Seminario: El protocolo CAN y su uso en redes de tiempo real.

Luis Sánchez Velasco y Juan Antonio Martín Galicia

Diciembre 2016

1. Introducción

Empezó a desarrollarse en 1983 por BOSCH para comunicar los distintos dispositivos de un coche. Se trata de un protocolo multimaster que presenta una arquitectura en bus, por el cual se envían todos los mensajes de forma que los nodos que quieran acceder a los paquetes de información que se envíen por el pueden hacerlo.

Lo que resultó atractivo de este protocolo era principalmente que era un modelo sencillo, barato y que además presume de tener una alta resistencia a las interferencias electromagnéticas y la capacidad de recuperación en caso de que se produzcan errores; aunque en la actualidad es considerado un protocolo poco eficaz si lo que se pretende es asegurar que ciertas funciones críticas se realicen en un tiempo máximo; es decir, un funcionamiento a tiempo real.

2. El estándar CAN

El protocolo de comunicaciones CAN (ISO-11898) especifica cómo deben de ser las capas física y de enlace del modelo OSI , las cuales explicaremos brevemente a continuación:

2.1. Capa física

Como se mencionó anteriormente, los diferentes nodos del sistema están conectados con una topología de bus serie, y el número máximo de elementos que podemos comunicar depende de la versión de CAN que se use (tenemos tanto CAN como Extended CAN) y por otro lado, la separación máxima que puede haber entre los nodos más distantes del sistema debe de estar acotada, para que se pueda considerar que todos los nodos reciben la información de forma aproximadamente simultánea y para que haya una probabilidad de colisión aún menor.

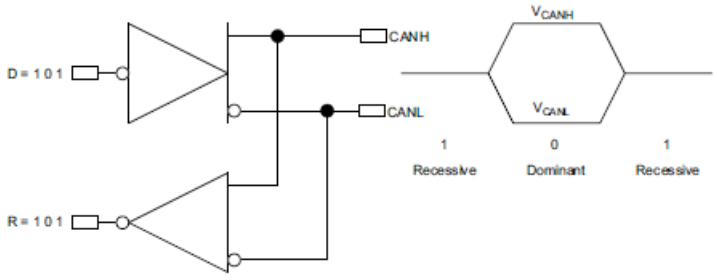

Figura 1: Lógica invertida del bus CAN

Por otro lado, los nodos deben de estar sincronizados en todo momento para que sean capaces tanto de leer correctamente los mensajes como de enviarlos

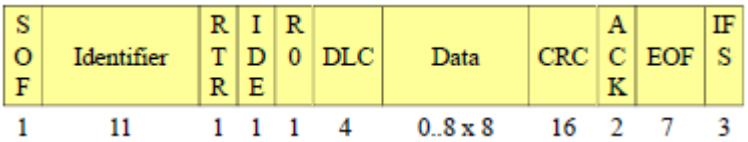

Figura 2: Frame CAN

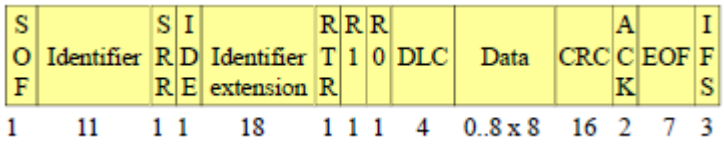

Figura 3: Frame ECAN

y que los demás los reciban correctamente. Para facilitar el mantenimiento de esta sincronización que se conoce como bit stuffing, que consiste en añadir ceros o unos para forzar transiciones (cada 5 unos consecutivos debe de añadirse un cero, mientras que cada 5 ceros debe de añadirse un uno), que son precisamente las que permiten a los nodos resincronizarse. Esta técnica es transparente a la capa de enlace, pues cada nodo en su capa física elimina estos bits antes de pasarlos a la capa de enlace, pues esta funcionalidad se explota en la capa física.

Otro detalle importante sobre el protocolo es que los 1s son recesivos mientras que los 0s son dominantes, lo cual quiere decir que si una estación trata de escribir un 1 y al mismo tiempo otra estación intenta poner un 0 sobre la linea se verá un 0 (podemos ver esto a nivel lógico en la figura 1). La utilidad de esto se verá a continuación, cuando expliquemos cómo funciona la capa de enlace.

2.2. Capa de enlace

Anteriormente resaltamos que la sincronización de los nodos es muy importante, esto es así porque en el protocolo CAN los nodos acceden al medio de forma simultánea. Periódicamente, todos ellos intentan escribir sobre el bus el frame que necesiten enviar; en la figura 1 podemos ver el frame que enviaría el protocolo CAN y en la 2 su versión extendida o ECAN.

Justo después del bit SOF (Start Of Frame), los nodos tratarán de escribir su identificador, leyendo también lo que se acaba escribiendo sobre el bus bit a bit. Esto se hace de esta manera porque los identificadores no sólo se usan para identificar de forma unívoca a todos los componentes de la red, también se usa para asignar prioridades a unos nodos frente a otros, de forma que los nodos con un identificador menor tienen mayor prioridad. Entonces, tras tratar de escribir un bit en el medio un nodo también leerá lo que se ha acabado escribiendo sobre la línea y en el momento que detecte dos valores diferentes dejará de in-

tentar escribir y pasará a modo sleep o lectura, dependiendo de su configuración.

Para que esto último quede completamente claro lo ilustraremos con un ejemplo de tres nodos intentando acceder al medio de forma simultánea: nodo A con ID 11001000101, nodo B con ID 10001000101 y nodo C con ID 10001000100. En primer lugar, los 3 nodos escribirán un 1 en la línea, y como no hay ninguno escribiendo un 0 el 1 se mantiene. A continuación los nodos B y C escribirán un 0 en la línea, el 1 que intenta transmitir el nodo A no tendrá efecto porque como se dijo antes el 0 es el bit dominante; tras detectar que se ha escrito un valor diferente en la línea el nodo A dejará de intentar acceder al medio para no interferir y los dos nodos restantes al diferir sólo en el último bit escribirán con éxito los 10 primeros bits de su identificador hasta llegar al último de ellos, el último bit que se escribirá en la línea será un 0 por el mismo motivo que se explico antes y el nodo C habrá ganado entonces el acceso al medio.

Entonces, tras esta competición por el medio sólo tendremos un nodo escribiendo, que rellenará el resto de campos como el DLC que incluye la longitud del campo de datos y el mismo campo de datos. El resto de nodos sólo participarán escribiendo en el campo ACK para indicar que se ha recibido el paquete (no se comprueba que esté libre de errores)

3. Corrección y detección de errores

Como se mencionó anteriormente una de las cosas más atractivas de CAN era su capacidad para detectar errores, y lo realiza de forma eficaz de 5 maneras diferentes, tres se realizan sobre el frame y las otras dos bit a bit.

Una vez se ha recibido el frame completo se pueden comprobar ciertos campos del mismo, como su CRC comparándolo con el campo de datos y el ACK; pero además de esto el frame contiene campos que deben de estar permanentemente rellenos con bits recesivos, que obviamente también son comprobados.

Además de estas tres formas de detectar errores a nivel de frame existen dos más que operan a nivel de bit. Primero, si un nodo escribe el bus **después del arbitraje** y lee del mismo un bit contrario se genera un error. Finalmente, los errores producidos en el bit stuffing también se toman en consideración. En caso de que se detecte cualquiera de estos 5 errores el mensaje es descartado, y cada nodo contabiliza los errores que producen al transmitir para que cuando alcance un cierto valor umbral pierdan la capacidad de transmitir hasta que el administrador intervenga.

En la figura 4 podemos apreciar el diagrama de estados de cada uno de los nodos del sistema, teniendo en cuenta que TEC es el número de errores de transmisión y REC de recepción. Estos contadores aumentan cuando el error se detecta y se decrementa por cada transmisión y recepción que se haga con éxito.

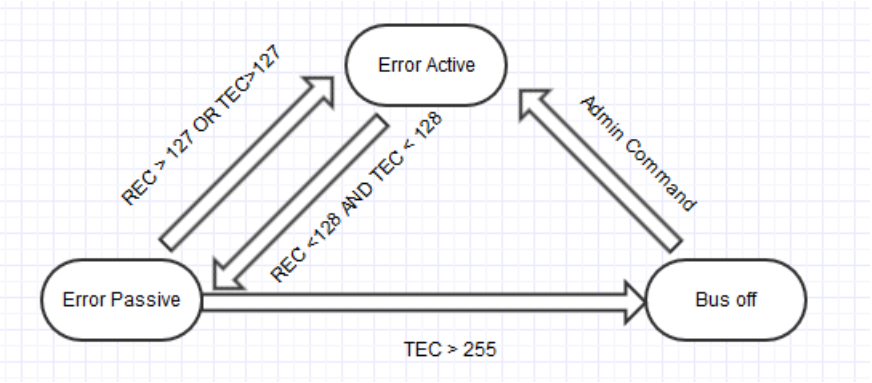

Figura 4: Diagrama de estados

4. Introducción al análisis temporal de CAN

Anteriormente comentamos que CAN tiene bastantes puntos fuertes, entre ellos su simplicidad y bajo coste; sin embargo esto no implica que pueda ser utilizado para todo tipo de aplicaciones. En este apartado estudiaremos si es posible el uso de este protocolo para aplicaciones en tiempo real, y para ello resulta necesario un análisis temporal de CAN con objetivo de calcular el tiempo de retardo de un mensaje en el peor caso posible para asegurar así un límite superior en el mismo retardo. El primer análisis fue llevado a cabo en 1994, el análisis de Tindell, pero años más tarde se demostró que este análisis era incorrecto porque partía de suposiciones que no siempre se daban, las cuales se mostrarán durante este documento al igual que una versión corregida de este análisis.

Antes del trabajo realizado por Tindell, el uso del protocolo CAN era bastante bajo debido a la extensa experimentación que había que llevar acabo para asegurarse de que los mensaje cumplían los límites temporales. Se estima un crecimiento desde el 30 % hasta el 80 % en su uso tras este primer análisis.

4.1. Análisis del peor caso

El estudio del peor caso de retardo normalmente comienza por la identificación del caso más deficiente en el envío de tramas para poder calcular el máximo retardo asociado al caso en cuestión. Empezaremos por identificar cual sería la trama con más posibilidades de causar un posible fallo en los tiempos del sistema. Esta será aquella con un mayor número de datos, pues esta tomará un mayor tiempo de transmisión (hay que tener en cuenta que el peor caso posible para el protocolo CAN no se conforma sólo de esta trama, después tendremos que considerar el hecho de que tenemos un bus único compartido por todos los nodos y que se produce un arbitraje del mismo que bloquea temporalmente los

nodos con menor ID o lo que es lo mismo, menor prioridad).

Para detectar cuál es la trama más larga no tendríamos que contar sólo con el mayor volumen de datos que se envía dentro del campo "data", también tendremos que tener en cuenta el bit stuffing y averiguar cuál sería la trama que añadiese más bits de este tipo, que obviamente sería la siguiente:

11111000011110000111 1...

Tras hacerse el procedimiento *stuffing bit* a la cadena anterior, insertando un bit de la polaridad inversa cada vez que se encuentran cinco bits de cierta polaridad seguidos obtenemos:

1111100000111110000...

Es decir, debido a que la primera trama de cinco unos seguidos introdujo un stuffing bit con un 0, la siguiente cadena de cuatro ceros también se ha visto obligada a generar un bit extra, provocando así que se genere siempre un quinto bit.

Entonces, siendo C_m el tiempo de transmisión del mensaje, τ_{bit} el tiempo de transmisión de un único bit, g la longitud en bits del identificador inicial (si se usan identificadores de once bits g valdría 34, mientras que usándolos de veintinueve bits valdría 54) y s_m el número de bits enviados en el campo de datos podemos concluir que:

$$C_m = (g + 8s_m + 13 + ExtraBit)\tau_{bit} \tag{1}$$

Siendo ExtraBit la cantidad máxima de bits que se añadiría a causa del bit sttufing, que como vimos antes se obtendría de dividir la longitud de la parte del mensaje a la cual se lo pueden añadir este tipo de bits (la parte inicial y los datos) entre 4, es decir:

$$ExtraBit = \left\lfloor \frac{g + 8s_m - 1}{4} \right\rfloor \tag{2}$$

Concluimos pues que el tiempo de transmisión para las tramas con identificadores de once y veintinueve bits serían respectivamente:

$$C_m = (55 + 10s_m)\tau_{bit} \tag{3}$$

$$C_m = (80 + 10s_m)\tau_{bit} \tag{4}$$

4.2. Parámetros del modelo

A continuación definiremos los parámetros a usar para nuestro modelo:

1. m: La prioridad del mensaje
2. J_m: El tiempo que se tarda en poner en cola el mensaje.
3. T_m: El periodo con el que se envía el mensaje
4. D_m: El retardo máximo permitido.
5. R_m: El tiempo máximo que transcurre desde que se inicia el evento que origina el mensaje hasta que se envía. Se dirá que un mensaje se envía a tiempo real si ($R_m \leq D_m$).

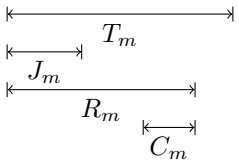

Figura 5: Modelo de tiempos de CAN

5. Análisis de Tindell

A continuación veremos cómo se resuelve el problema de calcular el retardo más extenso que podría experimentar un mensaje en una red CAN. Se encuentran tres elementos principales:

1. J_m, el tiempo máximo que transcurriría desde que se genera el mensaje hasta que éste entra en la cola.
2. w_n, el retardo de cola.
3. C_m, el tiempo de transmisión.

Siendo esto así, el tiempo de respuesta será:

$$R_m = J_m + w_m + C_m \tag{5}$$

El concepto de w_n, o tiempo de cola, engloba el lapso desde que el mensaje esta listo para ser enviado hasta que se obtienen los permisos para ello, pues estamos en un medio compartido. En el caso del nodo con mayor prioridad solo podría verse retrasado porque exista un nodo de menor prioridad que haya empezado a enviar su contenido antes de que el de alta prioridad se generase y enviase. En este caso se produciría un retardo de cola.

$$B_m = \max_{k \in lp(m)} (C_k) \tag{6}$$

En este párrafo se introducirá el concepto de periodo de ocupación como el tiempo que empieza en t_s cuando el mensaje de nuestro dispositivo tiene la mayor prioridad, y acaba en el instante t_e, cuando acaba la trasmisión.

Por lo tanto Tindell presenta la siguiente fórmula para el peor caso de espera de cola:

$$w_n = B_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_n + J_k + \tau_{bit}}{T_k} \right\rceil C_K \tag{7}$$

El fallo en la ecuación (7) es que asume que D_m es menor T_m, es decir, que el tiempo máximo que puede tardar en emitir el mensaje es siempre menor que periodo de envío del mensaje.

6. Análisis corregido

Podemos observar que con la especificación de CAN de prioridad fija y sin multitarea apropiativa el periodo de ocupación del canal se extenderá más allá de T_m, por lo que tras aplicar la corrección el nuevo periodo de ocupación, (t_m) será:

$$t_m^{n+1} = B_m + \sum_{k \in hp(m) \cup m} \left\lceil \frac{t_m^n + J_k}{T_k} \right\rceil C_K \tag{8}$$

Donde $\forall hp(m) \cup m$ representa todos los mensaje con prioridad m o mayor. La serie anterior converge si la utilización del bus es menor a 1:

$$U_m = \sum_{\forall k \in hp(m) \cup m} \frac{C_k}{T_k} \tag{9}$$

Podemos ver que en este caso, si:

1. $t_m \leq T_m - J_m$: Entonces la ecuación de Tindell nos serviría para calcular el retardo máximo.
2. $t_m > T_m - J_m$: Habría que usar la nueva ecuación.

6.1. Cálculo para varias instancias en cola

El número de mensajes en cola que se generan en un tiempo de ocupación del canal viene dado por:

$$Q_m = \left\lceil \frac{t_m + J_k}{T_k} \right\rceil \tag{10}$$

Para determinar el peor tiempo de respuesta del mensaje m es necesario calcular el peor tiempo de respuesta para los Q_m mensajes en cola. En la siguiente

ecuación se presenta el valor 1 como la variable que recorre los mensajes desde 0 a $Q_m - 1$.

$$w_m^{n+1}(q) = B_m + qC_m + \sum_{k \in hp(m)} \left\lceil \frac{w_n^m + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{11}$$

y por lo tanto su tiempo de respuesta R_m es:

$$R_m(q) = J_m + w_m(q) - qT_m + C_m \tag{12}$$

El cual será peor cuando:

$$R_m = \max_{q=0..Q_{m-1}} R_m(q) \tag{13}$$

7. Referencias

[1] - Introduction to the Controller Area Network (CAN) - Texas Instruments. Application Report: SLOA101A–August 2002–Revised July 2008

[2] - Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. By Robert I. Davis · Alan Burns · Reinder J. Bril · Johan J. Lukkien. Published online: 30 January 2007

[3] - Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. By Robert I. Davis and Alan Burns Real-Time Systems Research Group, Department of Computer Science, University of York, YO10 5DD, York (UK) , Reinder J. Bril and Johan J. Lukkien Technische Universiteit Eindhoven (TU/e), Den Dolech 2, 5600 AZ Eindhoven, The Netherlands.