Articulo **Revista Tecnología e Inovación**
Diciembre 2016 Vol. 3No.9 67-75

Prueba de nodos CAN por medio de un SoC Cyclone V

MORALES, Salvador*†, CASTAÑEDA, Josefina, MINO, Gerardo

Recibido Abril 07, 2016; Aceptado Octubre 17, 2016

Resumen

La red controladora de área y su protocolo surge de la necesidad de compartir información entre distintos sistemas electrónicos ahorrando el mayor número de cables posible. En un principio el protocolo CAN fue utilizado en la industria automotriz, pero a través de los años su uso se diversifico haciendo del protocolo un estándar ampliamente aceptado. Cuando se realiza el diseño de una red de nodos CAN el diseñador debe poseer una herramienta con la cual probar de una manera fiable el funcionamiento de cada nodo conectado a la red. En este trabajo se propone el uso de una tarjeta de desarrollo con un SoC Cyclone V, mediante la cual se realiza el envío de tramas CAN por medio de un programa corriendo bajo Linux. El SoC Cyclone V se divide en dos partes; el FPGA de fábrica y el llamado Hard Proccesor System (HPS) el cual integra un procesador de doble núcleo ARM-A9 bajo el cual se puede correr Linux y periféricos que tienen la posibilidad de interactuar directamente con este. Entre los periféricos disponibles para el uso mediante el HPS se encuentran dos controladores CAN, de los cuales uno es utilizado para el desarrollo de este trabajo.

SoC Cyclone V, Nodos CAN, Linux.

Abstract

The controller area network and its protocol arises from the need of sharing information between the diferent electronic systems and saving the maximum number of wires. At the beginning the CAN protocol was used by the automotive industry, but through the years its use was diversified making it a standard widely accepted. When a network of CAN nodes is done the designer needs to have a tool with which to test in a reliable way the performance of each node connected to the network. In this paper the use of a development board with a SoC Cyclone V is proposed to realize the sending of CAN frames through a program running under Linux. The SoC Cyclone V is divided in two parts, the FPGA fabric and the Hard Processor System (HPS) which integrates a dual core ARM-A9 processor where you can run a Linux operating system, it also contains peripherals that can interact directly with the HPS. Among the peripherals available to use there are two CAN controllers and one of them is used for the development of this work.

SoC Cyclone V, CAN Nodes, Linux.

Citación: MORALES, Salvador, CASTAÑEDA, Josefina, MINO, Gerardo. Prueba de nodos CAN por medio de un SoC Cyclone V. Revista Tecnlogía e Innovación 2016. 3-9 : 67-75

*Correspondencia al Autor (Correo Electrónico: *salvador.moralesc@alumno.buap.mx)*
† Investigador contribuyendo como primer autor.

Introducción

En la actualidad existe un gran número de protocolos de comunicación, los cuales surgen debido a la necesidad de transmitir información de una manera fiable y segura, tal es el caso del protocolo CAN, el cual es uno de los protocolos más utilizados principalmente en la industria automotriz, ya que fue creado con el propósito de reducir el número de cables que sirven como medio de comunicación entre las muchas unidades de control electrónico que existen en el automóvil, con el paso del tiempo ha sido adoptado ampliamente en la industria y en otras ramas.

El protocolo CAN, es utilizado en redes de topología tipo BUS, denominadas en general redes CAN. En la figura 1 se muestra el esquema general de una red CAN de tres nodos.

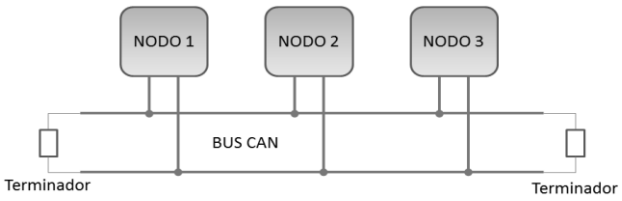

Figura 1 Red CAN.

Como se puede ver en la figura 1, hay tres nodos conectados al bus CAN, lo cual les otorga la posibilidad de enviar información a través del bus, de acuerdo a las reglas establecidas por el protocolo CAN. Dicha información es enviada en forma de tramas de datos seriales, las cuales están divididas en campos bien definidos.

El diseñador de una red CAN define qué información será transmitida por cada nodo y la prioridad que cada nodo tendrá para acceder al bus, por ello es necesario que conozca todos y cada uno de los campos de la trama de datos del protocolo.

Al llevar a cabo el diseño de una red de nodos CAN, es importante poseer una herramienta con la cual probar de una manera fiable, el funcionamiento de cada nodo conectado a la red.

Tramas de datos CAN

El CAN soporta dos diferentes formatos de trama, los cuales están especificados en el CAN 2.0 A o estándar y el CAN 2.0 B o extendido. La diferencia más importante entre estos dos formatos es la longitud del identificador, el cual es de longitud de 11 bits para el CAN 2.0 A y de 29 bits para el 2.0 B, ambos formatos son compatibles entre sí y pueden ser utilizados en una misma red.

La trama de datos estándar se divide en diferentes campos de longitud específica como se muestra en la figura 2.

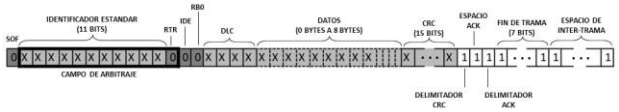

Figura 2 Trama de datos CAN 2.0 A.

Comienza con el bit de inicio de trama (SOF) en estado dominante. El SOF es seguido por el campo de arbitraje, el cual consta de 12 bits; el identificador de 11 bits y el bit de solicitud de transmisión remota (RTR).

En el campo de arbitraje se determina la prioridad del mensaje de acuerdo a una regla simple, "el valor numérico más bajo tiene la mayor prioridad". Si el bus se encuentra desocupado (en estado recesivo) y los mensajes están disponibles para ser enviados, cada nodo es libre de iniciar el envío de su mensaje. Cuando varios nodos comienzan a transmitir simultáneamente, el sistema responde aplicando un arbitraje para resolver los conflictos resultantes sobre el acceso al bus.

ISSN-En línea: 2410-3993

Después del campo de arbitraje sigue el campo de control. En una trama CAN estándar, el campo de control comprende el bit IDE (bit de extensión de identificador), el cual siempre es enviado como dominante, seguido por un bit reservado para extensiones futuras, el cual es enviado como recesivo (RB0), y el campo de código de longitud de datos (DLC); formado por cuatro bits que admite valores del cero al ocho. Debido a que este campo es de cuatro bits se pueden indicar valores más grandes a ocho. Si el valor del DLC es mayor que ocho se asume que la trama contiene ocho bytes.

Después sigue el campo de datos, el cual contiene información contenida entre cero y ocho bytes definido por el DLC.

Después sigue el campo de comprobación de redundancia cíclica (CRC); contiene un checksum de quince bits. El bit dieciséis en este campo es recesivo y cierra el checksum. El siguiente campo es el de reconocimiento, el cual no es colocado por el emisor de la trama, sino por un nodo diferente, que es capaz de reconocer la recepción de la trama directamente después del campo de datos. El espacio ACK también se transmite recesivamente por el emisor y es sobrescrito como dominante por un receptor hasta que el mensaje sea recibido correctamente. Solo así es confirmada la correcta recepción.

El campo de fin de trama, marca el final del mensaje y comprende siete bits recesivos. Después sigue un espacio de inter trama, el cual consta de una sucesión de tres bits recesivos para separar los mensajes sucesivos.

SoC Cyclone V

El SoC (sistema en un chip) Cyclone V es un dispositivo que consiste de dos partes distintas, una porción HPS (Hard proccesor system) y una porción FPGA.

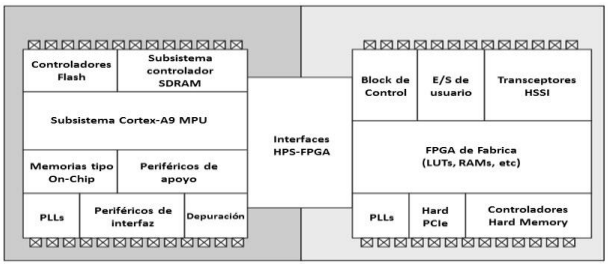

Figura 3 Diagrama a bloques del dispositivo Altera SoC FPGA.

En la figura 3 se muestra el diagrama a bloques de un SoC Altera, cuya principal característica es poseer un procesador ARM-A9, ya sea de núcleo único o de doble núcleo.

Mapa de direcciones del HPS

El mapa de direcciones del HPS especifica las direcciones de los esclavos, tal y como las ven el Microprocesador (MPU) y otros maestros en el HPS. La interfaz de cada periférico esclavo, tiene un rango de dirección dedicado en la región de periféricos. En la tabla 1 se muestra una lista de algunos de los periféricos del HPS.

IDENTIFICADOR DE ESCLAVO	DESCRIPCIÓN	DIRECCIÓN BASE	TAMAÑO
LWFPGASLAVES	Esclavos FPGA accesados a traves del puente lightweigth HPS-FPGA	0xFF200000	2 MB
EMAC 0	Ethernet Mac 0	0xFF700000	8 KB
EMAC 1	Ethernet Mac 1	0xFF702000	8 KB
SDMMC	SD/MMC	0xFF704000	4 KB
GPIO0	GPIO 0	0xFF708000	4 KB
GPIO1	GPIO 1	0xFF709000	4 KB
GPIO2	GPIO 2	0xFF70A000	4 KB
USB0	Registros del controlador USB 2.0 OTG 0	0xFFB00000	256 KB
USB1	Registros del controlador USB 2.0 OTG 1	0xFFB40000	256 KB
CAN0	Registros del controlador CAN 0	0xFFC00000	4 KB
CAN1	Registros del controlador CAN 1	0xFFC01000	4 KB
UART0	UART 0	0xFFC02000	4 KB
UART1	UART 1	0xFFC03000	4 KB

Tabla 1 Mapa de direcciones de la región de periféricos del HPS.

ISSN-En línea: 2410-3993

El modelo de programación para accesar a los periféricos del HPS de la tabla 1, es mediante la dirección base asignada a cada periférico, en la cual un cierto número de registros puede ser encontrado, de tal modo que se puede leer y escribir a un conjunto de estos registros, para modificar el comportamiento del periférico.

Controlador CAN

El HPS provee dos controladores CAN, para comunicación serial con el microprocesador ARM. Los controladores CAN cumplen con el ISO 11898-1. Estos controladores pueden almacenar hasta 128 mensajes, el acceso a estos mensajes por parte del procesador, es llevado a cabo a través de la interfaz de mensaje de la RAM. El controlador provee distintos tipos de modos de prueba, los cuales son utilizados principalmente para pruebas de funcionamiento del propio controlador.

Modo Silencioso

En modo Silencioso, el controlador CAN es capaz de recibir tramas de datos válidos, pero mantiene el pin CAN_TXD en alto.

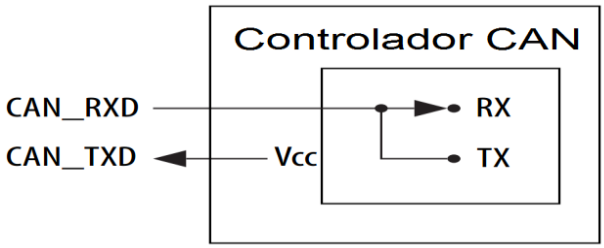

Figura 4 Controlador CAN en modo silencioso.

Modo Loopback

En modo Loopback, el controlador CAN trata sus propios mensajes transmitidos como mensajes recibidos y los guarda.

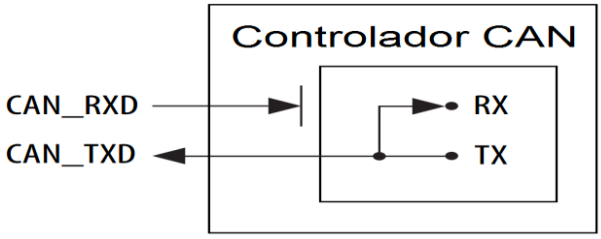

Figura 5 Controlador CAN en modo Loopback.

Modo Combinado

Es posible poner el controlador CAN en modo combinado; Loopback y Silencioso. El modo combinado puede ser utilizado para hacer pruebas del hardware del CAN, sin afectar otros dispositivos conectados al bus CAN. En este modo, el pin CAN_RXD es desconectado del controlador CAN y el pin CAN_TXD es mantenido en alto.

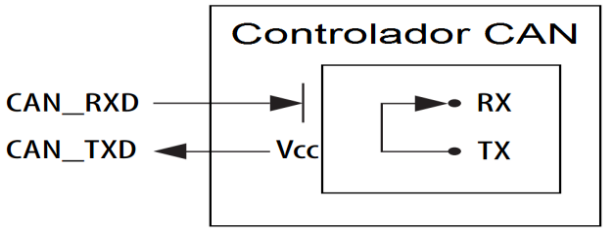

Figura 6 Controlador CAN en modo combinado.

Cuando se realiza un proyecto con nodos CAN se necesitan algunas configuraciones de hardware, tales como son las conversiones de interfaz. Estas incluyen las conversiones entre niveles TTL/CMOS a los niveles de transmisión para todos los bits de mensaje y las conexiones de hardware, por medio de conectores especiales. En el caso de llevar a cabo la implementación física entre dos o más nodos CAN, es necesario cumplir con los requerimientos antes mencionados, es por eso que se optó por utilizar el modo combinado, ya que se puede desarrollar el software que gestiona el funcionamiento del controlador sin la necesidad de agregar hardware extra.

Tiempo de bit nominal

Para poder transmitir y recibir satisfactoriamente objetos de mensaje a través del bus CAN, el proceso de transmisión y recepción debe ser controlado y sincronizado para hacer que el transmisor y el receptor trabajen en la misma base de tiempo para reducir posibles errores. Por tanto el tiempo de bit nominal (TNB=1⁄(bit_rate)), debe ser evaluado lo más exacto como sea posible y asegurar que cada bit pueda ser transmitido y recibido en un rango de tiempo que pueda ser tolerado por el bus CAN. Los parámetros de tiempo de cada TNB pueden ser configurados individualmente para cada nodo, creando una misma tasa de bit aun cuando los nodos CAN puedan utilizar diferentes fuentes de reloj. En la figura 7 se muestra la configuración del TNB.

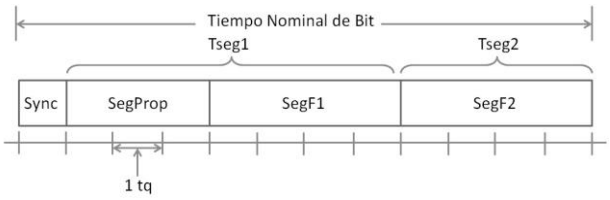

Figura 7 Configuración estándar de tiempo de bit.

Cada periodo de TNB está compuesto de cuatro segmentos, los cuales consisten de un número específico y programable de tiempo quanta (tq). La longitud del tq, la cual es la unidad básica de tiempo del TNB, se define en términos del reloj de entrada del controlador (〖clk〗 _CAN) y la tasa de transferencia del preescaler (BRP).

$$t_q = BRP/clk_{CAN} \quad (1)$$

Estos cuatro segmentos, son utilizados para hacer que el tiempo de bit se ajuste automáticamente al tiempo de bit normal, al agregar o remover algunos números de t_q para hacerlos coincidir con el tiempo de bit real y prevenir errores de tiempo.

Causados por las variaciones en la temperatura y el ambiente. El número total de t_q de los cuales está conformado el TNB se obtiene de la siguiente ecuación:

$$Num_{tq} = TNB/t_q \quad (2)$$

El segmento de sincronización no es programable y está fijado a 1t_q.

El Segmento de Propagación (SegProp) se define como.

$$t_{Prop} = 2(t_{bus} + t_{TX} + t_{RX}) \quad (3)$$

Donde t_{bus} es el tiempo de retardo de propagación a lo largo la longitud más grande del bus entre dos nodos, t_{TX} es el retardo de propagación de la parte transmisora de la interfaz física y t_{RX} es de la parte receptora.

Entonces se tiene que el número de t_q que contiene este segmento se calcula como:

$$SegProp = round\_up(t_{SegProp}/t_q) \quad (4)$$

Los dos segmentos de fase, SegF1 y SegF2 se obtienen de la siguiente ecuación:

$$SegF1 + SegF2 = Num_{tq} - SegProp - 1 \quad (5)$$

Si el resultado es 3 entonces: $SegF1 = 1$ y $SegF2 = 2$.

El ancho de salto de sincronización (SJW) ajusta el reloj de bit como sea necesario entre 1 y 4 t_q (como sea configurado) para mantener la sincronización con el mensaje transmitido.

El SJW se elige como el menor de 4 y SegF1.

A continuación se listan los requerimientos que se deben cumplir al término de los cálculos.

$$SegProp + SegF1 \geq SegF2$$
$$SegProp + SegF1 \geq t_{Prop}$$
$$SegF2 > SJW$$

Inicialización del controlador CAN

Para inicializar el controlador CAN, el microprocesador debe programar el registro de tiempo de bit del CAN (CBT) y configurar los objetos de mensaje que serán utilizados para la comunicación CAN, previo a la configuración de los objetos de mensaje es necesario inicializar la RAM de mensaje.

Una vez que el controlador CAN es inicializado, se sincroniza a si mismo con el bus CAN y comienza la transferencia de mensajes.

Los mensajes recibidos son almacenados con sus objetos de mensaje apropiados. El procesador central puede leer o actualizar cada mensaje en cualquier momento, utilizando los registros de interfaz de mensaje.

Archivo de imagen Linux

Se utilizó un archivo de imagen provisto por Altera, el cual contiene todos los elementos necesarios para correr Linux en la tarjeta de desarrollo.

Dicho archivo de imagen es vaciado en una tarjeta micro SD.

Es primordial generar un preloader y un device tree de acuerdo a las necesidades del proyecto y reemplazar los existentes en la tarjeta micro SD por medio de las herramientas de altera como se ve en la figura 8.

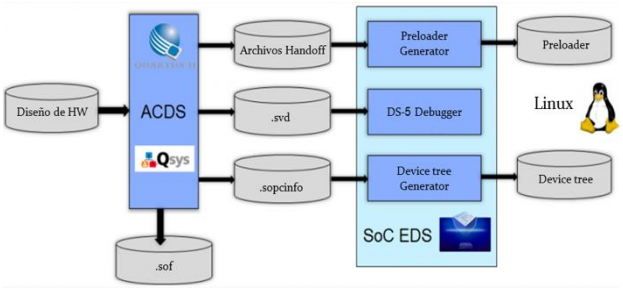

Figura 8 Esquema de diseño.

Configuración del HPS en Qsys

Como primer paso, fue necesario configurar adecuadamente el HPS mediante QSys.

Como se puede ver en la figura 9 se utilizó el controlador CAN1, el cual comparte pines con uno de los periféricos SPI y las entradas/salidas de propósito general 59 y 60.

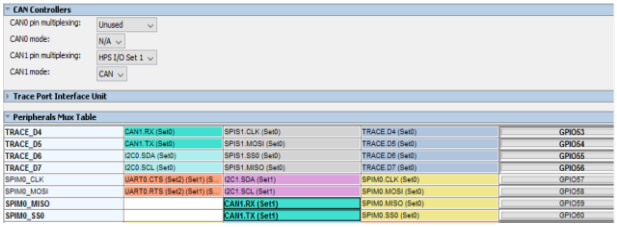

Figura 9 Configuración del controlador CAN.

Una vez que se realizó la configuración mediante QSys, se procedió a generar el código Verilog que describe estas configuraciones, además se instanciaron los periféricos utilizados a través de Quartus II.

```
.hps_0_hps_io_hps_io_can1_inst_RX     (hps_0_hps_io_hps_io_can1_inst_RX),
.hps_0_hps_io_hps_io_can1_inst_TX     (hps_0_hps_io_hps_io_can1_inst_TX),
```

Figura 10 Instanciación del controlador CAN.

Debido a que se utiliza una tarjeta de desarrollo prediseñada, cuenta con la limitante de que esta solo permite el acceso a uno de los controladores CAN, en este caso el CAN1, debido a que los pines dedicados al CAN0 son utilizados para otros dispositivos conectados a la tarjeta y es imposible el acceso a ellos.

ISSN-En línea: 2410-3993

El controlador CAN1 comparte pines con uno de los periféricos SPI de los cuales un par de ellos pertenece a un conector denominado LTC, lo cual brinda la posibilidad de comunicar el CAN1 al exterior. En la figura 11 se muestra con más detalle el multiplexaje de dichos pines.

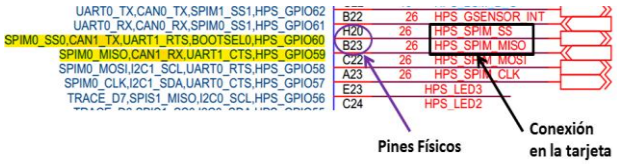

Figura 11 Pines compartidos del CAN1.

Generación del Preloader

El siguiente paso fue compilar el proyecto, lo cual da como resultado los archivos necesarios para la generación del preloader. En la figura 12 se muestra el diagrama de flujo para la configuración y generación del preloader.

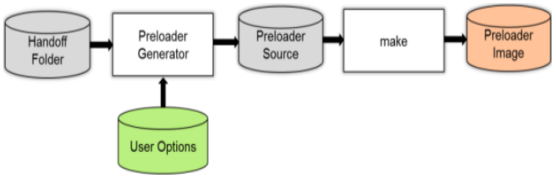

Figura 12 Diagrama de flujo para la generación del preloader.

Por medio de la herramienta EDS se ejecutó el editor bsp (board-suport-package) para la generación del preloader, el cual tiene como entradas los archivos del folder Handoff creado a través de Quartus II y las opciones de usuario, con las que se indica en donde se encuentran los archivos Handoff, el tipo de sistema operativo y la ruta donde se generan los archivos del preloader.

Entre los archivos que se generaron durante esta etapa, se encuentran algunos archivos de cabecera, en donde se indican los periféricos del HPS habilitados, como se muestra en la figura 13.

```
22  #define CONFIG_HPS_CAN0 (0)
23  #define CONFIG_HPS_CAN1 (1)
```

Figura 13 Archivo de cabecera generado.

Generación del device tree

El device tree es utilizado por el Kernel de Linux para determinar que driver de dispositivo cargar a la hora del inicio, por tanto fue necesario la modificación del device tree provisto por altera, para dar de alta el CAN1. En la figura 14 se muestra el diagrama de flujo para la generación del device tree.

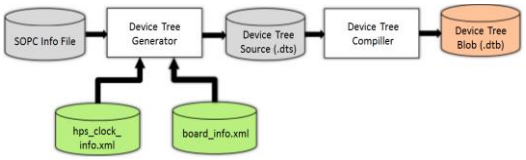

Figura 14 Diagrama de flujo para la generación del device tree.

Al momento de la compilación del proyecto se genera el device tree source, por tanto después de la compilación se procedió a la generación del device tree blob (DTB), el cual es la versión binaria del device tree source. Una vez generado DTB, se colocó directamente en la partición fat de la micro SD, junto con el archivo de configuración *rbf*, el cual contiene la información necesaria para configurar la tarjeta de desarrollo, lo cual es equivalente a cargar un archivo *sof* mediante Quartus II programmer.

```
can1: can1@ffc01000 {
    compatible = "bosch,d_can";
    reg = <0xffc01000 0x1000>;
    interrupts = <0 135 4>, <0 136 4>, <0 137 4>, <0 138 4>;
    clocks = <0x18>;
    status = "okay";
};
```

Figura 15 Device tree generado.

Cálculo de los parámetros del tiempo de bit

A continuación se muestran los parámetros utilizados para el cálculo del tiempo de bit:

Parámetro	Valor
CLK_{CAN}	100 MHz
Tasa de bit	1 MHz
Longitud del bus	40 mts
Tiempo de propagación del bus	5 ns/m
Retardo de transmisión	50 ns
Retardo de recepción	80 ns

Tabla 2 Parámetros para cálculo del tiempo de bit.

ISSN-En línea: 2410-3993

A continuación se muestran los parámetros utilizados para el cálculo del tiempo de bit:

Utilizando la ecuación (3) se tiene:

$$t_{SegProp} = 2(50ns + 80ns + 200ns) = 660ns$$
$$t_q = BRP/clk_{CAN}, \qquad tomando\ BRP = 9$$
$$t_q = 7/100MHz = 70ns$$

Utilizando la ecuación (2) se tiene:

$$1000ns/70ns \approx 14$$

Se ve que el tiempo de bit está formado aproximadamente por $14\,t_q$. Utilizando la ecuación (4) se tiene:

$$SegProp = 660ns/70ns = 9.42$$
$$\therefore\ SegProp = 10$$

Utilizando la ecuación (5) se tiene:

$$Fase1 + Fase2 = 14 - 1 - 10 = 3$$
$$\therefore Fase1 = 1\ y\ Fase2 = 2$$

De aquí que:

$$RJW = 1$$

Una vez obtenidos los resultados anteriores y de acuerdo a la figura 7 se tiene que:

$$Tseg1 = 10 + 1 = 11$$
$$Tseg2 = 2$$

En la tabla 3 se muestran los valores que se programan en el registro.

Parámetro	Valor calculado	Valor en el registro
BRP	7	6
Tseg1	11	10
Tseg2	2	1
SJW	1	0

Tabla 3 Parámetros calculados.

Generación del Código

Como se mencionó antes, se optó por utilizar el modo combinado del controlador CAN para realizar un primer acercamiento de lo que será la implementación del sistema para su uso en redes CAN existentes. El programa fue realizado en lenguaje C y compilado con la ayuda del Embedded Command Shell de EDS Altera Edition. En la figura 16 se muestra el diagrama de flujo del programa desarrollado.

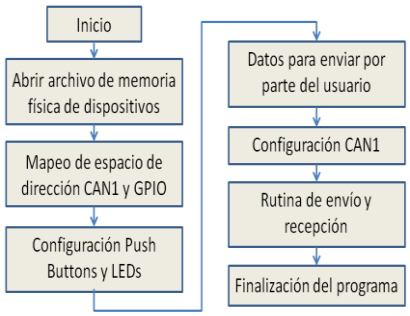

Figura 16 Diagrama de flujo del programa desarrollado.

Se utilizó el método de mapeo de memoria debido a que el desarrollo de un driver para interactuar directamente con el controlador CAN va más allá del objetivo de este trabajo. Para poder mapear en memoria el controlador CAN, es necesario primero abrir la memoria física de los dispositivos y después utilizar la función *mmap()* para mapear el espacio de dirección que ocupa el CAN1 y el GPIO. La manipulación de los registros de configuración del CAN1 y del GPIO1 se realizó por medio de las siguientes funciones de bajo nivel de Altera:

- alt_write_byte(dir_dest, byte)
- alt_read_byte(dir_fuente)
- alt_setbits_byte(dir_dest, byte)
- alt_clrbits_byte(dir_dest, byte)

A partir de estas funciones se implementaron otras para modificar los registros de configuración de una manera más ágil.

ISSN-En línea: 2410-3993

Una vez mapeados los periféricos que se utilizaron, se realizó la configuración de dos GPIOs conectadas a pulsadores, uno para dar la orden de envío de trama y otro para finalizar el programa, además de configurar 4 GPIOs conectadas a LEDs para reflejar el contenido del campo de datos de la trama CAN recibida, el número de bits a enviar se acoto por comodidad debido a que la tarjeta de desarrollo cuenta con 4 LEDs conectados directamente al HPS. El programa pide al usuario que ingrese datos de 4 bits que son los que se utilizaran en el campo de datos de la trama, una vez que se ha almacenado esta información, es utilizada para configurar el controlador CAN, después el programa entra a la rutina de envío y recepción, en donde si el usuario presiona un pulsador, la trama de datos es enviada y una vez recibida se reflejan los datos recibidos en los LEDs, si el otro pulsador es presionado se finaliza el programa.

Ya que al monitorear los registros de configuración solo se pueden imprimir en formato decimal o hexadecimal, se implementó una función para convertir los datos hexadecimales a binarios y que fuera más claro ubicar los campos de bits modificados.

En la figura 17 se muestra con más detalle la configuración del controlador CAN1.

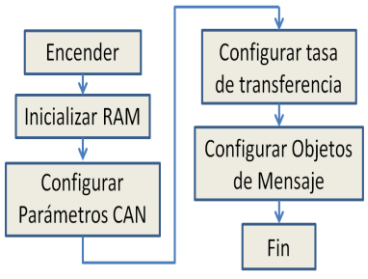

Figura 17 Configuración del controlador.

Al encender se inicializa la RAM de mensaje, mediante este proceso todos los objetos de mensaje son puestos a cero. Después se configuran los parámetros del CAN, en esta etapa se define el modo de funcionamiento combinado. Los parámetros obtenidos al calcular los segmentos de TNB son colocados a continuación en el registro de tiempo de bit. Como último paso fue necesario configurar dos objetos de mensaje uno en modo recepción y uno en modo transmisión, y se finaliza el proceso de configuración del controlador CAN.

Conclusiones

Se logró configurar adecuadamente el controlador CAN y utilizarlo en modo combinado para enviar tramas CAN personalizadas y evaluar los datos enviados por medio del propio controlador. El siguiente paso es diseñar un circuito impreso el cual contenga un transceptor CAN y pueda ser conectado a la tarjeta de desarrollo para enviar tramas CAN a una red existente. Las modificaciones necesarias al programa son mínimas ya que bastara con omitir la configuración en modo combinado.

Referencias

Marco Di Natale, Haibo Zeng, Paolo Guisto, Arkadeb Ghosal. (2012). Understanding and Using the Controller Area Network Comunication Protocol. USA: Springer.

Altera Corporation. (2015). Cyclone V Hard Processor System Technical Reference Manual. USA: Altera.

Microchip Technology Inc. (2001). Understanding Microchip's CAN Module Bit Timing. USA: Microchip Technology Inc.

ISSN-En línea: 2410-3993