(**int** &dlc0, **int**&b7, … , **int**&b0)	Esta función es utilizada para obtener los valores de una trama de datos recibida, y convertirlos en cadenas de caracteres.

Al terminar la rutina de inicialización, se despliega una segunda ventana la cual contiene dos pestañas que son utilizadas para el envío de tramas personalizadas y la recepción de las mismas. En la figura 12 se muestra la apariencia de la pestaña de envío de tramas de datos CAN.

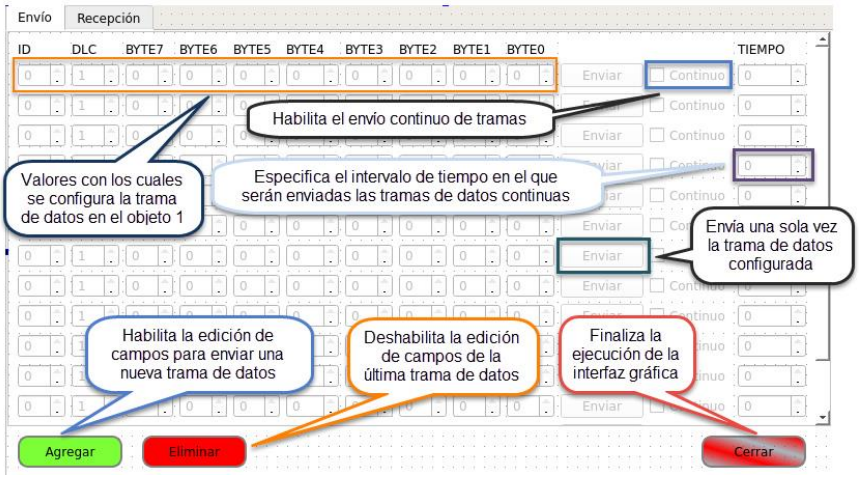


Figura 12 Ejemplo de la pestaña de envío de tramas.

La pestaña de envío de tramas consiste de los siguientes elementos:

- Campo de ID. En este se define el ID del mensaje a enviar.
- Campo de DLC. En este se define la longitud de datos. Si en este campo se ingresan valores menores a 8, no se deshabilitan los campos de bytes no utilizados. Sin embargo, el objeto de mensaje solo tomará en cuenta el número de bytes configurados de acuerdo al DLC. El hecho de que no se deshabiliten los campos de bytes no utilizados fue solo cuestión de diseño, pero fácilmente se podría cambiar esta cuestión si se deseara.
- Campos de bytes de datos. En estos se definen los datos a ser enviados. Debido a que cada byte consiste de 8 bits, el número máximo permitido en decimal para cada uno de estos campos está acotado a 255.
- Botón enviar. Por medio de este botón se envía la trama que se encuentre configurada en los campos, al momento de ser presionado.
- Envío continuo y Tiempo. El checkbox correspondiente al envío continuo actúa de manera conjunta con el campo de tiempo. Una vez que se activa

el checkbox de envío continuo se deshabilita el botón enviar y a continuación el parámetro configurado en el campo de tiempo es tomado para enviar la trama cada x milisegundos, donde x es el valor configurado en el campo de tiempo. Por comodidad este parámetro se ajustó para solo ser configurado en intervalos de 500 milisegundos.

- Botón agregar. Cada que se pulsa este botón, se habilitan los campos de edición de un nuevo mensaje.
- Botón eliminar. Este botón deshabilita los campos de edición del último mensaje agregado.

En la pestaña de recepción se configuran los objetos de mensaje que recibirán las tramas de datos CAN. Como se ve en la figura 13, en la pestaña de recepción, el usuario sólo necesita configurar el campo de ID, el cual sirve como filtro de aceptación para recibir tramas de datos que contengan el mismo ID. Además, esta pestaña contiene un botón de solicitud de trama de datos y un checkbox de monitoreo.

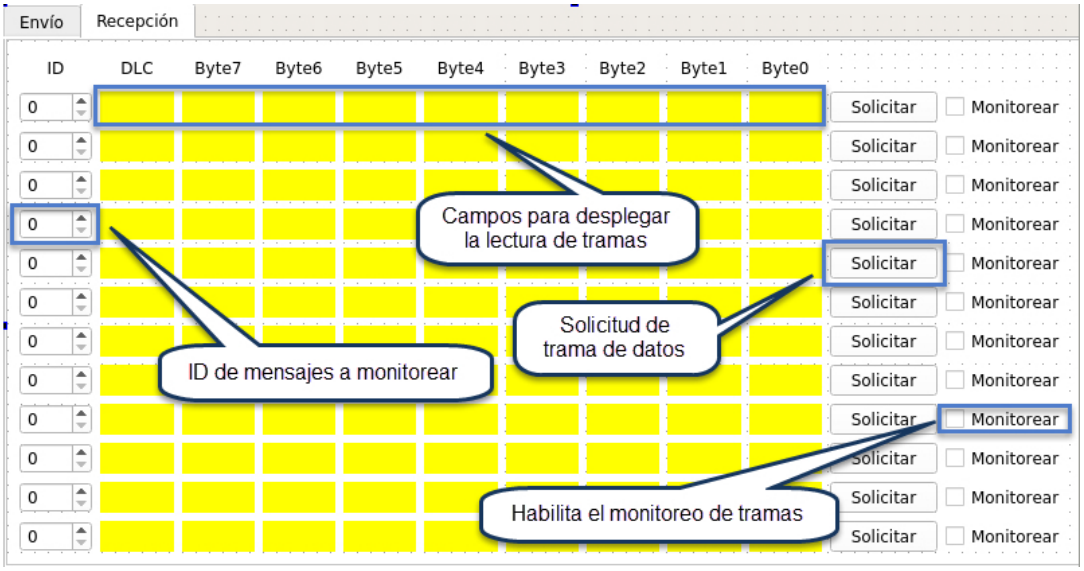


Figura 13 Ejemplo de la pestaña de recepción de tramas.

Al pulsar el botón de solicitar, se genera una trama remota, que por medio de la cual como ya se había mencionado, el nodo que tenga configurado el mismo ID

responderá con una trama de datos. El checkbox de monitoreo habilita cada cierto tiempo la lectura del objeto en busca de nuevos datos. Una vez que se han recibido datos, estos son desplegados en los campos de DLC y Bytes 7-0.

El proceso de monitoreo es similar al de envío continuo de tramas, salvo que en este caso el usuario no define cada cuanto se genera la señal proveniente del temporizador. Primero se inicializan los registros para la configuración de objetos de recepción, una vez hecho, se obtiene el valor presente en el campo de ID para utilizarlo en la configuración del objeto de mensaje de recepción. Después de concluir la configuración del objeto de recepción, se ejecuta la rutina de recepción de datos la cual sigue el diagrama de flujo mostrado en la figura 14.

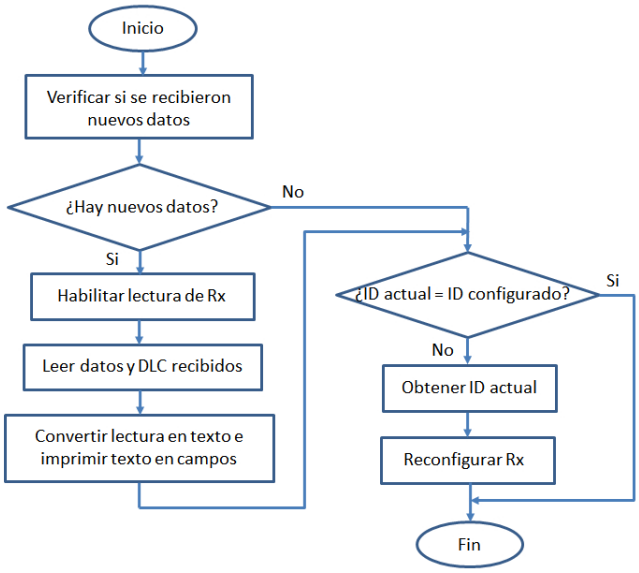


Figura 14 Ejemplo del diagrama de flujo para la recepción y lectura de tramas de datos.

Primero se verifica si se recibieron nuevos datos. En caso de que existan nuevos datos sin leer se habilita la lectura del objeto de mensaje correspondiente. El valor del DLC y los datos recibidos se convierten a texto por medio de funciones propias de Qt. El texto resultante se coloca en los espacios dedicados al despliegue de la información de trama. Una vez que se ha completado el proceso de lectura, se verifica si el ID configurado por el usuario ha cambiado, de ser así, se reconfigura el objeto de mensaje para que coincida con el ID actual. Los campos de recepción siempre reflejan la última lectura de mensaje del ID actual.

La solicitud de trama actúa en conjunto con el monitoreo de tramas debido a que no se necesita un objeto de mensaje de transmisión para el envío de tramas remotas, y por ello basta con utilizar un objeto de recepción y generar la trama remota desde él mismo.

3. Resultados

Una vez desarrollada la interfaz gráfica se procedió a ejecutarla en la tarjeta de desarrollo. Para propósitos de prueba se realizó una modificación a esta, en la pestaña de envío se agregó un campo de recepción, para poder visualizar de manera más ágil los cambios en este al recibir tramas de datos. En la figura 15 se muestra la apariencia de la ventana principal ejecutándose en la pantalla táctil. Al presionar el botón de inicialización, como ya se había mencionado se ejecuta una rutina que inicializa los parámetros generales del controlador, además de abrir la memoria física del dispositivo y mapear los periféricos del HPS.

En la figura 16 se muestra el monitoreo de los registros involucrados en el proceso de inicialización, mientras que en la figura 17 se muestra la apariencia de la pestaña de envío modificada.

Figura 15 Ejemplo de la ventana principal.

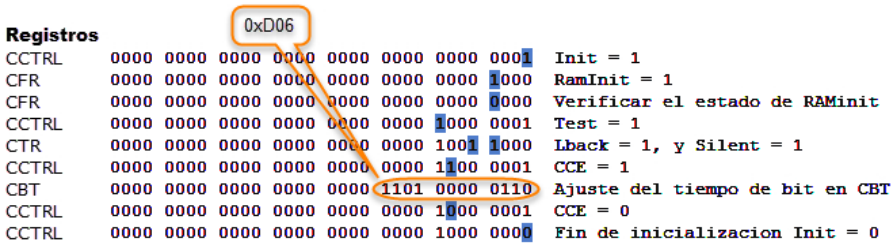


Figura 16 Ejemplo del monitoreo de la rutina de inicialización.

Figura 17 Ejemplo de la pestaña de envío modificada.

Se realizaron pruebas de envío de tramas continuas, y se verificó que efectivamente se recibieran los datos y se reflejaran en los campos de recepción. En la figura 18 se muestra el envío y recepción de una trama continua con ID igual a seis. Como se puede ver, en los campos de monitoreo se reflejan los datos correspondientes a la trama configurada, además del DLC asignado. Para comprobar que efectivamente se estuvieran generando las tramas de datos y que estas fueran recibidas se monitoreó el comportamiento de los registros del controlador. En la figura 19 se muestra el monitoreo de los registros involucrados en la recepción de la trama de datos que se mostró en la figura 18.

Cuando se configuran los objetos de mensaje es posible habilitar banderas de interrupción, que son puestas a uno cuando se transmite o se recibe satisfactoriamente una trama de datos. Para el caso que se muestra en la figura 18, se habilitó la interrupción por transmisión y la interrupción por recepción. Con ayuda de las banderas que se habilitaron, se pudo verificar que efectivamente se generaron tramas de datos y que estas fueron recibidas satisfactoriamente como se puede ver en la figura 19. El caso de la solicitud de tramas de datos no es presentado, debido a que por medio de imágenes no es muy evidente la diferencia entre la solicitud de una trama de datos y el monitoreo de tramas generadas continuamente o de manera única.

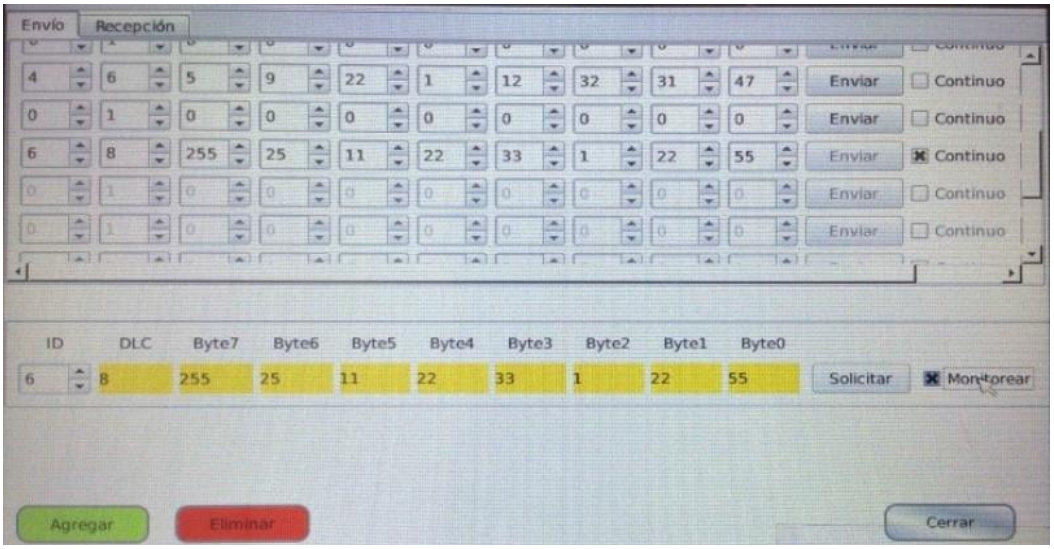


Figura 18 Ejemplo del envío y recepción de una trama de datos continua.

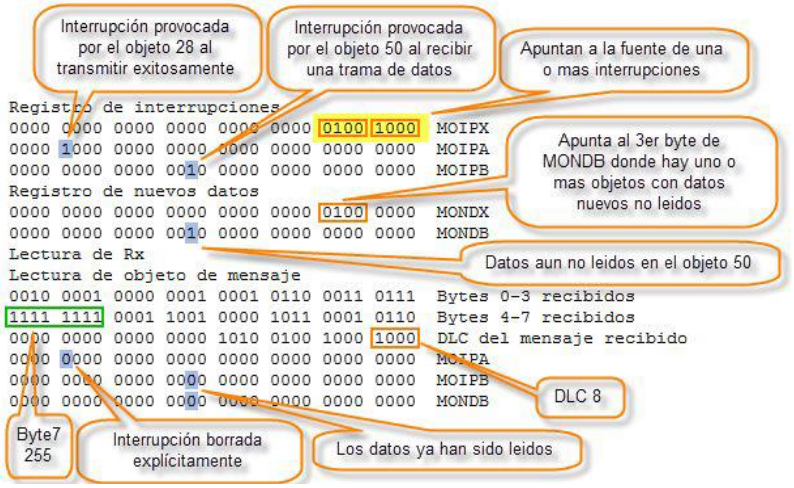


Figura 19 Ejemplo del monitoreo de registros en la recepción de una trama de datos.

4. Discusión

Gracias al modo de operación combinado del controlador CAN, y por medio de la verificación del estado de los registros del controlador, se logró probar con éxito la funcionalidad de la interfaz gráfica desarrollada. Por lo cual se pudo confirmar que se generaron tramas de datos, y es posible su monitoreo.

Si bien se logró probar la funcionalidad de la interfaz gráfica desarrollada, sería importante habilitar los pines del controlador CAN para conectarlos a un transceptor y hacer que el sistema de monitoreo interactúe con un bus real.

Para este articulo sólo se analizó la posibilidad de tener un sistema de monitoreo de tramas CAN, pero como se mencionó, además de controladores CAN el SoC utilizado posee controladores SPI, I2C y UART. Esto abre muchas posibilidades para el crecimiento del proyecto, ya que podría pensarse no sólo en un sistema de monitoreo para un protocolo, sino un sistema de monitoreo multiprotocolo. Sería necesario estudiar cada protocolo en particular, para agregar nuevos apartados a la interfaz gráfica dedicados a cada uno. La configuración de cada módulo en específico, representaría el aprender a utilizar los registros de configuración de cada módulo, pero con la ventaja de que ya se conoce el procedimiento que se debe seguir para ganar acceso a los registros de cualquier periférico del HPS.

El sistema propuesto podría ser de gran ayuda en el ámbito académico para el aprendizaje de protocolos de comunicación, ya que si lo comparamos con cualquier analizador comercial tiene la ventaja de que se puede monitorear el comportamiento de registros de configuración, un hecho que no es importante para un usuario normal, pero que es de vital importancia para el estudiante que desea comprender a fondo el funcionamiento de los dispositivos encargados de generar algún protocolo de comunicación.

En conclusión, es posible crecer el proyecto ya que el dispositivo elegido cuenta con los recursos necesarios para permitirlo, y aunado al hecho de que se adquirió experiencia en la configuración de periféricos del HPS y desarrollo de interfaces gráficas en Qt facilitaría la posibilidad de llevarlo a cabo.

5. Bibliografía y Referencias

[1] Altera Coporation. (2016). Cyclone V Hard Processor System Technical Reference Manual: https://goo.gl/SY1Wip.

[2] Lawrenz, W. (1997, 2013). CAN System Engineering: From theory to Practical Aplications. London: Springer-Verlag.

[3] Ledesma, T., Coya, L. (2012). Herramientas de monitorización y análisis del tráfico de redes de datos. Revista telem@tica. Vol. 11. No. 2, p. 46-55.

[4] Navet, N., Simonot-Lion, F. (2009). Automotive Embedded Systems Handbook. Boca Raton: CRC Press Taylor & Francis Group.

[5] Di Natale, M., Zeng, H., Guisto, P., & Ghosal, A. (2012). Understanding and Using the Controller Area Network Comunication Protocol. New York, USA: Springer.

[6] Stallings, W. (2004). Comunicaciones y redes de computadores. Madrid, España: Pearson Educación, SA.

[7] Terasic Technologies Inc. (2014). Software Development Guide for touch-screen display: https://goo.gl/PKWuCD.

[8] Terasic Technologies Inc (2015). SoCkit User Manual: https://goo.gl/Ccd7r2.

[9] Zhi, L. (2016). Qt5 C++ GUI Programming Cookbook. Birmingham, UK: Packt Publishing Ltd.