Aplicación Práctica de SOTR con bus CAN

Miguel Roselló Reinés, Bernat Mayol March

Enginyeria Tècnica en Informàtica de Sistemes

***Sumario* — En este documento se describe la programación de un reloj despertador con indicador de temperatura mediante un sistema operativo de tiempo real (SOTR) distribuido en varios nodos comunicados mediante el protocolo CAN. El microcontrolador utilizado en los nodos es el dsPIC30F4011 de Microchip.**

I. Introducción

Un sistema operativo (SO) es un conjunto de programas que se encarga de gestionar los recursos de un ordenador y proporcionar una interfaz con el usuario. Algunos sistemas operativos dan soporte a sistemas concurrentes, en el sentido que permiten ejecutar varias tareas (programas) de forma simultánea.

Un sistema operativo de tiempo real (SOTR en adelante) es pues un SO que proporciona mecanismos para que las tareas puedan ejecutarse concurrentemente respetando sus plazos temporales. El núcleo gestiona la ejecución de las tareas, el uso de la memoria, y provee funciones de entrada/salida de datos a bajo nivel. El SOTR que se va a utilizar se denomina SALVO OS. Este sistema operativo es apropiativo, por consiguiente son las tareas que se ejecutan las encargadas de retornar el control al procesador.

El protocolo de comunicaciones utilizado es el bus CAN. Es un protocolo basado en una topología bus para la transmisión de mensajes y soporta control distribuido en tiempo real. CAN es un protocolo orientado a mensajes, es decir, la información que se va a intercambiar se descompone en mensajes, a los cuales se les asigna un identificador y se encapsulan en tramas para su transmisión. Todo mensaje llega a todos y cada uno de los nodos, proporcionando una excelente consistencia de datos a nivel de mensajes. Cada mensaje tiene un identificador único dentro de la red, con el cual los nodos deciden aceptar o no dicho mensaje. En la práctica utilizaremos como mínimo dos nodos, uno para la gestión de interfaz con el usuario y otro para la gestión del reloj y temperatura. No obstante, el diseño realizado permite la interconexión de más de un nodo gestor de interfaz de usuario, incrementándose así las aplicaciones prácticas del sistema.

El dispositivo utilizado es un modulo iCM4011 (dsPIC en adelante) conectado a una placa UIB-PC-104 (UI en adelante). Se utilizará la pantalla LCD de la placa UI para mostrar la información. En modo normal, el sistema presentará la hora y la temperatura manteniendo el formato que se muestra en la Fig. 1. Los datos se introducirán mediante el teclado de la placa UI, organizándose éste tal y como se aprecia en la Fig. 2. Para la simulación del sensor de temperatura usaremos el conversor A/D y el potenciómetro R7 de la placa UI. Los valores de temperatura estarán comprendidos entre -19.9ºC y +99.9ºC.

Fig. 1 Ejemplo LCD nodo A

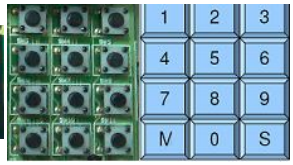

Fig. 2 Ejemplo del teclado

Fig. 3 Ejemplo LCD nodo B

II. Objetivos

El objetivo es la implementación de un sistema multinodo, que implique el intercambio de mensajes entre nodos, también denominados tramas en este documento.

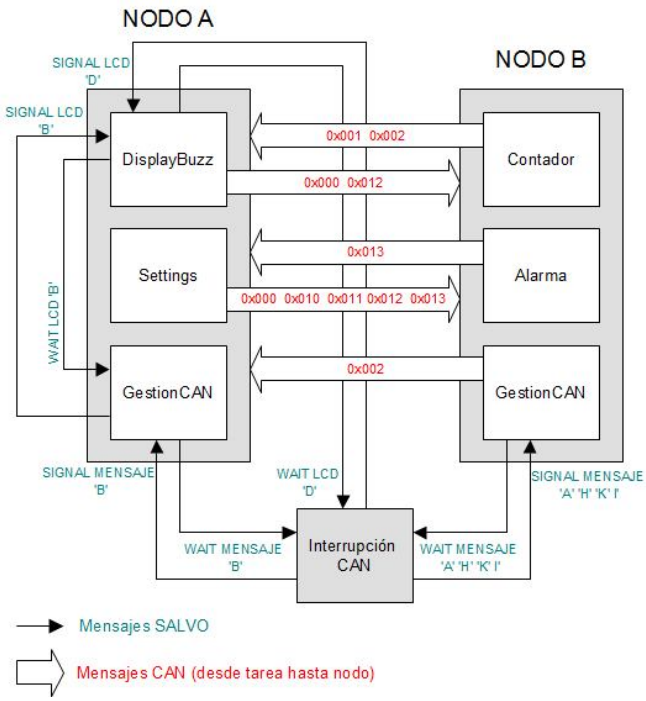

Fig. 4 Esquema básico

Para este fin se implementará un reloj despertador mediante dos tipos de nodo, A y B. El tipo de nodo A realizará tareas de interfase con el usuario, mientras que el tipo de nodo B gestionará la hora, almacenará y disparará la alarma. El sistema admitirá múltiples nodos de tipo A, sin embargo, solo admitirá un único nodo de tipo B por red. El nodo de tipo B se distinguirá de los demás nodos porque aparecerá el texto

"Time Server" en la primera línea del LCD, como se aprecia en la Fig. 3.

A nivel local, el objetivo será la comunicación entre las tareas que se ejecutarán dentro de un mismo nodo sobre el sistema operativo SALVO, anteriormente descrito.

El comportamiento del sistema, es el descrito a continuación.

A. Programación de Alarma

En modo normal de funcionamiento, mediante una primera pulsación de la tecla "M" se entrará en modo de programación de alarma (A), apareciendo el texto "Alarm HH:MM off" en la primera línea de LCD, como vemos en la Fig. 5. En caso de estar activada, aparecerá "on" en lugar de "off" en dicha línea. El sistema esperará la pulsación de cuatro dígitos válidos correspondientes a la hora de alarma, la pulsación de la tecla "S" activará o desactivará la misma, actualizándose el último campo de la primera línea. Mediante una nueva pulsación de la tecla "M" se entrará en el modo de programación de la hora del reloj (B). En caso de no haber introducido 4 dígitos en el anterior modo de programación de alarma, ésta no se verá alterada.

Fig. 5 Programación de Alarma

B. Programación de Hora

Aparecerá en este modo el texto "Clock Set" en la primera línea del LCD, la Fig. 6 representa la visualización de la programación de hora. Nuevamente el sistema esperará la pulsación de cuatro dígitos. Una tercera pulsación de la tecla "M" retornará el sistema al modo normal de visualización de hora y temperatura. Como en el caso anterior, la no introducción de cuatro dígitos válidos mantendrá la hora inalterada.

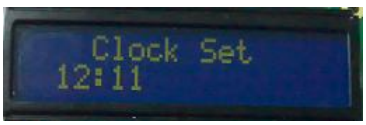

Fig. 6 Programación de Hora

III. IMPLEMENTACIÓN

Se dispone de dos dispositivos interconectados, llamados nodo A y nodo B, que mediante el SOTR se intercambian información a través del protocolo de comunicaciones CAN. El nodo A se encarga de la gestión de interfaz de usuario y el nodo B de la gestión del reloj, la alarma y temperatura.

A. Configuración CAN

Es evidente que si se debe realizar una comunicación entre diferentes nodos, éstos deben utilizar una configuración del canal (tipo de trama, formato de trama, tiempo de bit) idéntico. En cuanto al tiempo de bit se ha configurado el canal de tal forma que:

$$t_{SYNCSEG} = 1\ T_Q \quad (1)$$

$$t_{TSEG1} = 6\ T_Q \quad (2)$$

$$t_{TSEG2} = 3\ T_Q \quad (3)$$

$$SJW = 3\ T_Q \quad (4)$$

El modulo CAN tiene 3 buffers de recepción. No obstante, uno de los buffers de recepción se dedica continuamente a escuchar el bus en espera de mensajes entrantes (MAB). Durante todo el desarrollo únicamente se utiliza el búfer de recepción RXB1, así forzamos a que todos los mensajes sean cargados en un solo búfer de recepción para tener un mayor control e indicamos que el búfer RXB0 está lleno tras cada interrupción. Adoptamos pues una filosofía de funcionamiento de tipo BasicCAN, con un único buzón con una mascara y varios filtros.

Configurando el bit RX1IE = 1 se generará una interrupción cuando se reciba un mensaje. Una vez un mensaje válido ha sido recibido por el MAB, el campo de identificación del mensaje es comparado, primeramente con la máscara, y si es válido, es comparado posteriormente con los valores de los filtros. Si hay una coincidencia, ese mensaje será cargado en el búfer de recepción RXB1.

El contenido del mensaje queda determinado únicamente por su identificador CAN, teniendo en cuenta que cuanto más bajo es el identificador del mensaje mayor prioridad tiene.

Por último se han definido dos máscaras de aceptación de mensajes, una para cada nodo (0x7FC nodo A y 0x7EC nodo B). La interrupción CAN se encarga de gestionar la recepción de mensajes, una vez aceptados, es el SOTR quien se encarga de gestionar el evento.

B. Tareas

Cada tarea tiene asignada una prioridad estática, es decir, que no cambia durante la ejecución de la tarea. En este caso, la asignación de prioridades está determinada por la importancia de cada tarea. Las tareas ejecutadas por un tipo de nodo diferirán de las tareas ejecutadas por el otro tipo de nodos.

Es importante remarcar que el sistema operativo utilizado en la práctica se ha empleado con una licencia de estudiante que sólo permite la definición de tres tareas por cada nodo. A continuación se describe el nombre, el tipo de nodo en que se ejecuta cada una y una breve descripción de la misma:

1) DisplayBuzz, nodo A:

Es la tarea con mayor prioridad del nodo A. Tiene la prioridad mayor ya que se encarga de actualizar los datos de la pantalla LCD cada 0.5 segundos. En primer lugar inicializa los datos de la pantalla con la información almacenada en el nodo B, mediante la función transmetreRemota(). La tarea solo puede recibir dos tipos de mensajes, el disparo de la alarma desde la interrupción CAN y la actualización de la temperatura y hora desde la tarea GestionCAN() (los segundos se muestran con el parpadeo del carácter ":").

Tras recibir un mensaje de disparo alarma se activa el buzzer y cabe la posibilidad de detectar que se ha pulsado la tecla "S", en este caso, se transmitirá una trama de tipo 0x000 mediante la función transmetreRemotaStop() y en consecuencia el nodo B dejara de transmitir disparos de alarma.

2) Settings, nodo A:

Es la tarea con menor prioridad del nodo A. Se dispara tras la pulsación de la tecla "M". Inicialmente solicita al nodo B los datos actuales de la hora, alarma y estado de alarma mediante una llamada a la función transmetreRemota().

Se encarga de la gestión de configuración de la alarma mediante la lectura por teclado de los datos introducidos por el usuario. El feedback se puede observar por la pantalla LCD.

Posteriormente se transmiten los nuevos datos al nodo B mediante la función transmetreNormal() usando los identificadores 0x010 y 0x011. El primero se transmite en caso de haberse modificado la alarma, el segundo en caso de haberse modificado la hora. Durante el proceso es posible cambiar el estado de activación de la alarma, mediante la función transmetreSetAlarm().

3) GestionCAN, nodos A y B:

Se encarga de la recepción de mensajes CAN de los dos tipos de nodos. Es la tarea receptora de mayor número de mensajes por parte del sistema operativo SALVO y desde ella se envían mediante la función transmetreNormal() los mensajes CAN tipo 0x002 de actualización de hora, alarma y alarma activa como consecuencia de una inicialización, un cambio de hora, alarma o estado de alarma.

4) Contador, nodo B:

Es la tarea más importante y con mayor prioridad del nodo B, puesto que representa la tarea de reloj propiamente dicha. En primer lugar inicializa todos los nodos de tipo A enviando un mensaje de tipo 0x002 a la red CAN, mediante la función transmetreNormal(). Este comportamiento se realizará periódicamente coincidiendo con el cambio de hora o minuto.

Se encarga además, de realizar la conversión de la temperatura de A/D, dato que se transmite cada medio segundo mediante el envió de un mensaje de tipo 0x001 usando la función transmetreNorrmal().

5) Alarma, nodo B:

Es la tarea con menor prioridad del nodo B. Se encarga de verificar si la hora programada como alarma coincide con la hora actual del reloj y si la variable *alarmaActiva* vale 1. En ese caso se envía un mensaje de tipo 0x003 mediante la función transmetreNormal(). Este comportamiento se repetirá periódicamente hasta que cambie la hora o se reciba un mensaje de tipo 0x000.

C. Comunicaciones CAN (Nivel de aplicación)

La recepción de mensajes CAN se realiza por interrupciones. Una vez recibido un mensaje válido, el servicio de interrupción se encargará de procesarlo y generar un mensaje a la tarea correspondiente (GestionCAN o DisplayBuzz), o bien actualizará las variables globales *hora*, *minuto*, *alarmaHora*, *alarmaMinuto* y *alarmaActiva.*

Por lo que respecta a la transmisión de mensajes se han implementado cuatro funciones diferenciadas según el tipo de información a transmitir. Las diferentes funciones son:

1) *transmetreNormal:* Transmisión de un mensaje CAN de tamaño e identificador indicados.

2) *transmetreRemota:* Mensaje con identificador 0x012 transmitido por parte de un nodo de tipo A, solicitando información para la inicialización o para la actualización de datos. Este mensaje es consecuencia de la inicialización o de un cambio de hora o alarma. Implica una respuesta por parte del nodo B de una trama de tipo 0x002.

3) *transmetreRemotaStop:* Mensaje con identificador 0x000 transmitido por parte de un nodo de tipo A, solicitando el cese de transmisión de mensajes de alarma 0x003 desde el nodo B.

4) *transmetreSetAlarm:* Mensaje con indetificador 0x013 transmitido por parte del nodo A, que indica el estado en que debe quedar la alarma (on/off). Implica una respuesta por parte del nodo B de una trama de tipo 0x002.

TABLA I
FORMATO DE MENSAJES CAN

Ident.	Bytes	Contenido
0x000	0	(No contiene datos)
0x001	1	Temperatura
0x002	5	Hora, Alarma, Estado Activación Alarma
0x003	0	No contiene datos
0x010	2	Valor de Alarma
0x011	2	Valor de Hora
0x012	0	(No contiene datos)
0x013	1	Estado Activación Alarma

D. Interrupciones y Hardware

Los retardos empleados en las tareas que se ejecutan bajo SALVO, se logran mediante la llamada a la función OS_Delay. Si bien, debido a la implementación propia del sistema operativo, ha sido necesario programar un timer en el hardware del dsPIC con un periodo de 250ms, que realiza una llamada a la función OSTimer del SALVO.

Para la gestión del teclado, se ha programado una interrupción mediante un timer que se dispara cada 100ms, también a nivel de hardware, que se encarga de leer la tecla pulsada, almacenar su valor en la variable global 'llegit_teclat' y de actualizar el valor de la variable también global 'tecla_pitjada' a uno si se esta pulsando una tecla o a cero en caso contrario.

Dado que el LCD es un recurso compartido, cabe pensar que debe estar controlado su uso para evitar conflictos entre tareas concurrentes, DisplayBuzz y Settings en este caso. El motivo de la no gestión del LCD como recurso compartido es que el SALVO es un sistema operativo apropiativo y que ambas tareas han sido programada de tal forma que no devuelven el control a la CPU hasta que se ha finalizado el uso del LCD, por tanto una y solo una de ellas accederá al recurso a la vez.

También cabe añadir que para conservar la consistencia de datos del sistema, las variables *hora*, *minuto*, *alarmaHora*, *alarmaMinuto* y *alarmaActiva* se actualizan dentro de la rutina de servicio de la interrupción CAN justo en el momento de la recepción de un mensaje de tipo 0x002, puesto que si se actualizasen dichos valores en la tarea GestionCAN, como

seria lo mas lógico, la tarea Settings no actualizaría sus datos tras una petición de transmetreRemota, debido a la forma en que la tarea Settings ha sido diseñada, mencionada en el párrafo anterior.

Derivado del hecho de que se actualicen de los valores de las variables mencionadas en el párrafo anterior dentro de una rutina de servicio de interrupción, surge la posibilidad de la recepción consecutiva de un mensaje CAN de tipo 0x002 y otro de tipo 0x001, provocando en el peor de los casos, la modificación de dichas variables durante el proceso de salida por pantalla, en la tarea DisplayBuzz. Para evitarlo, si la tarea Contador envía un mensaje de tipo 0x002 debido a un cambio de hora o minuto, no se enviara el correspondiente 0x001, evitando así dicho solapamiento.

Dado que el servicio de interrupción del teclado es lento y puede interferir al correcto funcionamiento de las funciones de salida de datos por LCD, haciendo aparecer caracteres incorrectos, se ha añadido una variable global inhibe teclado que impide la ejecución del dicho código en caso de estar ejecutándose la tarea DisplayBuzz y estar ésta realizando llamadas a alguna de las funciones de las librerías de acceso al LCD empleadas.

E. Mensajes entre Tareas

Para la comunicación entre tareas se ha optado por el intercambio de mensajes vía SALVO. Se han definido dos tipos de mensajes: los mensajes para las comunicaciones con el CAN (MENSAJE OSECBP) y los mensajes relacionados con el LCD y disparo del buzzer (LCD OSECBP). A continuación se describen los tipos de mensajes entre tareas, así como las tareas que implican:

1) 'B': El carácter 'B' se utiliza para indicar que se debe actualizar la pantalla LCD, mostrando el parpadeo de medio segundo y la temperatura. La tarea GestionCAN() envía el mensaje 'B' a DisplayBuzz().

2) 'D': El carácter 'D' se utiliza para disparar la alarma. Desde la interrupción del CAN se envía el mensaje 'D' a DisplayBuzz().

3) 'A': El carácter 'A' se utiliza para programar la hora de la alarma. Desde la interrupción CAN y la tarea GestionCAN() se transmite y se recibe el mensaje 'A'.

4) 'H': El carácter 'H' se utiliza para programar la hora del reloj. Desde la interrupción CAN y la tarea GestionCAN() se transmite y se recibe el mensaje 'H'

5) 'I': El carácter 'I' se utiliza para solicitar esporádicamente la hora y temperatura a través de GestionCAN().

6) 'K': El carácter 'K' se utiliza para modificar el estado de la alarma en cualquier momento a través de la tarea *GestionCAN().*

La transmisión y recepción de mensajes entre tareas de un mismo nodo se realiza mediante las funciones del sistema operativo OSSignalMsg y OS_WaitMsg, respectivamente.

IV. CONCLUSIONES

Estamos satisfechos con el resultado obtenido ya que inicialmente se planteó el diseño para dos únicos nodos (display y servidor de tiempo) y se ha conseguido adaptar para tener múltiples nodos A con un solo servidor de tiempo.

Durante la asignación de prioridades de las tareas del SOTR tuvimos ciertas dificultades con la transmisión y recepción de mensajes CAN. Tuvimos que rehacer las prioridades de los tipos de mensajes CAN según su importancia.

Debemos también comentar los continuos problemas de las placas debido a la modificación de los timers internos del dsPIC. Dichos problemas reducían muy notablemente la velocidad de las placas y producían errores continuos en nuestros programas.

Una posible aplicación en el mundo real de este sistema podría ser la distribución a lo largo de diferentes estancias (separadas entre ellas una distancia no superior a la permitida por el tiempo de bit del bus CAN) de terminales reloj-alarma (nodos A). Incluso sería posible ampliar el tipo de alarmas de una manera sencilla para señalizar eventos esporádicos, como emergencias, u otros eventos periódics. Otra aplicación podría ser la distribución de la hora y la temperatura y otros posibles parámetros de interés en un vehículo o medio de transporte, lo cual se haría inhibiendo la programación de la alarma por parte de los nodos A normales, y permitiendo que únicamente un nodo A especial pudiera programarla o dispararla al llegar a un destino o momento interesante.

AGRADECIMIENTOS

Esta asignatura ha sido impartida por los profesores Guillermo Rodríguez-Navas González y Julián Proenza Arenas.

REFERENCIAS

[1] A. Burns, A. Wellings. *Sistemas de Tiempo Real y Lenguajes de Programación*, AddisonWesley, 2003

[2] (1999-2008) Pumpkin, Inc. Home of Salvo ™. The RTOS that runs in tiny places. [Online]. Available: http://www.pumpkininc.com/

[3] *DsPIC30F4011/4012 Data Sheet, High Perfomance, Digital Signal Controllers*, Microship Tecnology In 2005.

[4] (2008) Ingenia Motion Control Solutions, digital servo drives and software for high precision brushless and DC. [Online]. Available: http://www.ingenia-cat.com/

[5] (1979 - 2008) Ingenia ICM4011 dsPIC [Online]. Available: http://www.phaedsys.com/principals/ingenia/inicm4011.html

[6] (1979 – 2008) UIB-PC104 Development Kit for DsPIC modules [Online].Available:http://www.phaedsys.com/principals/ingenia/inuibpc104.html

Práctica realizada en la asignatura *Sistemes Encastats.*
Profesores: Julián Proenza y Guillermo Rodríguez-Navas.

Miguel Rossello Reinés. Estudiante de la titulación de Ingeniería Técnica en Informática de Sistemas en la Universitat de les Illes Balears (UIB).

Bernat Mayol March. Estudiante de la titulación de Ingeniería Técnica en Informática de Sistemas en la Universitat de les Illes Balears (UIB). Trabaja actualmente en Informallorca S.L, empresa dedicada al sector turístico.